



# IEEE Standard for Information Technology—Portable Operating System Interface (POSIX™)

## Base Specifications, Issue 8

IEEE Computer Society

and

The Open Group



Developed by the  
Microprocessor Standards Committee

**IEEE Std 1003.1™-2024**  
(Revision of IEEE Std 1003.1-2017)

The Open Group Standard Base Specifications, Issue 8

# **IEEE Standard for Information Technology—Portable Operating System Interface (POSIX™)**

## **Base Specifications, Issue 8**

Developed by the

**Microprocessor Committee**  
of the  
**IEEE Computer Society**

and

**The Open Group**

Approved 20 May 2024

**IEEE SA Standards Board**

**Abstract:** POSIX.1-2024 is simultaneously IEEE Std 1003.1™-2024 and The Open Group Standard Base Specifications, Issue 8.

POSIX.1-2024 defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. POSIX.1-2024 is intended to be used by both application developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of POSIX.1-2024 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

---

The Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York, NY 10016-5997, USA

The Open Group  
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, UK

Copyright © 2024 by The Institute of Electrical and Electronics Engineers, Inc. and The Open Group  
All rights reserved.

Published 14 June 2024 by IEEE in the United States of America.

PDF: ISBN 979-8-8557-0793-9 STD26978  
Print: ISBN 979-8-8557-0794-6 STDPD26978

Published 14 June 2024 by The Open Group in the United Kingdom

Doc. Number: C243  
ISBN: 1-957866-40-6

IEEE is a registered trademark in the U.S. Patent & Trademark Office and POSIX is a trademark owned by The Institute of Electrical and Electronics Engineers, Incorporated.

*This release of this standard is dedicated to the memory of Jörg Schilling and Donn Terry.*

*This standard has been prepared by the Austin Group. Feedback relating to the material contained within this standard may be submitted by using the Austin Group web site at [www.opengroup.org/austin/defectform.html](http://www.opengroup.org/austin/defectform.html).*

*IEEE prohibits discrimination, harassment, and bullying.*

*For more information, visit <https://www.ieee.org/about/corporate/governance/p9-26.html>.*

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher. Permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests should be sent by email to [austin-group-permissions@opengroup.org](mailto:austin-group-permissions@opengroup.org).*

The following areas are outside the scope of POSIX.1-2024:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-2024 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

**Keywords:** application program interface (API), argument, asynchronous, basic regular expression (BRE), built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, IEEE 1003.1™, input/output (I/O), job control, network, parent, portable operating system interface (POSIX™), shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

## The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards and open standards by fostering a culture of collaboration, inclusivity, and mutual respect among our diverse membership of more than 900 organizations. Our membership includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The mission of The Open Group is to drive the creation of Boundaryless Information Flow™ achieved by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

Further information on The Open Group is available at <https://www.opengroup.org>.

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at <https://www.opengroup.org/library>.

## Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (<https://standards.ieee.org/ipr/disclaimers.html>), appear in all IEEE standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

### Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers involved in technical working groups are not necessarily members of IEEE or IEEE SA and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning all standards, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. IEEE Standards documents do not guarantee safety, security, health, or environmental protection, or compliance with law, or guarantee against interference with or from other devices or networks. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document should rely upon their own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

## Translations

The IEEE consensus balloting process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English language version published by IEEE is the approved IEEE standard.

## Use by artificial intelligence systems

In no event shall material in this document be used for the purpose of creating, training, enhancing, developing, maintaining, or contributing to any artificial intelligence systems without the express, written consent of IEEE SA and The Open Group in advance. “Artificial intelligence” refers to any software, application, or other system that uses artificial intelligence, machine learning, or similar technologies, to analyze, train, process, or generate content. Requests for consent can be submitted by email to [austin-group-permissions@opengroup.org](mailto:austin-group-permissions@opengroup.org).

## Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual is not, and shall not be considered or inferred to be, the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE or IEEE SA. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter’s views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s). News releases about IEEE standards issued by entities other than IEEE SA should be considered the view of the entity issuing the release rather than the formal position of IEEE or IEEE SA.

## Comments on standards

Feedback relating to the material contained within this standard may be submitted by using the Austin Group web site at <http://www.opengroup.org/austin/defectform.html>.

## Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

## Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

## Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, neither IEEE nor its licensors waive any rights in copyright to the documents.

## Photocopies

Subject to payment of the appropriate licensing fees, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400; <https://www.copyright.com/>. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

## Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit [IEEE Xplore](#) or [contact IEEE](#).<sup>1</sup> For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

## Errata

Errata, if any, for all IEEE standards can be accessed on the [IEEE SA Website](#).<sup>2</sup> Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in [IEEE Xplore](#). Users are encouraged to periodically check for errata.

---

<sup>1</sup> Available at: <https://ieeexplore.ieee.org/browse/standards/collection/ieee>.

<sup>2</sup> Available at: <https://standards.ieee.org/standard/index.html>.

## Patents

IEEE standards are developed in compliance with the [IEEE SA Patent Policy](#).<sup>3</sup>

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at <https://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

## IMPORTANT NOTICE

Technologies, application of technologies, and recommended procedures in various industries evolve over time. The IEEE standards development process allows participants to review developments in industries, technologies, and practices, and to determine what, if any, updates should be made to the IEEE standard. During this evolution, the technologies and recommendations in IEEE standards may be implemented in ways not foreseen during the standard's development. IEEE standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, data privacy, and interference protection practices and all applicable laws and regulations.

---

<sup>3</sup> Available at: <https://standards.ieee.org/about/sasb/patcom/materials.html>.



## Participants

IEEE Std 1003.1™-2024 was prepared by the Austin Group, sponsored by the Microprocessor Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22.

### The Austin Group

At the time this IEEE standard was completed, the Austin Group had the following membership:

**Andrew Josey**, Chair  
**Donald W. Cragun**, Organizational Representative, IEEE MSC  
**Nicholas M. Stoughton**, Organizational Representative, ISO/IEC JTC 1/SC22  
**Eric Blake**, Organizational Representative, The Open Group  
**Cathy Fox, Geoff Clare**, Technical Editors

### Austin Group Technical Reviewers

William Ahern	Dmitry Goncharov	Quentin Rameau
Mohamed Akram	Christopher M. Graff	Martin Řehák
Joe Auricchio	Quinn Grier	Torvald Riegel
Ori Avtalion	Philip Guenther	G. Branden Robinson
Bogdan Barbu	Bruno Haible	Xavier Roche
Steve Bartolomei	Richard Hansen	Bastien Roucaries
Petr Baudis	Guy Harris	Daniel Sabogal
Fabrice Bauzac	Mark Harris	Askar Safin
Eric Blake	Gavin Howard	Jörg Schilling
Mark S. Brown	Elliott Hughes	Ed Schouten
Erik Cederstrand	Roland Illig	Konrad Schwarz
Stéphane Chazelas	Jarmo Jaakkola	Ingo Schwarze
Scott Cheloha	Andrew Josey	Martin Sebor
Alexander Cherepanov	Nickolas Raymond Kaczynski	Olaf 'Rhialto' Seibert
Geoff Clare	Nate Karstens	Joel Sherrill
Robert Clausecker	Michael Kerrisk	Curtis Smith
Daniel Colascione	Alexey Khoroshilov	Paul Smith
Garrett Cooper	Elad Lahav	Job Snijders
Alan Coopersmith	Jeff Layton	Oliver Soong
Ralph Corderoy	Vincent Lefèvre	Dimitri Staessens
Ciprian Dorin Craciun	Mark Lundblad	Nicholas M. Stoughton
Donald W. Cragun	Roger Marquis	Sören Tempel
Mike Crowe	Nikos Mavrogiannopoulos	Jilles Tjoelker
Martijn Dekker	Davin McCall	William Toth
Andrés Delfino	Mihail Mihaylov	Fred J. Tydeman
D.J. Delorie	Todd C. Miller	Stijn van Dronrgelen
Matthew Dempsky	Christoph Anton Mitterer	Lawrence Velázquez
Antonio Diaz	Mihai Moldovan	Evgeny Vereshchagin
Ulrich Drepper	Ed Morton	Rasmus Villoeoes
Paul Eggert	Joseph S. Myers	Dennis Wölfing
Robert Elz	Szabolcs Nagy	Jonathan Wakely
Steve Emmerson	Jonathan Nieder	Colin Watson
Laszlo Ersek	Danny Niu	Nathan Weeks
Andras Farkas	Steffen Nurpmeso	Florian Weimer
Richard Felker	Richard Palethorpe	Zack Weinberg
Dirk Fieldhouse	Daniele Palumbo	David A. Wheeler
Mike Frysinger	Isabella Parakiss	Nicolas Williams
Mark Galeck	Ben Pfaff	Yousong Zhou
Enrique Garcia	J. William Piggott	Mark Ziegast
Thorsten Glaser	Wayne Pollock	Roman Žilka

## Austin Group Working Group Members

Hans Aberg	Jan Hafer	Chet Ramey
Eric Ackermann	Bruno Haible	Gabriel Ravier
Godmar Back	Richard Hansen	G. Branden Robinson
Eric Blake	Mark Harris	Eric Sanchis
Volodymyr Boyko	David Holland	Daniel Santos
Andries E. Brouwer	Gavin Howard	Jörg Schilling
Mark S. Brown	Elliott Hughes	Ed Schouten
Jefferson Carpenter	Roland Illig	Konrad Schwarz
Olivier Certner	Lennart Jablonka	Ingo Schwarze
Stéphane Chazelas	Chris F.A. Johns	John Scott
Tom Cherry	Darrin Johnson	Simon Ser
Earl Chew	Andrew Josey	Joel Sherrill
Geoff Clare	Nate Karstens	Thor Lancelot Simon
Joshua M. Clulow	Dan Kegel	Keld Simonsen
Alan Coopersmith	Michael Kerrisk	Paul Smith
Donald W. Cragun	Anton Khikhlikha	Job Snijders
Mike Crowe	Ukko Koknevičs	Gabriel Soldani
Martijn Dekker	Bruce Korb	Oliver Soong
Matthew Dempsky	David Korn	Dimitri Staessens
Drew DeVault	Rob Landley	Marc J. Stephenson
Casper Dik	Vincent Lefèvre	Nicholas M. Stoughton
Deepa Dinamani	Wojtek Lerch	Oskar Sveinsen
Dan Douglas	Charlie Lin	Alfred M. Szmídt
Niall Douglas	Scott Lurndal	Tapani Tarvainen
Ulrich Drepper	Roger Marquis	Alexander Terekhov
Lawrence D.K.B. Dwyer	Davin McCall	Donn Terry
Paul Eggert	Stephen Michell	Jilles Tjoelker
Daniel Eischen	Per Mildner	Fred J. Tydeman
Julian Elischer	Christoph Anton Mitterer	Oğuz Uysal
Robert Elz	Thomas Mueller	Harald van Dijk
Bruce Evans	Wilhelm Mueller	Lawrence Velázquez
Richard Felker	Koichi Murase	Oleksii Vilchansk
Jeffrey K. Fellin	Joseph S. Myers	Corinna Vinschen
Dirk Fieldhouse	Danny Niu	Jonathan Wakely
Hal Finkel	Gian Ntzik	L.A. Walsh
Michael Forney	Steffen Nurpmeso	David A. Wheeler
Mike Frysinger	Carlos O'Donell	Jakub Wilk
Mark Galeck	Andrew Pennebaker	Dennis Wölfing
Thorsten Glaser	Steven Penny	Garrett Wollman
Andreas Grapentin	Colin Percival	Jörg Wunsch
Michael Greenberg	J. William Piggott	Ryan Zezeski
Philip Guenther	Wayne Pollock	Mark Ziegast
Joseph M. Gwinn	Quentin Rameau	Jason Zions

## The Open Group

When The Open Group approved the Base Specifications, Issue 8, (technically identical to this standard) on 21 March 2024, the membership of The Open Group Base Working Group was as follows:

**Andrew Josey**, Chair  
**Eric Blake**, Austin Group Liaison  
**Cathy Fox, Geoff Clare**, Technical Editor

## Base Working Group Members

Joe Auricchio	Geoff Clare	Andrew Josey
Eric Blake	Donald W. Cragun	Mark Ziegast

## IEEE

At the time this standard was completed, the Microprocessor Committee had the following membership:

**Ralph Baker Kearfott**, *Chair*  
**Leonard Tsai**, *Vice Chair and P754 Chair*  
**Andrew Josey**, *P1003.1 Chair*  
**Donald W. Cragun**, *Austin Group Liaison*  
**Joseph M. Gwinn**, *Ex-officio Emeritus*  
**Richard Bugg**, *P1722.1 Chair*  
**Kiran Gunnam**, *P3109 Chair*  
**David Hough**, *Outgoing P754 Chair*  
**Dave Olsen**, *P1722 Chair*  
**Nathalie Revol**, *P1788 Chair*  
**Blaise Vignon**, *P3109 Chair*

The following members of the individual Standards Association balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Boon Chong Ang  
Steven Bezner  
Diego Chiozzi  
Donald W. Cragun  
Andrew Fieldsend  
David Fuschi

Jie Guan  
Joseph M. Gwinn  
Werner Hoelzl  
Andrew Josey  
Piotr Karocki  
Kenneth Lang

Rajesh Murthy  
Venkatesha Prasad  
Stephen Schwarm  
Walter Struppler  
Oren Yuen  
Janusz Zalewski

When the IEEE SA Standards Board approved this standard on 20 May 2024, it had the following membership:

**David J. Law**, *Chair*  
**Jon Walter Rosdahl**, *Vice Chair*  
**Gary Hoffman**, *Past Chair*  
**Alpesh Shah**, *Secretary*

Sara R. Biyabani  
Ted Burse  
Stephen Dukes  
Doug Edwards  
J. Travis Griffith  
Guido R. Hiertz  
Ronald W. Hotchkiss

Hao Hu  
Yousef Kimiagar  
Joseph L. Koepfinger\*  
Howard Li  
Xiaohui Liu  
John Haiying Lu  
Kevin W. Lu  
Hiroshi Mano

Paul Nikolich  
Robby Robson  
Lei Wang  
F. Keith Waters  
Sha Wei  
Philip B. Winston  
Don Wright

\*Member Emeritus

## Introduction

This introduction is not part of IEEE Std 1003.1™-2024, IEEE Standard for Information Technology—Portable Operating System Interface (POSIX™)—Base Specifications, Issue 8.

This draft standard was developed, and is maintained, by a joint working group of members of the IEEE Microprocessor Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.<sup>4</sup>

The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group is to develop and maintain the core open systems interfaces that are the POSIX 1003.1 (and former 1003.2) standards, ISO/IEC 9945, and the core of the Single UNIX® Specification.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group Standard designation, and an ISO/IEC designation.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see [www.opengroup.org/austin](http://www.opengroup.org/austin).

## Background

The developers of POSIX.1-2024 represent a cross-section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, POSIX.1-2024 describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application developers to write portable applications – it was developed with that goal in mind – it has been designated POSIX,<sup>5</sup> an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE Standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

<sup>4</sup> The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

<sup>5</sup> The name POSIX was suggested by Richard Stallman. It is expected to be pronounced with the first two syllables as in positive, not poh-six, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

## Audience

The intended audience for POSIX.1-2024 is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

- Persons buying hardware and software systems
- Persons managing companies that are deciding on future corporate computing directions
- Persons implementing operating systems, and especially
- Persons developing applications where portability is an objective

## Purpose

Several principles guided the development of POSIX.1-2024:

- **Application-Oriented** – The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. POSIX.1-2024 codifies the common, existing definition of the UNIX system.
- **Interface, Not Implementation** – POSIX.1-2024 defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.
- **Source, Not Object, Portability** – POSIX.1-2024 has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. POSIX.1-2024 does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.
- **The C Language** – The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.
- **No Superuser, No System Administration** – There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in POSIX.1-2024. POSIX.1-2024 is also not concerned with hardware constraints or system maintenance.
- **Minimal Interface, Minimally Defined** – In keeping with the historical design principles of the UNIX system, the mandatory core facilities of POSIX.1-2024 have been kept as minimal as possible. Additional capabilities have been added as optional extensions.
- **Broadly Implementable** – The developers of POSIX.1-2024 endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:
  - All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
  - Compatible systems that are not derived from the original UNIX system code
  - Emulations hosted on entirely different operating systems
  - Networked systems
  - Distributed systems

- Systems running on a broad range of hardware

No direct references to this goal appear in POSIX.1-2024, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations – When the original version – IEEE Std 1003.1-1988 – was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various The Open Group (formerly X/Open) specifications, and IEEE Std 1003.1-2001 and its technical corrigenda have consolidated this consensus, and this version reflects the significantly increased level of consensus arrived at since the original versions. The authors of the original versions tried, as much as possible, to follow the principles below when creating new specifications:

- By standardizing an interface like one in an historical implementation; for example, directories
- By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended tar format defined in the pax utility
- By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the sigaction() function

POSIX.1-2024 is specifically not a codification of a particular vendor’s product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of POSIX.1-2024. These need to be used with careful management to be able to adapt to future extensions of POSIX.1-2024 and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code—A goal of POSIX.1-2024 was to minimize additional work for application developers. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

## POSIX.1-2024

POSIX.1-2024 defines the Portable Operating System Interface (POSIX) requirements and consists of the following topics arranged as a series of volumes within the standard:

- Base Definitions
- System Interfaces
- Shell and Utilities
- Rationale (Informative)

## Base Definitions

The Base Definitions volume provides common definitions for this standard, therefore readers should be familiar with it before using the other volumes.

This volume is structured as follows:

- Chapter 1 is an introduction.
- Chapter 2 defines the conformance requirements.
- Chapter 3 defines general terms used.
- Chapter 4 describes general concepts used.
- Chapter 5 describes the notation used to specify file input and output formats in this volume and the Shell and Utilities volume.
- Chapter 6 describes the portable character set and the process of character set definition.
- Chapter 7 describes the syntax for defining internationalization locales as well as the POSIX locale provided on all systems.
- Chapter 8 describes the use of environment variables for internationalization and other purposes.
- Chapter 9 describes the syntax of pattern matching using regular expressions employed by many utilities and matched by the *regcomp()* and *regexexec()* functions.
- Chapter 10 describes files and devices found on all systems.
- Chapter 11 describes the asynchronous terminal interface for many of the functions in the System Interfaces volume and the *stty* utility in the Shell and Utilities volume.
- Chapter 12 describes the policies for command line argument construction and parsing.
- Chapter 13 describes namespace reservation.
- Chapter 14 defines the contents of headers which declare the functions and global variables, and define types, constants, macros, and data structures that are needed by programs using the services provided by the System Interfaces volume.

Comprehensive references are available in the Index.

## System Interfaces

The System Interfaces volume describes the interfaces offered to application programs by POSIX-conformant systems. Readers are expected to be experienced C language programmers, and to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards.
- Chapter 2 contains important concepts, terms, and caveats relating to the rest of this volume.
- Chapter 3 defines the functional interfaces to the POSIX-conformant system.

Comprehensive references are available in the Index.

## Shell and Utilities

The Shell and Utilities volume describes the commands and utilities offered to application programs on POSIX-conformant systems. Readers are expected to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards. It also describes the defaults used by the utility descriptions.
- Chapter 2 describes the command language used in POSIX-conformant systems, and special built-in utilities.
- Chapter 3 consists of reference pages for all utilities, other than the special built-in utilities described in Chapter 2, available on POSIX-conformant systems.

Comprehensive references are available in the Index.

### Rationale (Informative)

The Rationale volume is published to assist in the process of review. It contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects of POSIX.1-2024 that may not be immediately apparent.

This volume is organized in parallel to the normative volumes of this standard, with a separate part for each of the three normative volumes.

Within this volume, the following terms are used:

- Base standard—The portions of POSIX.1-2024 that are not optional, equivalent to the definitions of classic POSIX.1 and POSIX.2.
- POSIX.0—Although this term is not used in the normative text of POSIX.1-2024, it is used in this volume to refer to IEEE Std 1003.0-1995.
- POSIX.1b—Although this term is not used in the normative text of POSIX.1-2024, it is used in this volume to refer to the elements of the POSIX Realtime Extension amendment. (This was earlier referred to as POSIX.4 during the standard development process.)
- POSIX.1c—Although this term is not used in the normative text of POSIX.1-2024, it is used in this volume to refer to the POSIX Threads Extension amendment. (This was earlier referred to as POSIX.4a during the standard development process.)
- Standard developers—The individuals and companies in the development organizations responsible for POSIX.1-2024: the IEEE P1003.1 working groups, The Open Group Base working group, advised by the hundreds of individual technical experts who balloted the draft standards within the Austin Group, and the member bodies and technical experts of ISO/IEC JTC 1/SC 22.
- XSI option—The portions of POSIX.1-2024 addressing the extension added for support of the Single UNIX Specification.

### Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as POSIX.1-2024, which is technically identical to The Open Group Base Specifications, Issue 8.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in POSIX.1-2024.

Reference	Example	Notes
C-Language Data Structure	<code>aiocb</code>	



Reference	Example	Notes
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	<b>long</b>	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	
C-Language Function Argument	<i>arg</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	< <b>sys/stat.h</b> >	
C-Language Keyword	<b>return</b>	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	NET_ADDRSTRLEN	
C-Language Preprocessing Directive	<b>#define</b>	
Commands within a Utility	<b>a, c</b>	
Conversion Specifier, Specifier/Modifier Character	%A, g, E	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	<b>Hello, World</b>	
Filename	<b>/tmp</b>	
Literal Character	'c', '\r'	2
Literal String	"abcde"	2
Optional Items in Utility Syntax	[ ]	
Parameter	< <i>directory pathname</i> >	
Special Character	<newline>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	
User Input and Example Code	echo Hello, World	5
Utility Name	<i>awk</i>	
Utility Operand	<i>file_name</i>	
Utility Option	<b>-c</b>	
Utility Option with Option-Argument	<b>-w width</b>	

Note that:

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf()* and *fscanf()* formatting functions.

2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. The literal characters <apostrophe> (also known as single-quote) and <backslash> are either shown as the C constants ' \ ' and ' \\ ', respectively, or as the special characters <apostrophe>, single-quote, and <backslash> depending on context.
3. The style selected for some of the special characters, such as <newline>, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters <tab> or <newline>.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct, or obtained at runtime from functions or utilities that return limit or configuration values.
5. Brackets shown in this font, "[ ]", are part of the syntax and do not indicate optional items. In syntax the '| ' symbol is used to separate alternatives, and ellipses (" . . . ") are used to show that additional arguments are optional.

Shading is used to identify extensions and options.

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

Ranges of values are indicated with parentheses or brackets as follows:

1.  $(a,b)$  means the range of all values from  $a$  to  $b$ , including neither  $a$  nor  $b$ .
2.  $[a,b]$  means the range of all values from  $a$  to  $b$ , including  $a$  and  $b$ .
3.  $[a,b)$  means the range of all values from  $a$  to  $b$ , including  $a$ , but not  $b$ .
4.  $(a,b]$  means the range of all values from  $a$  to  $b$ , including  $b$ , but not  $a$ .

NOTE— A symbolic limit beginning with POSIX is treated differently, depending on context. In a C-language header, the symbol POSIXstring (where string may contain underscores) is represented by the C identifier \_POSIXstring, with a leading underscore required to prevent ISO C standard name space pollution. However, in other contexts, such as languages other than C, the leading underscore is not used because this requirement does not exist.

# Contents

<b>Volume</b>	<b>1</b>	<b>Base Definitions, Issue 8.....</b>	<b>1</b>
<b>Chapter</b>	<b>1</b>	<b>Introduction.....</b>	<b>3</b>
	1.1	Scope .....	3
	1.2	Word Usage.....	4
	1.3	Conformance.....	4
	1.4	Normative References .....	5
	1.5	Change History .....	5
	1.6	Terminology .....	5
	1.7	Definitions and Concepts.....	6
	1.8	Portability.....	7
	1.8.1	Codes .....	7
	1.8.2	Margin Code Notation .....	12
<b>Chapter</b>	<b>2</b>	<b>Conformance.....</b>	<b>15</b>
	2.1	Implementation Conformance .....	15
	2.1.1	Requirements .....	15
	2.1.2	Documentation .....	16
	2.1.3	POSIX Conformance.....	17
	2.1.4	XSI Conformance .....	19
	2.1.5	Option Groups.....	20
	2.1.6	Options .....	25
	2.2	Application Conformance.....	27
	2.2.1	Strictly Conforming POSIX Application.....	27
	2.2.2	Conforming POSIX Application .....	28
	2.2.3	Conforming POSIX Application Using Extensions.....	28
	2.2.4	Strictly Conforming XSI Application .....	29
	2.2.5	Conforming XSI Application Using Extensions .....	29
	2.3	Language-Dependent Services for the C Programming Language .....	29
	2.4	Other Language-Related Specifications.....	29
<b>Chapter</b>	<b>3</b>	<b>Definitions.....</b>	<b>31</b>
	3.1	Abortive Release.....	31
	3.2	Absolute Pathname.....	31
	3.3	Access Mode .....	31
	3.4	Additional File Access Control Mechanism.....	31
	3.5	Address Space.....	31
	3.6	Advisory Information.....	31
	3.7	Affirmative Response .....	32
	3.8	Alert .....	32
	3.9	Alert Character (<alert>).....	32
	3.10	Alias Name.....	32
	3.11	Alignment .....	32

3.12	Alternate File Access Control Mechanism .....	32
3.13	Alternate Signal Stack.....	33
3.14	Ancillary Data.....	33
3.15	Angle Brackets.....	33
3.16	Anonymous Memory Object.....	33
3.17	Apostrophe Character (<apostrophe>).....	33
3.18	Application.....	33
3.19	Application Address.....	33
3.20	Application Program Interface (API).....	33
3.21	Appropriate Privileges .....	34
3.22	Argument .....	34
3.23	Arm (a Timer) .....	34
3.24	Asterisk Character (<asterisk>) .....	34
3.25	Async-Cancel-Safe Function.....	34
3.26	Asynchronous Events.....	34
3.27	Asynchronous Input and Output .....	34
3.28	Async-Signal-Safe Function.....	35
3.29	Asynchronously-Generated Signal.....	35
3.30	Asynchronous I/O Completion.....	35
3.31	Asynchronous I/O Operation.....	35
3.32	Atomic Operation .....	35
3.33	Authentication.....	35
3.34	Authorization .....	36
3.35	Background Job .....	36
3.36	Background Process.....	36
3.37	Background Process Group .....	36
3.38	Backquote Character.....	36
3.39	Backslash Character (<backslash>) .....	36
3.40	Backspace Character (<backspace>) .....	37
3.41	Barrier .....	37
3.42	Basename.....	37
3.43	Basic Regular Expression (BRE).....	37
3.44	Bind .....	37
3.45	Blank Character (<blank>).....	37
3.46	Blank Line.....	37
3.47	Blocked Process (or Thread) .....	37
3.48	Blocking.....	38
3.49	Block-Mode Terminal .....	38
3.50	Block Special File.....	38
3.51	Braces .....	38
3.52	Brackets.....	38
3.53	Broadcast .....	38
3.54	Built-In Utility (or Built-In).....	39
3.55	Byte.....	39
3.56	Byte Input/Output Functions.....	39
3.57	Carriage-Return Character (<carriage-return>) .....	39
3.58	Character .....	39
3.59	Character Array.....	40
3.60	Character Class.....	40
3.61	Character Set.....	40
3.62	Character Special File .....	40
3.63	Character String.....	40

3.64	Child Process .....	40
3.65	Circumflex Character (<circumflex>).....	40
3.66	Clock .....	41
3.67	Clock Jump.....	41
3.68	Clock Tick.....	41
3.69	Code Block .....	41
3.70	Coded Character Set .....	41
3.71	Codeset .....	41
3.72	Collating Element.....	41
3.73	Collation .....	42
3.74	Collation Sequence.....	42
3.75	Column Position.....	42
3.76	Command.....	42
3.77	Command Language Interpreter .....	42
3.78	Composite Graphic Symbol.....	43
3.79	Condition Variable .....	43
3.80	Connected Socket .....	43
3.81	Connection .....	43
3.82	Connection Mode.....	43
3.83	Connectionless Mode .....	43
3.84	Control Character.....	43
3.85	Control Operator .....	44
3.86	Controlling Process.....	44
3.87	Controlling Terminal .....	44
3.88	Conversion Descriptor .....	44
3.89	Core Image .....	44
3.90	CPU Time (Execution Time) .....	44
3.91	CPU-Time Clock.....	44
3.92	CPU-Time Timer.....	45
3.93	Current Job .....	45
3.94	Current Working Directory.....	45
3.95	Cursor Position.....	45
3.96	Datagram.....	45
3.97	Data Race.....	45
3.98	Data Segment.....	45
3.99	Decimal-Point Character .....	45
3.100	Declaration Utility.....	45
3.101	Device .....	46
3.102	Device ID .....	46
3.103	Directory .....	46
3.104	Directory Entry (or Hard Link).....	46
3.105	Directory Stream .....	46
3.106	Disarm (a Timer) .....	46
3.107	Display .....	46
3.108	Display Line.....	46
3.109	Dollar-Sign Character (<dollar-sign>) .....	46
3.110	Dot.....	47
3.111	Dot-Dot.....	47
3.112	Dot-Po File.....	47
3.113	Double-Quote Character .....	47
3.114	Downshifting .....	47
3.115	Driver .....	47

3.116	Effective Group ID .....	47
3.117	Effective User ID .....	47
3.118	Eight-Bit Transparency .....	48
3.119	Empty Directory .....	48
3.120	Empty Line .....	48
3.121	Empty String (or Null String) .....	48
3.122	Empty Wide-Character String .....	48
3.123	Encoding Rule .....	48
3.124	Entire Regular Expression .....	48
3.125	Epoch .....	48
3.126	Equivalence Class .....	49
3.127	Era .....	49
3.128	Event Management .....	49
3.129	Executable File .....	49
3.130	Execute .....	49
3.131	Execution Time .....	49
3.132	Execution Time Monitoring .....	49
3.133	Expand .....	50
3.134	Extended Regular Expression (ERE) .....	50
3.135	Extended Security Controls .....	50
3.136	Feature Test Macro .....	50
3.137	Field .....	50
3.138	FIFO Special File (or FIFO) .....	51
3.139	File .....	51
3.140	File Description .....	51
3.141	File Descriptor .....	51
3.142	File Group Class .....	51
3.143	File Lock .....	51
3.144	File Mode .....	52
3.145	File Mode Bits .....	52
3.146	Filename .....	52
3.147	Filename String .....	52
3.148	File Offset .....	52
3.149	File Other Class .....	52
3.150	File Owner Class .....	52
3.151	File Permission Bits .....	53
3.152	File Serial Number .....	53
3.153	File System .....	53
3.154	File Type .....	53
3.155	Filter .....	53
3.156	First Open (of a File) .....	53
3.157	Flow Control .....	53
3.158	Foreground Job .....	54
3.159	Foreground Process .....	54
3.160	Foreground Process Group .....	54
3.161	Foreground Process Group ID .....	54
3.162	Form-Feed Character (<form-feed>) .....	54
3.163	Graphic Character .....	54
3.164	Group Database .....	55
3.165	Group ID .....	55
3.166	Group Name .....	55
3.167	Hard Limit .....	55

3.168	Hard Link .....	55
3.169	Hole .....	55
3.170	Home Directory .....	56
3.171	Host Byte Order .....	56
3.172	Incomplete Line .....	56
3.173	Inf .....	56
3.174	Interactive Device .....	56
3.175	Interactive Shell .....	56
3.176	Internationalization .....	56
3.177	Interprocess Communication .....	56
3.178	Intrinsic Utility .....	57
3.179	Invoke .....	57
3.180	Job .....	57
3.181	Job Control .....	57
3.182	Job ID .....	57
3.183	Joinable Thread .....	58
3.184	Last Close (of a File) .....	58
3.185	Line .....	58
3.186	Linger .....	58
3.187	Link .....	58
3.188	Link Count .....	58
3.189	Live Process .....	59
3.190	Live Thread .....	59
3.191	Local Customs .....	59
3.192	Local Interprocess Communication (Local IPC) .....	59
3.193	Locale .....	59
3.194	Localization .....	59
3.195	Lock-Free Operation .....	59
3.196	Login .....	60
3.197	Login Name .....	60
3.198	Map .....	60
3.199	Matched .....	60
3.200	Memory Mapped Files .....	60
3.201	Memory Object .....	60
3.202	Memory-Resident .....	60
3.203	Message .....	61
3.204	Message Catalog .....	61
3.205	Message Catalog Descriptor .....	61
3.206	Message Queue .....	61
3.207	Messages Object .....	61
3.208	Mode .....	61
3.209	Monotonic Clock .....	61
3.210	Mount Point .....	62
3.211	Multi-Character Collating Element .....	62
3.212	Multi-Threaded Library .....	62
3.213	Multi-Threaded Process .....	62
3.214	Multi-Threaded Program .....	62
3.215	Mutex .....	62
3.216	Name .....	63
3.217	NaN (Not a Number) .....	63
3.218	Native Language .....	63
3.219	Negative .....	63

3.220	Negative Response.....	63
3.221	Network.....	63
3.222	Network Address.....	63
3.223	Network Byte Order.....	64
3.224	Newline Character (<newline>).....	64
3.225	Nice Value.....	64
3.226	Non-Blocking.....	64
3.227	Non-Spacing Characters.....	64
3.228	NUL.....	64
3.229	Null Byte.....	65
3.230	Null Pointer.....	65
3.231	Null String.....	65
3.232	Null Terminator.....	65
3.233	Null Wide-Character Code.....	65
3.234	Number-Sign Character (<number-sign>).....	65
3.235	Object File.....	65
3.236	Octet.....	65
3.237	OFD-Owned File Lock.....	66
3.238	Offset Maximum.....	66
3.239	Opaque Address.....	66
3.240	Open File.....	66
3.241	Open File Description.....	66
3.242	Operand.....	66
3.243	Operator.....	66
3.244	Option.....	67
3.245	Option-Argument.....	67
3.246	Orientation.....	67
3.247	Orphaned Process Group.....	67
3.248	Page.....	67
3.249	Page Size.....	67
3.250	Parameter.....	68
3.251	Parent Directory.....	68
3.252	Parent Process.....	68
3.253	Parent Process ID.....	68
3.254	Pathname.....	68
3.255	Pathname Component.....	69
3.256	Path Prefix.....	69
3.257	Pattern.....	69
3.258	Period Character (<period>).....	69
3.259	Permissions.....	69
3.260	Persistence.....	69
3.261	Pipe.....	70
3.262	Polling.....	70
3.263	Portable Character Set.....	70
3.264	Portable Filename.....	70
3.265	Portable Filename Character Set.....	70
3.266	Portable Messages Object Source File (or Dot-Po File).....	70
3.267	Positional Parameter.....	71
3.268	Positive.....	71
3.269	Preallocation.....	71
3.270	Preempted Process (or Thread).....	71



3.271	Previous Job .....	71
3.272	Printable Character .....	71
3.273	Printable File .....	71
3.274	Priority .....	72
3.275	Priority Inversion .....	72
3.276	Priority Scheduling .....	72
3.277	Priority-Based Scheduling .....	72
3.278	Privilege .....	72
3.279	Process .....	72
3.280	Process Group .....	72
3.281	Process Group ID .....	72
3.282	Process Group Leader .....	72
3.283	Process Group Lifetime .....	73
3.284	Process ID .....	73
3.285	Process Lifetime .....	73
3.286	Process Memory Locking .....	73
3.287	Process Termination .....	73
3.288	Process Virtual Time .....	74
3.289	Process-Owned File Lock .....	74
3.290	Process-To-Process Communication .....	74
3.291	Program .....	74
3.292	Protocol .....	74
3.293	Pseudo-Terminal .....	74
3.294	Radix Character (or Decimal-Point Character) .....	74
3.295	Read-Only File System .....	75
3.296	Read-Write Lock .....	75
3.297	Real Group ID .....	75
3.298	Real Time .....	75
3.299	Realtime Signal Extension .....	75
3.300	Real User ID .....	75
3.301	Record .....	75
3.302	Record Lock .....	76
3.303	Redirection .....	76
3.304	Redirection Operator .....	76
3.305	Referenced Shared Memory Object .....	76
3.306	Refresh .....	76
3.307	Regular Built-In Utility (or Regular Built-In) .....	76
3.308	Regular Expression .....	76
3.309	Region .....	76
3.310	Regular File .....	77
3.311	Relative Pathname .....	77
3.312	Relocatable File .....	77
3.313	Relocation .....	77
3.314	(Time) Resolution .....	77
3.315	Robust Mutex .....	77
3.316	Root Directory .....	77
3.317	Runnable Process (or Thread) .....	77
3.318	Running Process (or Thread) .....	77
3.319	Saved Resource Limits .....	78
3.320	Saved Set-Group-ID .....	78
3.321	Saved Set-User-ID .....	78
3.322	Scheduling .....	78

3.323	Scheduling Allocation Domain .....	78
3.324	Scheduling Contention Scope .....	78
3.325	Scheduling Policy .....	79
3.326	Screen .....	79
3.327	Scroll .....	79
3.328	Semaphore .....	79
3.329	Session .....	79
3.330	Session Leader .....	79
3.331	Session Lifetime .....	79
3.332	Shared Memory Object .....	80
3.333	Shell .....	80
3.334	Shell, the .....	80
3.335	Shell Script .....	80
3.336	Signal .....	80
3.337	Signal Stack .....	80
3.338	Single-Quote Character .....	80
3.339	Single-Threaded Process .....	80
3.340	Single-Threaded Program .....	81
3.341	Slash Character (<slash>) .....	81
3.342	Socket .....	81
3.343	Socket Address .....	81
3.344	Soft Limit .....	81
3.345	Source Code .....	81
3.346	Space Character (<space>) .....	82
3.347	Sparse File .....	82
3.348	Spawn .....	82
3.349	Special Built-In Utility (or Special Built-In) .....	82
3.350	Special Parameter .....	82
3.351	Spin Lock .....	82
3.352	Sporadic Server .....	82
3.353	Standard Error .....	82
3.354	Standard Input .....	83
3.355	Standard Output .....	83
3.356	Standard Utilities .....	83
3.357	Stream .....	83
3.358	String .....	83
3.359	Subshell .....	84
3.360	Successfully Transferred .....	84
3.361	Supplementary Group ID .....	84
3.362	Suspended Job .....	84
3.363	Symbolic Constant .....	84
3.364	Symbolic Link .....	85
3.365	Synchronization Operation .....	85
3.366	Synchronized Input and Output .....	85
3.367	Synchronized I/O Completion .....	85
3.368	Synchronized I/O Data Integrity Completion .....	85
3.369	Synchronized I/O File Integrity Completion .....	85
3.370	Synchronized I/O Operation .....	86
3.371	Synchronous I/O Operation .....	86
3.372	Synchronously-Generated Signal .....	86
3.373	System .....	86
3.374	System Boot .....	86

3.375	System Clock.....	86
3.376	System Console .....	86
3.377	System Crash .....	86
3.378	System Databases.....	87
3.379	System Documentation .....	87
3.380	System Process.....	87
3.381	System Reboot .....	87
3.382	System-Wide .....	87
3.383	Tab Character (<tab>).....	87
3.384	Terminal (or Terminal Device) .....	87
3.385	Text Column.....	87
3.386	Text Domain.....	88
3.387	Text File.....	88
3.388	Thread .....	88
3.389	Thread ID .....	88
3.390	Thread Lifetime .....	89
3.391	Thread List .....	89
3.392	Thread Termination .....	89
3.393	Thread-Safe .....	89
3.394	Thread-Specific Data Key.....	89
3.395	Tilde Character (<tilde>).....	90
3.396	Timeouts .....	90
3.397	Timer .....	90
3.398	Timer Overrun.....	90
3.399	Token.....	90
3.400	Typed Memory Name Space .....	90
3.401	Typed Memory Object.....	90
3.402	Typed Memory Pool .....	90
3.403	Typed Memory Port.....	91
3.404	Unbind .....	91
3.405	Unit Data .....	91
3.406	Upshifting .....	91
3.407	User Database .....	91
3.408	User ID.....	91
3.409	User Name .....	91
3.410	Utility .....	92
3.411	Variable .....	92
3.412	Vertical-Tab Character (<vertical-tab>).....	92
3.413	White Space.....	92
3.414	White-Space Byte .....	92
3.415	White-Space Character .....	92
3.416	White-Space Wide Character.....	92
3.417	Wide-Character Code (C Language) .....	93
3.418	Wide-Character Input/Output Functions .....	93
3.419	Wide-Character String.....	93
3.420	Word.....	93
3.421	Working Directory (or Current Working Directory) .....	93
3.422	Worldwide Portability Interface .....	93
3.423	Write.....	93
3.424	XSI .....	93
3.425	XSI-Conformant .....	94
3.426	Zombie Process.....	94

	3.427	Zombie Thread .....	94
	3.428	±0 .....	94
<b>Chapter</b>	<b>4</b>	<b>General Concepts .....</b>	<b>95</b>
	4.1	Case Insensitive Comparisons .....	95
	4.2	Concurrent Execution .....	95
	4.3	Default Initialization .....	95
	4.4	Directory Operations .....	96
	4.5	Directory Protection .....	96
	4.6	Extended Security Controls .....	96
	4.7	File Access Permissions .....	97
	4.8	File Hierarchy .....	97
	4.9	Filenames .....	97
	4.10	Filename Portability .....	98
	4.11	File System Cache .....	98
	4.12	File Times Update .....	98
	4.13	Host and Network Byte Orders .....	99
	4.14	Measurement of Execution Time .....	99
	4.15	Memory Ordering and Synchronization .....	100
	4.15.1	Memory Ordering .....	100
	4.15.2	Memory Synchronization .....	104
	4.16	Pathname Resolution .....	105
	4.17	Process ID Reuse .....	106
	4.18	Scheduling Policy .....	107
	4.19	Seconds Since the Epoch .....	107
	4.20	Semaphore .....	108
	4.21	Special Device Drivers .....	108
	4.22	Thread-Safety .....	108
	4.23	Treatment of Error Conditions for Mathematical Functions .....	109
	4.23.1	Domain Error .....	109
	4.23.2	Pole Error .....	109
	4.23.3	Range Error .....	110
	4.24	Treatment of NaN Arguments for the Mathematical Functions .....	110
	4.25	Utility .....	111
	4.26	Variable Assignment .....	111
<b>Chapter</b>	<b>5</b>	<b>File Format Notation .....</b>	<b>113</b>
<b>Chapter</b>	<b>6</b>	<b>Character Set .....</b>	<b>117</b>
	6.1	Portable Character Set .....	117
	6.2	Character Encoding .....	120
	6.3	C Language Wide-Character Codes .....	120
	6.4	Character Set Description File .....	121
	6.4.1	State-Dependent Character Encodings .....	125
<b>Chapter</b>	<b>7</b>	<b>Locale .....</b>	<b>127</b>
	7.1	General .....	127
	7.2	POSIX Locale .....	128
	7.3	Locale Definition .....	128
	7.3.1	LC_CTYPE .....	131

	7.3.2	LC_COLLATE.....	139
	7.3.3	LC_MONETARY .....	147
	7.3.4	LC_NUMERIC.....	151
	7.3.5	LC_TIME .....	152
	7.3.6	LC_MESSAGES .....	159
	7.4	Locale Definition Grammar .....	160
	7.4.1	Locale Lexical Conventions .....	160
	7.4.2	Locale Grammar.....	161
<b>Chapter</b>	<b>8</b>	<b>Environment Variables.....</b>	<b>167</b>
	8.1	Environment Variable Definition.....	167
	8.2	Internationalization Variables .....	169
	8.3	Other Environment Variables.....	174
<b>Chapter</b>	<b>9</b>	<b>Regular Expressions.....</b>	<b>179</b>
	9.1	Regular Expression Definitions.....	179
	9.2	Regular Expression General Requirements.....	180
	9.3	Basic Regular Expressions .....	181
	9.3.1	BREs Matching a Single Character or Collating Element.....	181
	9.3.2	BRE Ordinary Characters.....	181
	9.3.3	BRE Special Characters .....	182
	9.3.4	Periods in BREs .....	182
	9.3.5	RE Bracket Expression.....	182
	9.3.6	BREs Matching Multiple Characters .....	185
	9.3.7	BRE Precedence .....	186
	9.3.8	BRE Expression Anchoring.....	186
	9.4	Extended Regular Expressions.....	187
	9.4.1	EREs Matching a Single Character or Collating Element.....	187
	9.4.2	ERE Ordinary Characters.....	187
	9.4.3	ERE Special Characters .....	188
	9.4.4	Periods in EREs .....	188
	9.4.5	ERE Bracket Expression .....	188
	9.4.6	EREs Matching Multiple Characters .....	189
	9.4.7	ERE Alternation.....	190
	9.4.8	ERE Precedence .....	190
	9.4.9	ERE Expression Anchoring.....	190
	9.5	Regular Expression Grammar .....	191
	9.5.1	BRE/ERE Grammar Lexical Conventions.....	191
	9.5.2	RE and Bracket Expression Grammar.....	192
	9.5.3	ERE Grammar.....	194
<b>Chapter</b>	<b>10</b>	<b>Directory Structure and Devices .....</b>	<b>197</b>
	10.1	Directory Structure and Files.....	197
	10.2	Output Devices and Terminal Types.....	197
<b>Chapter</b>	<b>11</b>	<b>General Terminal Interface .....</b>	<b>199</b>
	11.1	Interface Characteristics .....	199
	11.1.1	Opening a Terminal Device File.....	199
	11.1.2	Process Groups .....	199
	11.1.3	The Controlling Terminal.....	200

	11.1.4	Terminal Access Control .....	200
	11.1.5	Input Processing and Reading Data .....	201
	11.1.6	Canonical Mode Input Processing.....	202
	11.1.7	Non-Canonical Mode Input Processing.....	202
	11.1.8	Writing Data and Output Processing .....	203
	11.1.9	Special Characters .....	203
	11.1.10	Modem Disconnect .....	205
	11.1.11	Closing a Terminal Device File.....	205
	11.2	Parameters that Can be Set .....	205
	11.2.1	The termios Structure .....	205
	11.2.2	Input Modes.....	206
	11.2.3	Output Modes.....	207
	11.2.4	Control Modes .....	209
	11.2.5	Local Modes.....	210
	11.2.6	Special Control Characters.....	212
<b>Chapter</b>	<b>12</b>	<b>Utility Conventions.....</b>	<b>213</b>
	12.1	Utility Argument Syntax.....	213
	12.2	Utility Syntax Guidelines.....	215
<b>Chapter</b>	<b>13</b>	<b>Namespace and Future Directions .....</b>	<b>219</b>
<b>Chapter</b>	<b>14</b>	<b>Headers .....</b>	<b>221</b>
<b>Volume</b>	<b>2</b>	<b>System Interfaces, Issue 8.....</b>	<b>491</b>
<b>Chapter</b>	<b>1</b>	<b>Introduction.....</b>	<b>493</b>
	1.1	Relationship to Other Formal Standards.....	493
	1.2	Format of Entries.....	493
<b>Chapter</b>	<b>2</b>	<b>General Information .....</b>	<b>495</b>
	2.1	Use and Implementation of Interfaces .....	495
	2.1.1	Use and Implementation of Functions.....	495
	2.1.2	Use and Implementation of Macros .....	496
	2.2	The Compilation Environment .....	496
	2.2.1	POSIX.1 Symbols.....	496
	2.2.2	The Name Space.....	498
	2.3	Error Numbers.....	507
	2.3.1	Additional Error Numbers .....	513
	2.4	Signal Concepts .....	513
	2.4.1	Signal Generation and Delivery.....	513
	2.4.2	Realtime Signal Generation and Delivery .....	515
	2.4.3	Signal Actions .....	516
	2.4.4	Signal Effects on Other Functions.....	520
	2.5	Standard I/O Streams .....	521
	2.5.1	Interaction of File Descriptors and Standard I/O Streams .....	522
	2.5.2	Stream Orientation and Encoding Rules .....	524
	2.6	File Descriptor Allocation.....	525
	2.7	XSI Interprocess Communication .....	526
	2.7.1	IPC General Description .....	526

2.8	Realtime .....	527
2.8.1	Realtime Signals .....	528
2.8.2	Asynchronous I/O .....	528
2.8.3	Memory Management .....	529
2.8.4	Process Scheduling .....	531
2.8.5	Clocks and Timers .....	535
2.9	Threads .....	537
2.9.1	Thread-Safety .....	537
2.9.2	Thread IDs .....	538
2.9.3	Thread Mutexes .....	539
2.9.4	Thread Scheduling .....	540
2.9.5	Thread Cancellation .....	542
2.9.6	Thread Read-Write Locks .....	547
2.9.7	Thread Interactions with File Operations .....	547
2.9.8	Use of Application-Managed Thread Stacks .....	548
2.9.9	Synchronization Object Copies and Alternative Mappings .....	548
2.10	Sockets .....	549
2.10.1	Address Families .....	549
2.10.2	Addressing .....	549
2.10.3	Protocols .....	549
2.10.4	Routing .....	550
2.10.5	Interfaces .....	550
2.10.6	Socket Types .....	550
2.10.7	Socket I/O Mode .....	551
2.10.8	Socket Owner .....	551
2.10.9	Socket Queue Limits .....	551
2.10.10	Pending Error .....	551
2.10.11	Socket Receive Queue .....	552
2.10.12	Socket Out-of-Band Data State .....	552
2.10.13	Connection Indication Queue .....	553
2.10.14	Signals .....	553
2.10.15	Asynchronous Errors .....	553
2.10.16	Use of Options .....	554
2.10.17	Use of Sockets for Local UNIX Connections .....	557
2.10.18	Use of Sockets over Internet Protocols .....	558
2.10.19	Use of Sockets over Internet Protocols Based on IPv4 .....	558
2.10.20	Use of Sockets over Internet Protocols Based on IPv6 .....	558
2.11	Data Types .....	561
2.11.1	Defined Types .....	562
2.11.2	The char Type .....	563
2.12	Status Information .....	563
<b>Chapter 3</b>	<b>System Interfaces .....</b>	<b>565</b>
<b>Volume 3</b>	<b>Shell and Utilities, Issue 8 .....</b>	<b>2451</b>
<b>Chapter 1</b>	<b>Introduction .....</b>	<b>2453</b>
1.1	Relationship to Other Documents .....	2453

1.1.1	System Interfaces.....	2453
1.1.2	Concepts Derived from the ISO C Standard .....	2457
1.2	Utility Limits.....	2459
1.3	Grammar Conventions.....	2461
1.4	Utility Description Defaults.....	2462
1.5	Considerations for Utilities in Support of Files of Arbitrary Size.....	2469
1.6	Built-In Utilities .....	2470
1.7	Intrinsic Utilities.....	2470
<b>Chapter 2</b>	<b>Shell Command Language .....</b>	<b>2472</b>
2.1	Shell Introduction.....	2472
2.2	Quoting.....	2472
2.2.1	Escape Character (Backslash).....	2473
2.2.2	Single-Quotes.....	2473
2.2.3	Double-Quotes.....	2473
2.2.4	Dollar-Single-Quotes .....	2474
2.3	Token Recognition.....	2475
2.3.1	Alias Substitution.....	2477
2.4	Reserved Words.....	2478
2.5	Parameters and Variables.....	2478
2.5.1	Positional Parameters .....	2479
2.5.2	Special Parameters .....	2479
2.5.3	Shell Variables.....	2481
2.6	Word Expansions .....	2483
2.6.1	Tilde Expansion .....	2485
2.6.2	Parameter Expansion.....	2485
2.6.3	Command Substitution .....	2489
2.6.4	Arithmetic Expansion.....	2490
2.6.5	Field Splitting .....	2491
2.6.6	Pathname Expansion .....	2493
2.6.7	Quote Removal.....	2493
2.7	Redirection .....	2493
2.7.1	Redirecting Input .....	2494
2.7.2	Redirecting Output .....	2494
2.7.3	Appending Redirected Output .....	2495
2.7.4	Here-Document .....	2495
2.7.5	Duplicating an Input File Descriptor .....	2497
2.7.6	Duplicating an Output File Descriptor .....	2497
2.7.7	Open File Descriptors for Reading and Writing.....	2497
2.8	Exit Status and Errors .....	2497
2.8.1	Consequences of Shell Errors .....	2497
2.8.2	Exit Status for Commands .....	2499
2.9	Shell Commands .....	2499
2.9.1	Simple Commands.....	2500
2.9.2	Pipelines .....	2504
2.9.3	Lists .....	2505
2.9.4	Compound Commands.....	2508
2.9.5	Function Definition Command .....	2511
2.10	Shell Grammar.....	2512
2.10.1	Shell Grammar Lexical Conventions.....	2512
2.10.2	Shell Grammar Rules.....	2513



2.11	Job Control .....	2518
2.12	Signals and Error Handling.....	2521
2.13	Shell Execution Environment.....	2522
2.14	Pattern Matching Notation.....	2523
2.14.1	Patterns Matching a Single Character.....	2523
2.14.2	Patterns Matching Multiple Characters.....	2524
2.14.3	Patterns Used for Filename Expansion.....	2525
2.15	Special Built-In Utilities.....	2526
<b>Chapter 3</b>	<b>Utilities.....</b>	<b>2573</b>
<b>Volume 4</b>	<b>Rationale (Informative), Issue 8.....</b>	<b>3633</b>
<b>Part A</b>	<b>Base Definitions .....</b>	<b>3635</b>
<b>Appendix A</b>	<b>Rationale for Base Definitions.....</b>	<b>3637</b>
A.1	Introduction .....	3637
A.1.1	Scope .....	3637
A.1.2	Word Usage.....	3639
A.1.3	Conformance.....	3639
A.1.4	Normative References .....	3639
A.1.5	Change History .....	3639
A.1.6	Terminology .....	3639
A.1.7	Definitions and Concepts.....	3642
A.1.8	Portability.....	3642
A.2	Conformance.....	3643
A.2.1	Implementation Conformance .....	3643
A.2.2	Application Conformance.....	3647
A.2.3	Language-Dependent Services for the C Programming Language.....	3648
A.2.4	Other Language-Related Specifications.....	3648
A.3	Definitions .....	3648
A.4	General Concepts .....	3676
A.4.1	Case Insensitive Comparisons .....	3676
A.4.2	Concurrent Execution.....	3676
A.4.3	Default Initialization.....	3677
A.4.4	Directory Operations .....	3677
A.4.5	Directory Protection.....	3677
A.4.6	Extended Security Controls.....	3677
A.4.7	File Access Permissions.....	3677
A.4.8	File Hierarchy .....	3678
A.4.9	Filenames.....	3678
A.4.10	Filename Portability.....	3679
A.4.11	File System Cache .....	3680
A.4.12	File Times Update .....	3680
A.4.13	Host and Network Byte Order .....	3681
A.4.14	Measurement of Execution Time .....	3681
A.4.15	Memory Ordering and Synchronization .....	3681
A.4.16	Pathname Resolution.....	3683
A.4.17	Process ID Reuse .....	3685
A.4.18	Scheduling Policy .....	3685

A.4.19	Seconds Since the Epoch .....	3685
A.4.20	Semaphore.....	3686
A.4.21	Special Device Drivers.....	3686
A.4.22	Thread-Safety.....	3687
A.4.23	Treatment of Error Conditions for Mathematical Functions .....	3687
A.4.24	Treatment of NaN Arguments for Mathematical Functions .....	3687
A.4.25	Utility .....	3687
A.4.26	Variable Assignment.....	3687
A.5	File Format Notation .....	3688
A.6	Character Set.....	3688
A.6.1	Portable Character Set .....	3688
A.6.2	Character Encoding .....	3689
A.6.3	C Language Wide-Character Codes .....	3689
A.6.4	Character Set Description File.....	3690
A.7	Locale .....	3692
A.7.1	General.....	3692
A.7.2	POSIX Locale .....	3693
A.7.3	Locale Definition .....	3693
A.7.4	Locale Definition Grammar .....	3701
A.7.5	Locale Definition Example.....	3701
A.8	Environment Variables .....	3704
A.8.1	Environment Variable Definition.....	3704
A.8.2	Internationalization Variables .....	3705
A.8.3	Other Environment Variables.....	3706
A.9	Regular Expressions .....	3709
A.9.1	Regular Expression Definitions.....	3709
A.9.2	Regular Expression General Requirements.....	3710
A.9.3	Basic Regular Expressions .....	3711
A.9.4	Extended Regular Expressions.....	3715
A.9.5	Regular Expression Grammar .....	3716
A.10	Directory Structure and Devices.....	3717
A.10.1	Directory Structure and Files.....	3717
A.10.2	Output Devices and Terminal Types.....	3717
A.11	General Terminal Interface .....	3718
A.11.1	Interface Characteristics .....	3719
A.11.2	Parameters that Can be Set .....	3723
A.12	Utility Conventions.....	3724
A.12.1	Utility Argument Syntax.....	3724
A.12.2	Utility Syntax Guidelines.....	3725
A.13	Namespace and Future Directions .....	3728
A.14	Headers.....	3728
A.14.1	Format of Entries.....	3728
A.14.2	Removed Headers in Issue 8 .....	3728
<b>Part</b>	<b>B System Interfaces.....</b>	<b>3729</b>
<b>Appendix</b>	<b>B Rationale for System Interfaces.....</b>	<b>3731</b>
B.1	Introduction .....	3731
B.1.1	Change History .....	3731

B.1.2	Relationship to Other Formal Standards.....	3735
B.1.3	Format of Entries.....	3735
B.2	General Information.....	3735
B.2.1	Use and Implementation of Interfaces.....	3735
B.2.2	The Compilation Environment.....	3737
B.2.3	Error Numbers.....	3742
B.2.4	Signal Concepts.....	3746
B.2.5	Standard I/O Streams.....	3757
B.2.6	File Descriptor Allocation.....	3758
B.2.7	XSI Interprocess Communication.....	3758
B.2.8	Realtime.....	3759
B.2.9	Threads.....	3806
B.2.10	Sockets.....	3835
B.2.11	Data Types.....	3837
B.2.12	Status Information.....	3840
B.3	System Interfaces.....	3840
B.3.1	System Interfaces Removed in this Version.....	3840
B.3.2	System Interfaces Removed in the Previous Version.....	3842
B.3.3	Examples for Spawn.....	3842
<b>Part</b>	<b>C Shell and Utilities.....</b>	<b>3853</b>
<b>Appendix</b>	<b>C Rationale for Shell and Utilities.....</b>	<b>3855</b>
C.1	Introduction.....	3855
C.1.1	Change History.....	3855
C.1.2	Relationship to Other Documents.....	3856
C.1.3	Utility Limits.....	3857
C.1.4	Grammar Conventions.....	3860
C.1.5	Utility Description Defaults.....	3860
C.1.6	Considerations for Utilities in Support of Files of Arbitrary Size.....	3864
C.1.7	Built-In Utilities.....	3864
C.1.8	Intrinsic Utilities.....	3865
C.2	Shell Command Language.....	3866
C.2.1	Shell Introduction.....	3866
C.2.2	Quoting.....	3866
C.2.3	Token Recognition.....	3871
C.2.4	Reserved Words.....	3873
C.2.5	Parameters and Variables.....	3874
C.2.6	Word Expansions.....	3880
C.2.7	Redirection.....	3890
C.2.8	Exit Status and Errors.....	3894
C.2.9	Shell Commands.....	3895
C.2.10	Shell Grammar.....	3905
C.2.11	Job Control.....	3906
C.2.12	Signals and Error Handling.....	3907
C.2.13	Shell Execution Environment.....	3907
C.2.14	Pattern Matching Notation.....	3908
C.2.15	Special Built-In Utilities.....	3912
C.3	Utilities.....	3912
C.3.1	Utilities Removed in this Version.....	3912

	C.3.2	Utilities Removed in the Previous Version.....	3912
	C.3.3	Exclusion of Utilities.....	3913
<b>Part</b>	<b>D</b>	<b>Portability Considerations.....</b>	<b>3917</b>
<b>Appendix</b>	<b>D</b>	<b>Portability Considerations (Informative).....</b>	<b>3919</b>
	D.1	User Requirements.....	3919
	D.1.1	Configuration Interrogation .....	3920
	D.1.2	Process Management .....	3920
	D.1.3	Access to Data.....	3920
	D.1.4	Access to the Environment .....	3920
	D.1.5	Access to Determinism and Performance Enhancements.....	3920
	D.1.6	Operating System-Dependent Profile .....	3921
	D.1.7	I/O Interaction .....	3921
	D.1.8	Internationalization Interaction .....	3921
	D.1.9	C-Language Extensions.....	3921
	D.1.10	Command Language .....	3921
	D.1.11	Interactive Facilities .....	3921
	D.1.12	Accomplish Multiple Tasks Simultaneously .....	3921
	D.1.13	Complex Data Manipulation .....	3922
	D.1.14	File Hierarchy Manipulation .....	3922
	D.1.15	Locale Configuration .....	3922
	D.1.16	Inter-User Communication.....	3922
	D.1.17	System Environment .....	3922
	D.1.18	Printing.....	3922
	D.1.19	Software Development.....	3922
	D.2	Portability Capabilities.....	3923
	D.2.1	Configuration Interrogation .....	3923
	D.2.2	Process Management .....	3924
	D.2.3	Access to Data.....	3924
	D.2.4	Access to the Environment .....	3925
	D.2.5	Bounded (Realtime) Response .....	3926
	D.2.6	Operating System-Dependent Profile .....	3926
	D.2.7	I/O Interaction .....	3926
	D.2.8	Internationalization Interaction .....	3927
	D.2.9	C-Language Extensions.....	3927
	D.2.10	Command Language .....	3927
	D.2.11	Interactive Facilities .....	3928
	D.2.12	Accomplish Multiple Tasks Simultaneously .....	3928
	D.2.13	Complex Data Manipulation .....	3928
	D.2.14	File Hierarchy Manipulation .....	3929
	D.2.15	Locale Configuration .....	3929
	D.2.16	Inter-User Communication.....	3929
	D.2.17	System Environment .....	3930
	D.2.18	Printing.....	3930
	D.2.19	Software Development.....	3930
	D.2.20	Future Growth .....	3930
	D.3	Profiling Considerations .....	3931
	D.3.1	Configuration Options .....	3931
	D.3.2	Configuration Options (Shell and Utilities) .....	3931

D.3.3	Configurable Limits .....	3932
D.3.4	Configuration Options (System Interfaces) .....	3933
D.3.5	Configurable Limits .....	3937
D.3.6	Optional Behavior .....	3940
<b>Part E</b>	<b>Subprofiling Considerations .....</b>	<b>3941</b>
<b>Appendix E</b>	<b>Subprofiling Considerations (Informative) .....</b>	<b>3943</b>
E.1	Subprofiling Option Groups .....	3943
	<b>Index .....</b>	<b>3951</b>
<b>List of Figures</b>		
3-1	pax Format Archive Example .....	3263
B-1	Example of a System with Typed Memory .....	3777
<b>List of Tables</b>		
3-1	Job ID Formats .....	58
5-1	Escape Sequences and Associated Actions .....	113
6-1	Portable Character Set .....	117
6-2	Non-Portable Control Characters .....	122
7-1	Valid Character Class Combinations .....	135
10-1	Control Character Names .....	198
2-1	Value of Level for Socket Options .....	554
2-2	Socket-Level Options .....	555
1-1	Actions when Creating a File that Already Exists .....	2455
1-2	Selected ISO C Standard Operators and Control Flow Keywords .....	2458
1-3	Utility Limit Minimum Values .....	2459
1-4	Symbolic Utility Limits .....	2460
1-5	Intrinsic Utilities .....	2470
3-1	Expressions in Decreasing Precedence in <i>awk</i> .....	2608
3-2	Escape Sequences in <i>awk</i> .....	2616
3-3	Operators in <i>bc</i> .....	2656
3-4	Programming Environments: Type Sizes .....	2675
3-5	Programming Environments: <i>c17</i> Arguments .....	2676
3-6	Threaded Programming Environment: <i>c17</i> Arguments .....	2677
3-7	Compression algorithms, <i>-m</i> option-argument values, and suffixes .....	2737
3-8	ASCII to EBCDIC Conversion .....	2780
3-9	ASCII to IBM EBCDIC Conversion .....	2781
3-10	File Utility Output Strings .....	2933
3-11	Table Size Declarations in <i>lex</i> .....	3040
3-12	Escape Sequences in <i>lex</i> .....	3042
3-13	ERE Precedence in <i>lex</i> .....	3042
3-14	Named Characters in <i>od</i> .....	3229
3-15	ustar Header Block .....	3268
3-16	ustar <i>mode</i> Field .....	3269
3-17	Octet-Oriented <i>cpio</i> Archive Entry .....	3272

## Contents

3-18	Values for <code>cpio c_mode</code> Field .....	3273
3-19	Variable Names and Default Headers in <i>ps</i> .....	3314
3-20	Control Character Names in <i>stty</i> .....	3409
3-21	Circumflex Control Characters in <i>stty</i> .....	3410
3-22	<code>uuencode</code> Base64 Values .....	3512
3-23	Internal Limits in <i>yacc</i> .....	3626
A-1	Historical Practice for Symbolic Links.....	3672

# Trademarks

The following information is given for the convenience of users of POSIX.1-2024 and does not constitute an endorsement by the IEEE or The Open Group of these products. Equivalent products may be used if they can be shown to lead to the same results.

There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

AIX<sup>®</sup> and IBM<sup>®</sup> are registered trademarks of International Business Machines Corporation.

ArchiMate<sup>®</sup>, FACE<sup>®</sup>, FACE<sup>®</sup> logo, Future Airborne Capability Environment<sup>®</sup>, Making Standards Work<sup>®</sup>, Open Footprint<sup>®</sup>, Open O<sup>®</sup> logo, Open O and Check<sup>®</sup> certification logo, OSDU<sup>®</sup>, Platform 3.0<sup>®</sup>, The Open Group<sup>®</sup>, TOGAF<sup>®</sup>, UNIX<sup>®</sup>, UNIXWARE<sup>®</sup>, and X<sup>®</sup> logo are registered trademarks and Boundaryless Information Flow<sup>™</sup>, Build with Integrity Buy with Confidence<sup>™</sup>, Commercial Aviation Reference Architecture<sup>™</sup>, Dependability Through Assuredness<sup>™</sup>, Digital Practitioner Body of Knowledge<sup>™</sup>, DPBoK<sup>™</sup>, EMMM<sup>™</sup>, FHIM Profile Builder<sup>™</sup>, FHIM logo, FPB<sup>™</sup>, IT4IT<sup>™</sup>, IT4IT<sup>™</sup> logo, O-AA<sup>™</sup>, O-DA<sup>™</sup>, O-DEF<sup>™</sup>, O-HERA<sup>™</sup>, O-PAS<sup>™</sup>, O-TTPS<sup>™</sup>, Open Agile Architecture<sup>™</sup>, Open FAIR<sup>™</sup>, Open Process Automation<sup>™</sup>, Open Subsurface Data Universe<sup>™</sup>, Open Trusted Technology Provider<sup>™</sup>, Sensor Integration Simplified<sup>™</sup>, Sensor Open Systems Architecture<sup>™</sup>, SOSA<sup>™</sup>, and SOSA<sup>™</sup> logo are trademarks of The Open Group.

AT&T<sup>®</sup> is a registered trademark of AT&T in the USA and other countries.

BSD<sup>™</sup> is a trademark of the University of California, Berkeley, USA.

Hewlett Packard<sup>®</sup>, HP<sup>®</sup>, and HP-UX<sup>®</sup> are registered trademarks of HP Hewlett Packard Group LLC.

IEEE<sup>®</sup> is a registered trademark, and POSIX<sup>™</sup>, 754<sup>™</sup>, 854<sup>™</sup>, 1003.0<sup>™</sup>, 1003.1<sup>™</sup>, 1003.1d<sup>™</sup>, 1003.1g<sup>™</sup>, 1003.1j<sup>™</sup>, 1003.1q<sup>™</sup>, 1003.2<sup>™</sup>, 1003.2a<sup>™</sup>, 1003.2d<sup>™</sup>, 1003.9<sup>™</sup>, and 1003.13<sup>™</sup> are trademarks of The Institute of Electrical and Electronic Engineers, Inc.

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Sun<sup>®</sup> and Sun Microsystems<sup>®</sup> are registered trademarks of Oracle America, Inc.

/usr/group<sup>®</sup> is a registered trademark of UniForum, the International Network of UNIX System Users.

# Acknowledgements

The contributions of the following organizations to the development of POSIX.1-2024 are gratefully acknowledged:

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation
- Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems Inc. for permission to reproduce portions of their copyrighted documentation
- ISO/IEC JTC 1/SC 22/WG 14 C Language Committee
- Red Hat Inc. for permission to reproduce portions of its copyrighted documentation

POSIX.1-2024 was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.



# Referenced Documents

## Normative References

Normative references for POSIX.1-2024 are defined in [Section 1.4](#) (on page 5).

## Informative References

The following documents are referenced in POSIX.1-2024:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*, SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559: 1989

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559: 1989).

IEEE Standards Terms

IEEE 100, The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition.

- IEEE Std 754<sup>TM</sup>-1985  
IEEE Std 754-1985 (Reaff 1990), IEEE Standard for Binary Floating-Point Arithmetic.
- IEEE Std 854<sup>TM</sup>-1987  
IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.
- IEEE Std 1003.9<sup>TM</sup>-1992  
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- IETF RFC 791  
Internet Protocol, Version 4 (IPv4), September 1981 (available at: [www.ietf.org/rfc/rfc0791.txt](http://www.ietf.org/rfc/rfc0791.txt)).
- IETF RFC 819  
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982 (available at: [www.ietf.org/rfc/rfc0819.txt](http://www.ietf.org/rfc/rfc0819.txt)).
- IETF RFC 919  
Broadcasting Internet Datagrams, J. Mogul, October 1984 (available at: [www.ietf.org/rfc/rfc0919.txt](http://www.ietf.org/rfc/rfc0919.txt)).
- IETF RFC 920  
Domain Requirements, J. Postel, J. Reynolds, October 1984 (available at: [www.ietf.org/rfc/rfc0920.txt](http://www.ietf.org/rfc/rfc0920.txt)).
- IETF RFC 921  
Domain Name System Implementation Schedule, J. Postel, October 1984 (available at: [www.ietf.org/rfc/rfc0921.txt](http://www.ietf.org/rfc/rfc0921.txt)).
- IETF RFC 922  
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984 (available at: [www.ietf.org/rfc/rfc0922.txt](http://www.ietf.org/rfc/rfc0922.txt)).
- IETF RFC 1034  
Domain Names — Concepts and Facilities, P. Mockapetris, November 1987 (available at: [www.ietf.org/rfc/rfc1034.txt](http://www.ietf.org/rfc/rfc1034.txt)).
- IETF RFC 1035  
Domain Names — Implementation and Specification, P. Mockapetris, November 1987 (available at: [www.ietf.org/rfc/rfc1035.txt](http://www.ietf.org/rfc/rfc1035.txt)).
- IETF RFC 1123  
Requirements for Internet Hosts — Application and Support, R. Braden, October 1989 (available at: [www.ietf.org/rfc/rfc1123.txt](http://www.ietf.org/rfc/rfc1123.txt)).
- IETF RFC 1951  
DEFLATE Compressed Data Format Specification version 1.3, P. Deutsch, May 1996 (available at: [www.ietf.org/rfc/rfc1951.txt](http://www.ietf.org/rfc/rfc1951.txt)).
- IETF RFC 1952  
GZIP file format specification version 4.3, P. Deutsch, May 1996 (available at: [www.ietf.org/rfc/rfc1952.txt](http://www.ietf.org/rfc/rfc1952.txt)).
- IETF RFC 2045  
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996 (available at: [www.ietf.org/rfc/rfc2045.txt](http://www.ietf.org/rfc/rfc2045.txt)).

- IETF RFC 2181  
Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997 (available at: [www.ietf.org/rfc/rfc2181.txt](http://www.ietf.org/rfc/rfc2181.txt)).
- IETF RFC 3596  
DNS Extensions to Support IP Version 6, S. Thomson, C. Huitema, V. Ksinant, M. Souissi, October 2003 (available at: [www.ietf.org/rfc/rfc3596.txt](http://www.ietf.org/rfc/rfc3596.txt)).
- IETF RFC 4291  
IP Version 6 Addressing Architecture, R. Hinden, S. Deering, February 2006 (available at: [www.ietf.org/rfc/rfc4291.txt](http://www.ietf.org/rfc/rfc4291.txt)).
- IETF RFC 5322  
Internet Message Format, P. Resnick, October 2008 (available at: [www.ietf.org/rfc/rfc5322.txt](http://www.ietf.org/rfc/rfc5322.txt)).
- IETF RFC 6557  
Procedures for Maintaining the Time Zone Database, E. Lear, P. Eggert, February 2012 (available at: [www.ietf.org/rfc/rfc6557.txt](http://www.ietf.org/rfc/rfc6557.txt)).
- IETF RFC 8200  
Internet Protocol, Version 6 (IPv6) Specification, S. Deering, R. Hinden, July 2017 (available at: [www.ietf.org/rfc/rfc8200.txt](http://www.ietf.org/rfc/rfc8200.txt)).
- Internationalisation Guide  
Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.
- ISO 2375: 1985  
ISO 2375: 1985, Data Processing — Procedure for Registration of Escape Sequences.
- ISO 8652: 1987  
ISO 8652: 1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).
- ISO/IEC 1539: 1991  
ISO/IEC 1539: 1991, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).
- ISO/IEC 4873: 1991  
ISO/IEC 4873: 1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.
- ISO/IEC 6429: 1992  
ISO/IEC 6429: 1992, Information Technology — Control Functions for Coded Character Sets.
- ISO/IEC 6937: 1994  
ISO/IEC 6937: 1994, Information Technology — Coded Graphic Character Set for Text Communication — Latin Alphabet.
- ISO/IEC 8802-3: 1996  
ISO/IEC 8802-3: 1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.
- ISO/IEC 8859  
ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

## Referenced Documents

- Part 1: Latin Alphabet No. 1
  - Part 2: Latin Alphabet No. 2
  - Part 3: Latin Alphabet No. 3
  - Part 4: Latin Alphabet No. 4
  - Part 5: Latin/Cyrillic Alphabet
  - Part 6: Latin/Arabic Alphabet
  - Part 7: Latin/Greek Alphabet
  - Part 8: Latin/Hebrew Alphabet
  - Part 9: Latin Alphabet No. 5
  - Part 10: Latin Alphabet No. 6
  - Part 11: Latin/Thai Alphabet
  - Part 13: Latin Alphabet No. 7
  - Part 14: Latin Alphabet No. 8 (Celtic)
  - Part 15: Latin Alphabet No. 9
  - Part 16: Latin Alphabet No. 10
- ISO/IEC 9899: 1990  
ISO/IEC 9899: 1990, Programming Languages — C, including Amendment 1: 1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).
- ISO POSIX-1: 1996  
ISO/IEC 9945-1: 1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.
- ISO POSIX-2: 1993  
ISO/IEC 9945-2: 1993, Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2<sup>TM</sup>-1992, as amended by ANSI/IEEE Std 1003.2a<sup>TM</sup>-1992).
- Issue 1  
X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).
- Issue 2  
X/Open Portability Guide, January 1987:
- Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
  - Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)
- Issue 3  
X/Open Specification, 1988, 1989, February 1992:
- Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
  - System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
  - Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

- Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)

Issue 4

CAE Specification, July 1992, published by The Open Group:

- System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
- Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
- System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

- System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

Technical Standard, February 1997, published by The Open Group:

- System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Issue 6

Technical Standard, April 2004, published by The Open Group:

- Base Definitions (XBD), Issue 6 (ISBN: 1-931624-43-7, C046)
- System Interfaces (XSH), Issue 6 (ISBN: 1-931624-44-5, C047)
- Shell and Utilities (XCU), Issue 6 (ISBN: 1-931624-45-3, C048)

Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

POSIX.0: 1995

IEEE Std 1003.0™-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

POSIX.1: 1988

IEEE Std 1003.1™-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

## Referenced Documents

### POSIX.1: 1990

IEEE Std 1003.1™-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

### POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — (C Language) Amendment.

### POSIX.1d: 1999

IEEE Std 1003.1d™-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 4: Additional Realtime Extensions [C Language].

### POSIX.1g: 2000

IEEE Std 1003.1g™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 6: Protocol-Independent Interfaces (PII).

### POSIX.1j: 2000

IEEE Std 1003.1j™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 5: Advanced Realtime Extensions [C Language].

### POSIX.1q: 2000

IEEE Std 1003.1q™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment 7: Tracing [C Language].

### POSIX.2: 1992

IEEE Std 1003.2™-1992, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities.

### POSIX.2b

P1003.2b, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment.

### POSIX.2d: 1994

IEEE Std 1003.2d™-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.

### POSIX.13: 1998

IEEE Std 1003.13™-1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.

### POSIX.26: 2003

IEEE Std 1003.26™-2003, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 26: Device Control Application Program Interface (API) [C Language].

### Sarwate Article

Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.

### Sprunt, Sha, and Lehoczky

Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.

- SVID, Issue 1  
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.
- SVID, Issue 2  
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.
- SVID, Issue 3  
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.
- The AWK Programming Language  
Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.
- The C Programming Language  
Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Englewood Cliffs, NJ, Prentice Hall, 1st Edition (February 1978) ISBN 0-13-110163-3; 2nd Edition (March 1988) ISBN 0-13-110362-8.
- UNIX Programmer's Manual  
American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.
- XNS, Issue 4  
CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.
- XNS, Issue 5  
CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.
- XNS, Issue 5.2  
Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.
- X/Open Curses, Issue 4, Version 2  
CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.
- Yacc  
*Yacc: Yet Another Compiler Compiler*, Stephen C. Johnson, 1978.

### Source Documents

Parts of the following documents were used to create the base documents for POSIX.1-2001:

- AIX 3.2 Manual  
AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).
- OSF/1  
OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).
- OSF AES  
Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

## *Referenced Documents*

### System V Release 2.0

- UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).
- UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).

### System V Release 4.2

Operating System API Reference, UNIX<sup>®</sup> SVR4.2 (1992) (ISBN: 0-13-017658-3).





*The Open Group Standard*

1 **Vol. 1:**  
2 **Base Definitions, Issue 8**

3 *The Open Group*  
4 *The Institute of Electrical and Electronics Engineers, Inc.*



## 1.1 Scope

POSIX.1-2024 defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. It is intended to be used by both application developers and system implementors.

POSIX.1-2024 comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of POSIX.1-2024, including utility conventions and C-language header definitions, are included in the Base Definitions volume of POSIX.1-2024.
2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of POSIX.1-2024.
3. Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume of POSIX.1-2024.
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of POSIX.1-2024 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of POSIX.1-2024.

The following areas are outside of the scope of POSIX.1-2024:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-2024 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

The facilities provided in POSIX.1-2024 are drawn from the following base documents:

- IEEE Std 1003.1-2017 (POSIX.1-2017)
- IEEE Std 1003.26-2003 (POSIX.26-2003)

- 40 • ISO/IEC 9899:2018, Programming Languages — C (C17)
- 41 • ISO/IEC TR 24731-2:2010, Programming languages, their environments and system  
42 software interfaces — Extensions to the C library — Part 2: Dynamic Allocation Functions
- 43 • The Open Group Standard, 2021, Additional APIs for the Base Specifications Issue 8, Part 1
- 44 • The Open Group Standard, 2022, Additional APIs for the Base Specifications Issue 8, Part 2

45 Emphasis has been placed on standardizing existing practice for existing users, with changes  
46 and additions limited to correcting deficiencies in the following areas:

- 47 • Issues raised by Austin Group defect reports and IEEE Interpretations against IEEE Std  
48 1003.1.
- 49 • Issues raised in corrigenda for The Open Group Standards and working group resolutions  
50 from The Open Group
- 51 • Changes to make the text self-consistent with the additional material merged
- 52 • Features, marked obsolescent in the base documents, have been considered for removal in  
53 this version
- 54 • Alignment with the ISO/IEC 9899:2018 standard

## 55 1.2 Word Usage

56 The word *shall* indicates mandatory requirements strictly to be followed in order to conform to  
57 the standard and from which no deviation is permitted (*shall* equals *is required to*).<sup>1, 2</sup>

58 The word *should* indicates that among several possibilities one is recommended as particularly  
59 suitable, without mentioning or excluding others; or that a certain course of action is preferred  
60 but not necessarily required (*should* equals *is recommended that*).

61 The word *may* is used to indicate a course of action permissible within the limits of the standard  
62 (*may* equals *is permitted to*).

63 The word *can* is used for statements of possibility and capability, whether material, physical, or  
64 causal (*can* equals *is able to*).

## 65 1.3 Conformance

66 Conformance requirements for POSIX.1-2024 are defined in [Chapter 2](#) (on page 15).

---

67 1. The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe  
68 unavoidable situations.

69 2. The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

## 1.4 Normative References

The following standards contain provisions which, through references in POSIX.1-2024, constitute provisions of POSIX.1-2024. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on POSIX.1-2024 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 646: 1991

ISO/IEC 646: 1991, Information Processing — ISO 7-Bit Coded Character Set for Information Interchange.<sup>3</sup>

ISO 4217: 2015

ISO 4217: 2015, Codes for the representation of currencies.

ISO 8601-1: 2019

ISO 8601-1: 2019, Date and time — Representations for information interchange — Part 1: Basic rules.

ISO C (C17)

ISO/IEC 9899: 2018, Programming Languages — C.

ISO/IEC 10646: 2020

ISO/IEC 10646: 2020, Information Technology — Universal coded character set (UCS).

## 1.5 Change History

Change history is described in the Rationale (Informative) volume of POSIX.1-2024, and in the CHANGE HISTORY section of reference pages.

## 1.6 Terminology

For the purposes of POSIX.1-2024, the following terminology definitions apply:

### **can**

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to POSIX.1-2024. An application can rely on the existence of the feature or behavior.

### **implementation-defined**

Describes a value or behavior that is not defined by POSIX.1-2024 but is selected by an implementor. The value or behavior may vary among implementations that conform to POSIX.1-2024. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

### **legacy**

Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable

---

3. ISO/IEC documents can be obtained from <https://www.iso.org/store.html>.

110 applications. New applications should use alternative means of obtaining equivalent  
111 functionality.

112 **may**

113 Describes a feature or behavior that is optional for an implementation that conforms to  
114 POSIX.1-2024. An application should not rely on the existence of the feature or behavior. An  
115 application that relies on such a feature or behavior cannot be assured to be portable across  
116 conforming implementations.

117 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

118 **shall**

119 For an implementation that conforms to POSIX.1-2024, describes a feature or behavior that  
120 is mandatory. An application can rely on the existence of the feature or behavior.

121 For an application or user, describes a behavior that is mandatory.

122 **should**

123 For an implementation that conforms to POSIX.1-2024, describes a feature or behavior that  
124 is recommended but not mandatory. An application should not rely on the existence of the  
125 feature or behavior. An application that relies on such a feature or behavior cannot be  
126 assured to be portable across conforming implementations.

127 For an application, describes a feature or behavior that is recommended programming  
128 practice for optimum portability.

129 **undefined**

130 Describes the nature of a value or behavior not defined by POSIX.1-2024 which results from  
131 use of an invalid program construct or invalid data input.

132 The value or behavior may vary among implementations that conform to POSIX.1-2024. An  
133 application should not rely on the existence or validity of the value or behavior. An  
134 application that relies on any particular value or behavior cannot be assured to be portable  
135 across conforming implementations.

136 **unspecified**

137 Describes the nature of a value or behavior not specified by POSIX.1-2024 which results  
138 from use of a valid program construct or valid data input.

139 The value or behavior may vary among implementations that conform to POSIX.1-2024. An  
140 application should not rely on the existence or validity of the value or behavior. An  
141 application that relies on any particular value or behavior cannot be assured to be portable  
142 across conforming implementations.

143 **1.7 Definitions and Concepts**

144 Definitions and concepts are defined in [Chapter 3](#) (on page 31) and [Chapter 4](#) (on page 95).

## 145 1.8 Portability

146 Some of the utilities in the Shell and Utilities volume of POSIX.1-2024 and functions in the  
 147 System Interfaces volume of POSIX.1-2024 describe functionality that might not be fully portable  
 148 to systems meeting the requirements for POSIX conformance (see [Chapter 2](#), on page 15).

149 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in  
 150 the margin identifies the nature of the option, extension, or warning (see [Section 1.8.1](#)). For  
 151 maximum portability, an application should avoid such functionality.

152 Unless the primary task of a utility is to produce textual material on its standard output,  
 153 application developers should not rely on the format or content of any such material that may be  
 154 produced. Where the primary task *is* to provide such material, but the output format is  
 155 incompletely specified, the description is marked with the OF margin code and shading.  
 156 Application developers are warned not to expect that the output of such an interface on one  
 157 system is any guide to its behavior on another system.

### 158 1.8.1 Codes

159 The codes and their meanings are as follows. See also [Section 1.8.2](#) (on page 12).

#### 160 ADV **Advisory Information**

161 The functionality described is optional. The functionality described is also an extension to the  
 162 ISO C standard.

163 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.  
 164 Where additional semantics apply to a function, the material is identified by use of the ADV  
 165 margin legend.

#### 166 CD **C-Language Development Utilities**

167 The functionality described is optional.

168 Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.  
 169 Where additional semantics apply to a utility, the material is identified by use of the CD margin  
 170 legend.

#### 171 CPT **Process CPU-Time Clocks**

172 The functionality described is optional. The functionality described is also an extension to the  
 173 ISO C standard.

174 Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.  
 175 Where additional semantics apply to a function, the material is identified by use of the CPT  
 176 margin legend.

#### 177 CX **Extension to the ISO C standard**

178 The functionality described is an extension to the ISO C standard or a deviation from it.  
 179 Application developers can make use of the functionality as it is supported on all  
 180 POSIX.1-2024-conforming systems.

181 With each function or header from the ISO C standard, a statement is included to the effect that  
 182 “any conflict is unintentional”, or “any other conflict is unintentional” if there is an intentional  
 183 conflict (deviation). That is intended to refer to a direct conflict. POSIX.1-2024 acts in part as a  
 184 profile of the ISO C standard, and it may choose to further constrain behaviors allowed to vary  
 185 by the ISO C standard. Such limitations and other compatible differences are not considered  
 186 conflicts, even if a CX mark is missing. The markings are for information only.

187 Where additional semantics apply to a function or header, the material is identified by use of the  
 188 CX margin legend.

---

189	DC	<b>Device Control</b>
190		The functionality described is optional. The functionality described is also an extension to the
191		ISO C standard.
192		Where applicable, functions are marked with the DC margin legend in the SYNOPSIS section.
193		Where additional semantics apply to a function, the material is identified by use of the DC
194		margin legend.
195	FR	<b>FORTTRAN Runtime Utilities</b>
196		The functionality described is optional.
197		Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.
198		Where additional semantics apply to a utility, the material is identified by use of the FR margin
199		legend.
200	FSC	<b>File Synchronization</b>
201		The functionality described is optional. The functionality described is also an extension to the
202		ISO C standard.
203		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
204		Where additional semantics apply to a function, the material is identified by use of the FSC
205		margin legend.
206	IP6	<b>IPV6</b>
207		The functionality described is optional. The functionality described is also an extension to the
208		ISO C standard.
209		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
210		Where additional semantics apply to a function, the material is identified by use of the IP6
211		margin legend.
212	MC1	<b>Non-Robust Mutex Priority Protection or Non-Robust Mutex Priority Inheritance or Robust</b>
213		<b>Mutex Priority Protection or Robust Mutex Priority Inheritance</b>
214		The functionality described is optional. The functionality described is also an extension to the
215		ISO C standard.
216		This is a shorthand notation for combinations of multiple option codes.
217		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
218		Where additional semantics apply to a function, the material is identified by use of the MC1
219		margin legend.
220		Refer to <a href="#">Section 1.8.2</a> (on page 12).
221	ML	<b>Process Memory Locking</b>
222		The functionality described is optional. The functionality described is also an extension to the
223		ISO C standard.
224		Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
225		Where additional semantics apply to a function, the material is identified by use of the ML
226		margin legend.
227	MLR	<b>Range Memory Locking</b>
228		The functionality described is optional. The functionality described is also an extension to the
229		ISO C standard.
230		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
231		Where additional semantics apply to a function, the material is identified by use of the MLR
232		margin legend.



233	MSG	<b>Message Passing</b>
234		The functionality described is optional. The functionality described is also an extension to the
235		ISO C standard.
236		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
237		Where additional semantics apply to a function, the material is identified by use of the MSG
238		margin legend.
239	MX	<b>IEC 60559 Floating-Point</b>
240		The functionality described is optional. The functionality described is mandated by the ISO C
241		standard only for implementations that define <code>__STDC_IEC_559__</code> .
242	MXC	<b>IEC 60559 Complex Floating-Point</b>
243		The functionality described is optional. The functionality described is mandated by the ISO C
244		standard only for implementations that define <code>__STDC_IEC_559_COMPLEX__</code> .
245	MXX	<b>IEC 60559 Floating-Point Extension</b>
246		The functionality described is optional. The functionality described is part of the IEC 60559
247		Floating-Point option, but is an extension to the ISO C standard.
248	OB	<b>Obsolescent</b>
249		The functionality described may be removed in a future version of this volume of POSIX.1-2024.
250		Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use
251		obsolescent features.
252		Where applicable, the material is identified by use of the OB margin legend.
253	OF	<b>Output Format Incompletely Specified</b>
254		The functionality described is an XSI extension. The format of the output produced by the
255		utility is not fully specified. It is therefore not possible to post-process this output in a consistent
256		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
257		Where applicable, the material is identified by use of the OF margin legend.
258	OH	<b>Optional Header</b>
259		In the SYNOPSIS section of some interfaces in the System Interfaces volume of POSIX.1-2024 an
260		included header is marked as in the following example:
261	OH	<code>#include &lt;sys/types.h&gt;</code>
262		<code>#include &lt;fcntl.h&gt;</code>
263		<code>int open(const char *path, int oflag, ...);</code>
264		The OH margin legend indicates that the optional header defines constants that will be needed if
265		the function is called with certain flag arguments; thus it may be required for some of the
266		functionality described, but is not needed otherwise.
267	PIO	<b>Prioritized Input and Output</b>
268		The functionality described is optional. The functionality described is also an extension to the
269		ISO C standard.
270		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
271		Where additional semantics apply to a function, the material is identified by use of the PIO
272		margin legend.
273	PS	<b>Process Scheduling</b>
274		The functionality described is optional. The functionality described is also an extension to the
275		ISO C standard.
276		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
277		Where additional semantics apply to a function, the material is identified by use of the PS

278		margin legend.
279	RPI	<b>Robust Mutex Priority Inheritance</b>
280		The functionality described is optional. The functionality described is also an extension to the
281		ISO C standard.
282		Where applicable, functions are marked with the RPI margin legend in the SYNOPSIS section.
283		Where additional semantics apply to a function, the material is identified by use of the RPI
284		margin legend.
285	RPP	<b>Robust Mutex Priority Protection</b>
286		The functionality described is optional. The functionality described is also an extension to the
287		ISO C standard.
288		Where applicable, functions are marked with the RPP margin legend in the SYNOPSIS section.
289		Where additional semantics apply to a function, the material is identified by use of the RPP
290		margin legend.
291	RS	<b>Raw Sockets</b>
292		The functionality described is optional. The functionality described is also an extension to the
293		ISO C standard.
294		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
295		Where additional semantics apply to a function, the material is identified by use of the RS
296		margin legend.
297	SD	<b>Software Development Utilities</b>
298		The functionality described is optional.
299		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
300		Where additional semantics apply to a utility, the material is identified by use of the SD margin
301		legend.
302	SHM	<b>Shared Memory Objects</b>
303		The functionality described is optional. The functionality described is also an extension to the
304		ISO C standard.
305		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
306		Where additional semantics apply to a function, the material is identified by use of the SHM
307		margin legend.
308	SIO	<b>Synchronized Input and Output</b>
309		The functionality described is optional. The functionality described is also an extension to the
310		ISO C standard.
311		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
312		Where additional semantics apply to a function, the material is identified by use of the SIO
313		margin legend.
314	SPN	<b>Spawn</b>
315		The functionality described is optional. The functionality described is also an extension to the
316		ISO C standard.
317		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
318		Where additional semantics apply to a function, the material is identified by use of the SPN
319		margin legend.
320	SS	<b>Process Sporadic Server</b>
321		The functionality described is optional. The functionality described is also an extension to the
322		ISO C standard.

323		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
324		Where additional semantics apply to a function, the material is identified by use of the SS
325		margin legend.
326	TCT	<b>Thread CPU-Time Clocks</b>
327		The functionality described is optional. The functionality described is also an extension to the
328		ISO C standard.
329		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
330		Where additional semantics apply to a function, the material is identified by use of the TCT
331		margin legend.
332	TPI	<b>Non-Robust Mutex Priority Inheritance</b>
333		The functionality described is optional. The functionality described is also an extension to the
334		ISO C standard.
335		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
336		Where additional semantics apply to a function, the material is identified by use of the TPI
337		margin legend.
338	TPP	<b>Non-Robust Mutex Priority Protection</b>
339		The functionality described is optional. The functionality described is also an extension to the
340		ISO C standard.
341		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
342		Where additional semantics apply to a function, the material is identified by use of the TPP
343		margin legend.
344	TPS	<b>Thread Execution Scheduling</b>
345		The functionality described is optional. The functionality described is also an extension to the
346		ISO C standard.
347		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
348		Where additional semantics apply to a function, the material is identified by use of the TPS
349		margin legend.
350	TSA	<b>Thread Stack Address Attribute</b>
351		The functionality described is optional. The functionality described is also an extension to the
352		ISO C standard.
353		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
354		Where additional semantics apply to a function, the material is identified by use of the TSA
355		margin legend.
356	TSH	<b>Thread Process-Shared Synchronization</b>
357		The functionality described is optional. The functionality described is also an extension to the
358		ISO C standard.
359		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
360		Where additional semantics apply to a function, the material is identified by use of the TSH
361		margin legend.
362	TSP	<b>Thread Sporadic Server</b>
363		The functionality described is optional. The functionality described is also an extension to the
364		ISO C standard.
365		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
366		Where additional semantics apply to a function, the material is identified by use of the TSP
367		margin legend.

368	TSS	<b>Thread Stack Size Attribute</b>
369		The functionality described is optional. The functionality described is also an extension to the
370		ISO C standard.
371		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
372		Where additional semantics apply to a function, the material is identified by use of the TSS
373		margin legend.
374	TYM	<b>Typed Memory Objects</b>
375		The functionality described is optional. The functionality described is also an extension to the
376		ISO C standard.
377		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
378		Where additional semantics apply to a function, the material is identified by use of the TYM
379		margin legend.
380	UP	<b>User Portability Utilities</b>
381		The functionality described is optional.
382		Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
383		Where additional semantics apply to a utility, the material is identified by use of the UP margin
384		legend.
385	UU	<b>UUCP Utilities</b>
386		The functionality described is optional. The functionality described is also an extension to the
387		ISO C standard.
388		Where applicable, functions are marked with the UU margin legend in the SYNOPSIS section.
389		Where additional semantics apply to a function, the material is identified by use of the UU
390		margin legend.
391	XSI	<b>X/Open System Interfaces</b>
392		The functionality described is part of the X/Open Systems Interfaces option. Functionality
393		marked XSI is an extension to the ISO C standard. Application developers may confidently
394		make use of such extensions on all systems supporting the X/Open System Interfaces option.
395		If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that
396		reference page is an extension. See <a href="#">Section 2.1.4</a> (on page 19).

## 397 1.8.2 Margin Code Notation

398 Some of the functionality described in POSIX.1-2024 depends on support of more than one  
 399 option, or independently may depend on several options. The following notation for margin  
 400 codes is used to denote the following cases.

### 401 A Feature Dependent on One or Two Options

402 In this case, margin codes have a <space> separator; for example:

403	SHM	This feature requires support for only the Shared Memory Objects option.
404	SHM TYM	This feature requires support for both the Shared Memory Objects option and the Typed
405		Memory Objects option; that is, an application which uses this feature is portable only between
406		implementations that provide both options.

**407 A Feature Dependent on Either of the Options Denoted**

408 In this case, margin codes have a ' | ' separator to denote the logical OR; for example:

409 SHM|TYM This feature is dependent on support for either the Shared Memory Objects option or the Typed  
410 Memory Objects option; that is, an application which uses this feature is portable between  
411 implementations that provide any (or all) of the options.

**412 A Feature Dependent on More than Two Options**

413 The following shorthand notations are used:

414 MC1 The MC1 margin code is shorthand for TPP | TPI | RPP | RPI. Features which are shaded with this  
415 margin code require support of either the Non-Robust Mutex Priority Protection option or the  
416 Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection option or  
417 the Robust Mutex Priority Inheritance option.

**418 Large Sections Dependent on an Option**

419 Where large sections of text are dependent on support for an option, a lead-in text block is  
420 provided and shaded accordingly; for example:

421 XSI This section describes extensions to support interprocess communication. The functionality  
422 described in this section shall be provided on implementations that support the XSI option (and  
423 the rest of this section is not further shaded).



# Conformance

## 2.1 Implementation Conformance

For the purposes of POSIX.1-2024, the implementation conformance requirements given in this section apply.

### 2.1.1 Requirements

A *conforming implementation* shall meet all of the following criteria:

1. The system shall support all utilities, functions, and facilities defined within POSIX.1-2024 that are required for POSIX conformance (see [Section 2.1.3](#), on page 17). These interfaces shall support the functional behavior described herein.

2. The system may support the X/Open System Interfaces (XSI) option as described in [Section 2.1.4](#) (on page 19).

3. The system may support one or more options as described under [Section 2.1.5](#) (on page 20). When an implementation claims that an option is supported, all of its constituent parts shall be provided.

4. The system may provide non-standard extensions. These are features not required by POSIX.1-2024 and may include, but are not limited to:

- Additional functions

- Additional headers

- Additional symbols in standard headers

- Additional utilities

- Additional options for standard utilities

- Additional environment variables

- Additional file types

- Non-conforming file systems (for example, legacy file systems for which `_POSIX_NO_TRUNC` is false, case-insensitive file systems, or network file systems)

- Dynamically populated file systems (for example, `/proc`)

- Additional character special files with special properties (for example, `/dev/stdin`, `/dev/stdout`, and `/dev/stderr`)

Non-standard extensions of the utilities, functions, or facilities specified in POSIX.1-2024 should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by POSIX.1-2024. The conformance document shall define an environment in which an application can be run with the behavior specified by POSIX.1-2024. In no case shall such an environment require modification of a Strictly Conforming POSIX Application (see

459                   Section 2.2.1, on page 27).

460       **Note:**       If the documented method of setting up a conforming environment includes the need to set one  
461                   or more environment variables, then the values of those environment variables cannot include  
462                   any <space> characters, since the *confstr()* function has to be able to return them in a  
463                   <space>-separated list of variable=value pairs. See XSH *confstr()* (on page 763).

## 464   2.1.2   Documentation

465       A conformance document with the following information shall be available for an  
466       implementation claiming conformance to POSIX.1-2024. The conformance document shall have  
467       the same structure as POSIX.1-2024, with the information presented in the appropriate sections  
468       and subsections. Sections and subsections that consist solely of subordinate section titles, with  
469       no other information, are not required. The conformance document shall not contain  
470       information about extended facilities or capabilities outside the scope of POSIX.1-2024.

471       The conformance document shall contain a statement that indicates the full name, number, and  
472       date of the standard that applies. The conformance document may also list international  
473       software standards that are available for use by a Conforming POSIX Application. Applicable  
474       characteristics where documentation is required by one of these standards, or by standards of  
475       government bodies, may also be included.

476       The conformance document shall describe the limit values found in the headers **<limits.h>** (on  
477       page 282) and **<unistd.h>** (on page 458), stating values, the conditions under which those values  
478       may change, and the limits of such variations, if any.

479       The conformance document shall describe the behavior of the implementation for all  
480       implementation-defined features defined in POSIX.1-2024. This requirement shall be met by  
481       listing these features and providing either a specific reference to the system documentation or  
482       providing full syntax and semantics of these features. When the value or behavior in the  
483       implementation is designed to be variable or customized on each instantiation of the system, the  
484       implementation provider shall document the nature and permissible ranges of this variation.

485       The conformance document may specify the behavior of the implementation for those features  
486       where POSIX.1-2024 states that implementations may vary or where features are identified as  
487       undefined or unspecified.

488       The conformance document shall not contain documentation other than that specified in the  
489       preceding paragraphs except where such documentation is specifically allowed or required by  
490       other provisions of POSIX.1-2024.

491       The phrases “shall document” or “shall be documented” in POSIX.1-2024 mean that  
492       documentation of the feature shall appear in the conformance document, as described  
493       previously, unless there is an explicit reference in the conformance document to show where the  
494       information can be found in the system documentation.

495       The system documentation should also contain the information found in the conformance  
496       document.



### 497 2.1.3 POSIX Conformance

498 A conforming implementation shall meet the following criteria for POSIX conformance.

#### 499 2.1.3.1 POSIX System Interfaces

500 The following requirements apply to the system interfaces (functions and headers):

- 501 • The system shall support all the mandatory functions and headers defined in  
502 POSIX.1-2024, and shall set the symbolic constant `_POSIX_VERSION` to the value 202405L.
- 503 • Although all implementations conforming to POSIX.1-2024 support all the features  
504 described below, there may be system-dependent or file system-dependent configuration  
505 procedures that can remove or modify any or all of these features. Such configurations  
506 should not be made if strict compliance is required.

507 The following symbolic constants shall be defined with a value other than `-1`. If a constant  
508 is defined with the value zero, applications should use the `sysconf()`, `pathconf()`, or  
509 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the  
510 system at that time or for the particular pathname in question.

511 — `_POSIX_CHOWN_RESTRICTED`

512 The use of `chown()` is restricted to a process with appropriate privileges, and to  
513 changing the group ID of a file only to the effective group ID of the process or to one  
514 of its supplementary group IDs.

515 — `_POSIX_NO_TRUNC`

516 Pathname components longer than `{NAME_MAX}` generate an error.

- 517 • The following symbolic constants shall be defined by the implementation as follows:

518 — Symbolic constants defined with the value 202405L:

519 `_POSIX_ASYNCHRONOUS_IO`  
520 `_POSIX_BARRIERS`  
521 `_POSIX_CLOCK_SELECTION`  
522 `_POSIX_MAPPED_FILES`  
523 `_POSIX_MEMORY_PROTECTION`  
524 `_POSIX_MONOTONIC_CLOCK`  
525 `_POSIX_READER_WRITER_LOCKS`  
526 `_POSIX_REALTIME_SIGNALS`  
527 `_POSIX_SEMAPHORES`  
528 `_POSIX_SPIN_LOCKS`  
529 `_POSIX_THREAD_SAFE_FUNCTIONS`  
530 `_POSIX_THREADS`  
531 `_POSIX_TIMEOUTS`  
532 `_POSIX_TIMERS`  
533 `_POSIX2_C_BIND`

534 — Symbolic constants defined with a value greater than zero:

535 `_POSIX_JOB_CONTROL`  
536 `_POSIX_REGEX`  
537 `_POSIX_SAVED_IDS`  
538 `_POSIX_SHELL`

539 — Symbolic constants defined with a value other than -1.

540 `_POSIX_VDISABLE`

541 **Note:** The symbols above represent historical options that are no longer allowed as options, but  
542 are retained here for backwards-compatibility of applications.

543 • The system may support one or more options (see [Section 2.1.6](#), on page 25) denoted by the  
544 following symbolic constants:

545 `_POSIX_ADVISORY_INFO`  
546 `_POSIX_CPUTIME`  
547 `_POSIX_DEVICE_CONTROL`  
548 `_POSIX_FSYNC`  
549 `_POSIX_IPV6`  
550 `_POSIX_MEMLOCK`  
551 `_POSIX_MEMLOCK_RANGE`  
552 `_POSIX_MESSAGE_PASSING`  
553 `_POSIX_PRIORITIZED_IO`  
554 `_POSIX_PRIORITY_SCHEDULING`  
555 `_POSIX_RAW_SOCKETS`  
556 `_POSIX_SHARED_MEMORY_OBJECTS`  
557 `_POSIX_SPAWN`  
558 `_POSIX_SPORADIC_SERVER`  
559 `_POSIX_SYNCHRONIZED_IO`  
560 `_POSIX_THREAD_ATTR_STACKADDR`  
561 `_POSIX_THREAD_CPUTIME`  
562 `_POSIX_THREAD_ATTR_STACKSIZE`  
563 `_POSIX_THREAD_PRIO_INHERIT`  
564 `_POSIX_THREAD_PRIO_PROTECT`  
565 `_POSIX_THREAD_PRIORITY_SCHEDULING`  
566 `_POSIX_THREAD_PROCESS_SHARED`  
567 `_POSIX_THREAD_SPORADIC_SERVER`  
568 `_POSIX_TYPED_MEMORY_OBJECTS`  
569 `_XOPEN_CRYPT`  
570 `_XOPEN_REALTIME`  
571 `_XOPEN_REALTIME_THREADS`  
572 `_XOPEN_UNIX`

573 If the Advisory Information option is supported, there shall be at least one file system that  
574 supports the functionality.

### 575 2.1.3.2 *POSIX Shell and Utilities*

576 The following requirements apply to the shell and utilities:

- 577 • The system shall provide all the mandatory utilities in the Shell and Utilities volume of  
578 POSIX.1-2024 with all the functional behavior described therein.
- 579 • The system shall support the Large File capabilities described in the Shell and Utilities  
580 volume of POSIX.1-2024.
- 581 • The system may support one or more options (see [Section 2.1.6](#), on page 25) denoted by the  
582 following symbolic constants. (The literal names below apply to the *getconf* utility.)

583           POSIX2\_C\_DEV  
 584           POSIX2\_CHAR\_TERM  
 585           POSIX2\_FORT\_RUN  
 586           POSIX2\_LOCALEDEF  
 587           POSIX2\_SW\_DEV  
 588           POSIX2\_UPE  
 589           XOPEN\_UNIX  
 590           XOPEN\_UUCP

591           Additional language bindings and development utility options may be provided in other related  
 592           standards or in a future version of this standard. In the former case, additional symbolic  
 593           constants of the same general form as shown in this subsection should be defined by the related  
 594           standard document and made available to the application without requiring POSIX.1-2024 to be  
 595           updated.

## 596 **2.1.4 XSI Conformance**

597 XSI       This section describes the criteria for implementations providing conformance to the X/Open  
 598           System Interfaces (XSI) option (see [Section 3.424](#), on page 93). The functionality described in this  
 599           section shall be provided on implementations that support the XSI option (and the rest of this  
 600           section is not further shaded).

601           POSIX.1-2024 describes utilities, functions, and facilities offered to application programs by the  
 602           X/Open System Interfaces (XSI) option. An XSI-conforming implementation shall meet the  
 603           criteria for POSIX conformance and the following requirements listed in this section.

604           XSI-conforming implementations shall set the symbolic constant `_XOPEN_UNIX` to a value  
 605           other than `-1` and shall set the symbolic constant `_XOPEN_VERSION` to the value `800`.

### 606 *2.1.4.1 XSI System Interfaces*

607           The following requirements apply to the system interfaces when the XSI option is supported:

- 608           • The system shall support all the functions and headers defined in POSIX.1-2024 as part of  
 609           the XSI option denoted by the XSI marking in the SYNOPSIS section, and any extensions  
 610           marked with the XSI option marking (see [Section 1.8.1](#), on page 7) within the text.
- 611           • The system shall support the following options defined within POSIX.1-2024 (see [Section](#)  
 612           [2.1.6](#), on page 25):

613                    `_POSIX_FSYNC`  
 614                    `_POSIX_THREAD_ATTR_STACKADDR`  
 615                    `_POSIX_THREAD_ATTR_STACKSIZE`  
 616                    `_POSIX_THREAD_PROCESS_SHARED`

- 617           • The system may support the following XSI Option Groups (see [Section 2.1.5.2](#), on page 22)  
 618           defined within POSIX.1-2024:

- 619                   — Encryption
- 620                   — Realtime
- 621                   — Advanced Realtime

- 622 — Realtime Threads
- 623 — Advanced Realtime Threads

#### 624 2.1.4.2 XSI Shell and Utilities Conformance

625 The following requirements apply to the shell and utilities when the XSI option is supported:

- 626 • The system shall support all the utilities defined in the Shell and Utilities volume of  
627 POSIX.1-2024 as part of the XSI option denoted by the XSI marking in the SYNOPSIS  
628 section, and any extensions marked with the XSI option marking (see [Section 1.8.1](#), on  
629 page 7) within the text.
- 630 • The system shall support the User Portability Utilities option and the Terminal  
631 Characteristics option.
- 632 • The system shall support creation of locales (see [Chapter 7](#), on page 127).
- 633 • The C-language Development utility *c17* shall be supported.
- 634 • The XSI Development Utilities option may be supported. It consists of the following  
635 software development utilities:

636	<i>admin</i>	<i>delta</i>	<i>rmdel</i>	<i>val</i>
637	<i>cflow</i>	<i>get</i>	<i>sact</i>	<i>what</i>
638	<i>ctags</i>	<i>nm</i>	<i>sccs</i>	
639	<i>cxref</i>	<i>prs</i>	<i>unget</i>	

### 640 2.1.5 Option Groups

641 An Option Group is a group of related functions or options defined within the System Interfaces  
642 volume of POSIX.1-2024.

643 If an implementation supports an Option Group, then the system shall support the functional  
644 behavior described herein.

645 If an implementation does not support an Option Group, then the system need not support the  
646 functional behavior described herein.

#### 647 2.1.5.1 Subprofiling Considerations

648 Profiling standards supporting functional requirements less than that required in POSIX.1-2024  
649 may subset both mandatory and optional functionality required for POSIX Conformance (see  
650 [Section 2.1.3](#), on page 17) or XSI Conformance (see [Section 2.1.4](#), on page 19). Such profiles shall  
651 organize the subsets into Subprofiling Option Groups.

652 XRAT [Appendix E](#) (on page 3943) describes a representative set of such Subprofiling Option  
653 Groups for use by profiles applicable to specialized realtime systems. POSIX.1-2024 does not  
654 require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols  
655 defined in any header) or at runtime (via *sysconf()* or *getconf()*).

656 A Subprofiling Option Group may provide basic system functionality that other Subprofiling  
657 Option Groups and other options depend upon.<sup>4</sup> If a profile of POSIX.1-2024 does not require an  
658 implementation to provide a Subprofiling Option Group that provides features utilized by a  
659 required Subprofiling Option Group (or option),<sup>5</sup> the profile shall specify<sup>6</sup> all of the following:

- 660 • Restricted or altered behavior of interfaces defined in POSIX.1-2024 that may differ on an  
661 implementation of the profile
- 662 • Additional behaviors that may produce undefined or unspecified results
- 663 • Additional implementation-defined behavior that implementations shall be required to  
664 document in the profile's conformance document

665 if any of the above is a result of the profile not requiring an interface required by POSIX.1-2024.

666 The following additional rules shall apply to all profiles of POSIX.1-2024:

- 667 • Any application that conforms to that profile shall also conform to POSIX.1-2024, unless  
668 the application depends on the definition of a profile support indicator macro in  
669 **<unistd.h>** (that is, a profile shall not require restricted, altered, or extended behaviors of  
670 an implementation of POSIX.1-2024).
- 671 • Profiles are permitted to require the definition of a *profile support indicator macro* with a  
672 name beginning `_POSIX_AEP_` in **<unistd.h>**.
- 673 • Profiles shall require the definition of the macro `_POSIX_SUBPROFILE` in **<unistd.h>** on  
674 implementations that do not meet all of the requirements of a POSIX.1-conforming  
675 implementation.
- 676 • Profiles are permitted to add additional requirements to the limits defined in **<limits.h>**  
677 and **<stdint.h>**, subject to the following:

678 For the limits in **<limits.h>** and **<stdint.h>**:

- 679 — If the limit is specified as having a fixed value, it shall not be changed by a profile.
- 680 — If a limit is specified as having a minimum or maximum acceptable value, it may be  
681 changed by a profile as follows:
  - 682 — A profile may increase a minimum acceptable value, but shall not make a  
683 minimum acceptable value smaller.
  - 684 — A profile may reduce a maximum acceptable value, but shall not make a  
685 maximum acceptable value larger.
- 686 • A profile shall not change a limit specified as having a minimum or maximum value into a  
687 limit specified as having a fixed value.
- 688 • A profile shall not create new limits.
- 689 • Any implementation that conforms to POSIX.1-2024 (including all options and extended  
690 limits required by the profile) shall also conform to that profile, except for the possible  
691 omission from **<unistd.h>** of a profile support indicator macro required by the profile.

- 
- 692 4. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are  
693 needed by any interface in POSIX.1-2024 that parses a *path* argument. If a profile requires support for the Device Input and Output  
694 profiling option group but does not require support for the File System profiling option group, the profile needs to specify how pathname  
695 resolution is to behave in that profile, how the `O_CREAT` flag to `open()` is to be handled (and the use of the character 'a' in the *mode*  
696 argument of `fopen()` when a pathname argument names a file that does not exist), and specify lots of other details.
  - 697 5. As an example, POSIX.1-2024 requires that implementations claiming to support the Range Memory Locking option also support the  
698 Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that  
699 the Process Memory Locking option be supplied as long as the profile specifies everything an application developer or system implementor  
700 would have to know to build an application or implementation conforming to the profile.
  - 701 6. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or unspecified  
702 results.

## 703 2.1.5.2 XSI Option Groups

704 XSI This section describes Option Groups to support the definition of XSI conformance within the  
 705 System Interfaces volume of POSIX.1-2024. The functionality described in this section shall be  
 706 provided on implementations that support the XSI option and the appropriate Option Group  
 707 (and the rest of this section is not further shaded).

708 The following Option Groups are defined.

709 **Encryption**

710 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes  
 711 the following functions:

712 OB `crypt()`, `encrypt()`, `setkey()`

713 These functions are marked CRYPT.

714 Due to export restrictions on the cryptographic algorithm in some countries, implementations  
 715 may be restricted in making these functions available. All the functions in the Encryption  
 716 Option Group may therefore return `[ENOSYS]` or, alternatively, `encrypt()` shall return `[ENOSYS]`  
 717 for the decryption operation.

718 An implementation that claims conformance to this Option Group shall set `_XOPEN_CRYPT` to  
 719 a value other than `-1`.

720 **Realtime**

721 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.

722 This Option Group includes a set of realtime functions drawn from options within POSIX.1-2024  
 723 (see [Section 2.1.6](#), on page 25).

724 Where entire functions are included in the Option Group, the NAME section is marked with  
 725 REALTIME. Where additional semantics have been added to existing pages, the new material is  
 726 identified by use of the appropriate margin legend for the underlying option defined within  
 727 POSIX.1-2024.

728 An implementation that claims conformance to this Option Group shall set  
 729 `_XOPEN_REALTIME` to a value other than `-1`.

730 This Option Group consists of the set of the following options from within POSIX.1-2024 (see  
 731 [Section 2.1.6](#), on page 25):

732 `_POSIX_FSYNC`  
 733 `_POSIX_MEMLOCK`  
 734 `_POSIX_MEMLOCK_RANGE`  
 735 `_POSIX_MESSAGE_PASSING`  
 736 `_POSIX_PRIORITIZED_IO`  
 737 `_POSIX_PRIORITY_SCHEDULING`  
 738 `_POSIX_SHARED_MEMORY_OBJECTS`  
 739 `_POSIX_SYNCHRONIZED_IO`

740 If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the  
 741 following symbolic constants shall be defined by the implementation to have the value 202405L:

742            \_POSIX\_MEMLOCK  
 743            \_POSIX\_MEMLOCK\_RANGE  
 744            \_POSIX\_MESSAGE\_PASSING  
 745            \_POSIX\_PRIORITY\_SCHEDULING  
 746            \_POSIX\_SHARED\_MEMORY\_OBJECTS  
 747            \_POSIX\_SYNCHRONIZED\_IO

748            The functionality associated with `_POSIX_FSYNC` shall always be supported on XSI-conformant  
 749            systems.

750            Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If  
 751            `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by  
 752            `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling  
 753            priority equal to a base scheduling priority minus `aioctx->aio_reqprio`. If Thread Execution  
 754            Scheduling is not supported, then the base scheduling priority is that of the calling process;  
 755            otherwise, the base scheduling priority is that of the calling thread. The implementation shall  
 756            also document for which files I/O prioritization is supported.

### 757            **Advanced Realtime**

758            An implementation that claims conformance to this Option Group shall also support the  
 759            Realtime Option Group.

760            Where entire functions are included in the Option Group, the NAME section is marked with  
 761            ADVANCED REALTIME. Where additional semantics have been added to existing pages, the  
 762            new material is identified by use of the appropriate margin legend for the underlying option  
 763            defined within POSIX.1-2024.

764            This Option Group consists of the set of the following options from within POSIX.1-2024 (see  
 765            [Section 2.1.6](#), on page 25):

766            \_POSIX\_ADVISORY\_INFO  
 767            \_POSIX\_CPUTIME  
 768            \_POSIX\_SPAWN  
 769            \_POSIX\_SPORADIC\_SERVER  
 770            \_POSIX\_TYPED\_MEMORY\_OBJECTS

771            If the implementation supports the Advanced Realtime Option Group, then the following  
 772            symbolic constants shall be defined by the implementation to have the value 202405L:

773            \_POSIX\_ADVISORY\_INFO  
 774            \_POSIX\_CPUTIME  
 775            \_POSIX\_SPAWN  
 776            \_POSIX\_SPORADIC\_SERVER  
 777            \_POSIX\_TYPED\_MEMORY\_OBJECTS

778            If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant  
 779            `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the  
 780            value 202405L.

781 **Realtime Threads**

782 The Realtime Threads Option Group is denoted by the symbolic constant  
783 `_XOPEN_REALTIME_THREADS`.

784 This Option Group consists of the set of the following options from within POSIX.1-2024 (see  
785 [Section 2.1.6](#), on page 25):

```
786     _POSIX_THREAD_PRIO_INHERIT
787     _POSIX_THREAD_PRIO_PROTECT
788     _POSIX_THREAD_PRIORITY_SCHEDULING
789     _POSIX_THREAD_ROBUST_PRIO_INHERIT
790     _POSIX_THREAD_ROBUST_PRIO_PROTECT
```

791 Where applicable, whole pages are marked `REALTIME_THREADS`, together with the  
792 appropriate option margin legend for the SYNOPSIS section (see [Section 1.8.1](#), on page 7).

793 An implementation that claims conformance to this Option Group shall set  
794 `_XOPEN_REALTIME_THREADS` to a value other than `-1`.

795 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the  
796 following options shall also be defined by the implementation to have the value `202405L`:

```
797     _POSIX_THREAD_PRIO_INHERIT
798     _POSIX_THREAD_PRIO_PROTECT
799     _POSIX_THREAD_PRIORITY_SCHEDULING
800     _POSIX_THREAD_ROBUST_PRIO_INHERIT
801     _POSIX_THREAD_ROBUST_PRIO_PROTECT
```

802 **Advanced Realtime Threads**

803 An implementation that claims conformance to this Option Group shall also support the  
804 Realtime Threads Option Group.

805 Where entire functions are included in the Option Group, the NAME section is marked with  
806 `ADVANCED_REALTIME_THREADS`. Where additional semantics have been added to existing  
807 pages, the new material is identified by use of the appropriate margin legend for the underlying  
808 option defined within POSIX.1-2024.

809 This Option Group consists of the set of the following options from within POSIX.1-2024 (see  
810 [Section 2.1.6](#), on page 25):

```
811     _POSIX_THREAD_CPUTIME
812     _POSIX_THREAD_SPORADIC_SERVER
```

813 If the symbolic constant `_POSIX_THREAD_SPORADIC_SERVER` is defined to have the value  
814 `202405L`, then the symbolic constant `_POSIX_THREAD_PRIORITY_SCHEDULING` shall also be  
815 defined by the implementation to have the value `202405L`.

816 If the implementation supports the Advanced Realtime Threads Option Group, then the  
817 following symbolic constants shall be defined by the implementation to have the value `202405L`:

```
818     _POSIX_THREAD_CPUTIME
819     _POSIX_THREAD_SPORADIC_SERVER
```



## 820 2.1.6 Options

821 The symbolic constants defined in `<unistd.h>`, [Constants for Options and Option Groups](#) (on  
822 page 458) reflect implementation options for POSIX.1-2024. These symbols can be used by the  
823 application to determine which of three categories of support for optional facilities are provided  
824 by the implementation.

825 1. Option not supported for compilation.

826 The implementation advertises at compile time (by defining the constant in `<unistd.h>`  
827 with value `-1`, or by leaving it undefined) that the option is not supported for compilation  
828 and, at the time of compilation, is not supported for runtime use. In this case, the headers,  
829 data types, function interfaces, and utilities required only for the option need not be  
830 present. A later runtime check using the `fpathconf()`, `pathconf()`, or `sysconf` functions  
831 defined in the System Interfaces volume of POSIX.1-2024 or the `getconf` utility defined in  
832 the Shell and Utilities volume of POSIX.1-2024 can in some circumstances indicate that  
833 the option is supported at runtime. (For example, an old application binary might be run  
834 on a newer implementation to which support for the option has been added.)

835 2. Option always supported.

836 The implementation advertises at compile time (by defining the constant in `<unistd.h>`  
837 with a value greater than zero) that the option is supported both for compilation and for  
838 use at runtime. In this case, all headers, data types, function interfaces, and utilities  
839 required only for the option shall be available and shall operate as specified. Runtime  
840 checks with `fpathconf()`, `pathconf()`, or `sysconf` shall indicate that the option is supported.

841 3. Option might or might not be supported at runtime.

842 The implementation advertises at compile time (by defining the constant in `<unistd.h>`  
843 with value zero) that the option is supported for compilation and might or might not be  
844 supported at runtime. In this case, the `fpathconf()`, `pathconf()`, or `sysconf()` functions  
845 defined in the System Interfaces volume of POSIX.1-2024 or the `getconf` utility defined in  
846 the Shell and Utilities volume of POSIX.1-2024 can be used to retrieve the value of each  
847 symbol on each specific implementation to determine whether the option is supported at  
848 runtime. All headers, data types, and function interfaces required to compile and execute  
849 applications which use the option at runtime (after checking at runtime that the option is  
850 supported) shall be provided, but if the option is not supported at runtime they need not  
851 operate as specified. Utilities or other facilities required only for the option, but not  
852 needed to compile and execute such applications, need not be present.

853 If an option is not supported for compilation, an application that attempts to use anything  
854 associated only with the option is considered to be requiring an extension. Unless explicitly  
855 specified otherwise, the behavior of functions associated with an option that is not supported at  
856 runtime is unspecified, and an application that uses such functions without first checking  
857 `fpathconf()`, `pathconf()`, or `sysconf` is considered to be requiring an extension.

858 Margin codes are defined for each option (see [Section 1.8.1](#), on page 7).

### 859 2.1.6.1 System Interfaces

860 Refer to `<unistd.h>`, [Constants for Options and Option Groups](#) (on page 458) for the list of  
861 options.

862 2.1.6.2 *Shell and Utilities*

863 Each of these symbols shall be considered valid names by the implementation. Refer to  
864 <**unistd.h**>, [Constants for Options and Option Groups](#) (on page 458).

865 The literal names shown below apply only to the *getconf* utility.

866 CD **POSIX2\_C\_DEV**

867 The system supports the C-Language Development Utilities option.

868 The utilities in the C-Language Development Utilities option are used for the development  
869 of C-language applications, including compilation or translation of C source code and  
870 complex program generators for simple lexical tasks and processing of context-free  
871 grammars.

872 The utilities listed below may be provided by a conforming system; however, any system  
873 claiming conformance to the C-Language Development Utilities option shall provide all of  
874 the utilities listed.

875 *c17*  
876 *lex*  
877 *yacc*

878 **POSIX2\_CHAR\_TERM**

879 The system supports the Terminal Characteristics option. This value need not be present on  
880 a system not supporting the User Portability Utilities option.

881 Where applicable, the dependency is noted within the description of the utility.

882 This option applies only to systems supporting the User Portability Utilities option. If  
883 supported, then the system supports at least one terminal type capable of all operations  
884 described in POSIX.1-2024; see [Section 10.2](#) (on page 197).

885 FR **POSIX2\_FORT\_RUN**

886 The system supports the FORTRAN Runtime Utilities option.

887 The *asa* utility is the only utility in the FORTRAN Runtime Utilities option.

888 The *asa* utility may be provided by a conforming system; however, any system claiming  
889 conformance to the FORTRAN Runtime Utilities option shall provide the *asa* utility.

890 **POSIX2\_LOCALEDEF**

891 The system supports the Locale Creation Utilities option.

892 If supported, the system supports the creation of locales as described in the *localedef* utility.

893 The *localedef* utility may be provided by a conforming system; however, any system  
894 claiming conformance to the Locale Creation Utilities option shall provide the *localedef*  
895 utility.

896 SD **POSIX2\_SW\_DEV**

897 The system supports the Software Development Utilities option.

898 The utilities in the Software Development Utilities option are used for the development of  
899 applications, including compilation or translation of source code, the creation and  
900 maintenance of library archives, and the maintenance of groups of inter-dependent  
901 programs.

902 The utilities listed below may be provided by the conforming system; however, any system  
903 claiming conformance to the Software Development Utilities option shall provide all of the  
904 utilities listed here.

905            *ar*  
 906            *make*  
 907            *nm*  
 908            *strip*

909 UP        **POSIX2\_UPE**

910        The system supports the User Portability Utilities option.

911        The utilities in the User Portability Utilities option shall be implemented on all systems that  
 912        claim conformance to this option, except for the *vi* utility which is noted as having features  
 913        that cannot be implemented on all terminal types; if the `POSIX2_CHAR_TERM` option is  
 914        supported, the system shall support all such features on at least one terminal type; see  
 915        [Section 10.2](#) (on page 197).

916        The list of utilities in the User Portability Utilities option is as follows:

917            *bg*   *fc*   *jobs*   *more*   *vi*  
 918            *ex*   *fg*   *man*    *talk*

919 XSI        **XOPEN\_UNIX**

920        The system supports the X/Open System Interfaces (XSI) option (see [Section 2.1.4](#), on page  
 921        19).

922 UU        **XOPEN\_UUCP**

923        The system supports the UUCP Utilities option.

924        The list of utilities in the UUCP Utilities option is as follows:

925            *uucp*  
 926            *uustat*  
 927            *uux*

928 **2.2 Application Conformance**

929        For the purposes of POSIX.1-2024, the application conformance requirements given in this  
 930        section apply.

931        All applications claiming conformance to POSIX.1-2024 shall use only language-dependent  
 932        services for the C programming language described in [Section 2.3](#) (on page 29), shall use only  
 933        the utilities and facilities defined in the Shell and Utilities volume of POSIX.1-2024, and shall fall  
 934        within one of the following categories.

935 **2.2.1 Strictly Conforming POSIX Application**

936        A Strictly Conforming POSIX Application is an application that requires only the facilities  
 937        described in POSIX.1-2024. Such an application:

- 938        1. Shall accept any implementation behavior that results from actions it takes in areas  
 939        described in POSIX.1-2024 as *implementation-defined* or *unspecified*, or where POSIX.1-2024  
 940        indicates that implementations may vary
- 941        2. Shall not perform any actions that are described as producing *undefined* results
- 942        3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-2024,  
 943        but shall not rely on any value in the range being greater than the minimums listed or  
 944        being less than the maximums listed in POSIX.1-2024

- 945 4. Shall not use facilities designated as *obsolescent*
- 946 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
947 facilities whose availability is indicated by [Section 2.1.3](#) (on page 17)
- 948 6. For the C programming language, shall not produce any output dependent on any  
949 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*  
950 *defined*, unless the System Interfaces volume of POSIX.1-2024 specifies the behavior
- 951 7. For the C programming language, shall not exceed any minimum implementation limit  
952 defined in the ISO C standard, unless the System Interfaces volume of POSIX.1-2024  
953 specifies a higher minimum implementation limit
- 954 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 202405L before  
955 any header is included
- 956 Within POSIX.1-2024, any restrictions placed upon a Conforming POSIX Application shall  
957 restrict a Strictly Conforming POSIX Application.

## 958 2.2.2 Conforming POSIX Application

### 959 2.2.2.1 ISO/IEC Conforming POSIX Application

960 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities  
961 described in POSIX.1-2024 and approved Conforming Language bindings for any ISO or IEC  
962 standard. Such an application shall include a statement of conformance that documents all  
963 options and limit dependencies, and all other ISO or IEC standards used.

### 964 2.2.2.2 <National Body> Conforming POSIX Application

965 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming  
966 POSIX Application in that it also may use specific standards of a single ISO/IEC member body  
967 referred to here as <National Body>. Such an application shall include a statement of  
968 conformance that documents all options and limit dependencies, and all other <National Body>  
969 standards used.

## 970 2.2.3 Conforming POSIX Application Using Extensions

971 A Conforming POSIX Application Using Extensions is an application that differs from a  
972 Conforming POSIX Application only in that it uses non-standard facilities that are consistent  
973 with POSIX.1-2024. Such an application shall fully document its requirements for these extended  
974 facilities, in addition to the documentation required of a Conforming POSIX Application. A  
975 Conforming POSIX Application Using Extensions shall be either an ISO/IEC Conforming  
976 POSIX Application Using Extensions or a <National Body> Conforming POSIX Application  
977 Using Extensions (see [Section 2.2.2.1](#) and [Section 2.2.2.2](#)).

## 978 2.2.4 Strictly Conforming XSI Application

979 A Strictly Conforming XSI Application is an application that requires only the facilities  
980 described in POSIX.1-2024. Such an application:

- 981 1. Shall accept any implementation behavior that results from actions it takes in areas  
982 described in POSIX.1-2024 as *implementation-defined* or *unspecified*, or where POSIX.1-2024  
983 indicates that implementations may vary
- 984 2. Shall not perform any actions that are described as producing *undefined* results
- 985 3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-2024,  
986 but shall not rely on any value in the range being greater than the minimums listed or  
987 being less than the maximums listed in POSIX.1-2024
- 988 4. Shall not use facilities designated as *obsolescent*
- 989 5. Is required to tolerate and permitted to adapt to the presence or absence of optional  
990 facilities whose availability is indicated by [Section 2.1.4](#) (on page 19)
- 991 6. For the C programming language, shall not produce any output dependent on any  
992 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*  
993 *defined*, unless the System Interfaces volume of POSIX.1-2024 specifies the behavior
- 994 7. For the C programming language, shall not exceed any minimum implementation limit  
995 defined in the ISO C standard, unless the System Interfaces volume of POSIX.1-2024  
996 specifies a higher minimum implementation limit
- 997 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 800 before any  
998 header is included

999 Within POSIX.1-2024, any restrictions placed upon a Conforming POSIX Application shall  
1000 restrict a Strictly Conforming XSI Application.

## 1001 2.2.5 Conforming XSI Application Using Extensions

1002 A Conforming XSI Application Using Extensions is an application that differs from a Strictly  
1003 Conforming XSI Application only in that it uses non-standard facilities that are consistent with  
1004 POSIX.1-2024. Such an application shall fully document its requirements for these extended  
1005 facilities, in addition to the documentation required of a Strictly Conforming XSI Application.

## 1006 2.3 Language-Dependent Services for the C Programming Language

1007 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX  
1008 conformance as described in [Section 2.1.3](#) (on page 17).

## 1009 2.4 Other Language-Related Specifications

1010 POSIX.1-2024 is currently specified in terms of the shell command language and ISO C. Bindings  
1011 to other programming languages are being developed.

1012 If conformance to POSIX.1-2024 is claimed for implementation of any programming language,  
1013 the implementation of that language shall support the use of external symbols distinct to at least  
1014 31 bytes in length in the source program text. (That is, identifiers that differ at or before the  
1015 thirty-first byte shall be distinct.) If a national or international standard governing a language  
1016 defines a maximum length that is less than this value, the language-defined maximum shall be

1017 supported. External symbols that differ only by case shall be distinct when the character set in  
1018 use distinguishes uppercase and lowercase characters and the language permits (or requires)  
1019 uppercase and lowercase characters to be distinct in external symbols.

1020

Chapter 3

1021

# Definitions

1022

For the purposes of POSIX.1-2024, the following terms and definitions apply. The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition should be referenced for terms not defined in this section.

1023

1024

1025

**Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and definitions given in this chapter are used elsewhere in text related to extensions and options, they are shaded as appropriate.

1026

1027

1028

## 3.1 Abortive Release

1029

An abrupt termination of a network connection that may result in the loss of data.

1030

## 3.2 Absolute Pathname

1031

A pathname beginning with a single or more than two <slash> characters; see also [Section 3.254](#) (on page 68).

1032

1033

**Note:** Pathname Resolution is defined in detail in [Section 4.16](#) (on page 105).

1034

## 3.3 Access Mode

1035

A particular form of access permitted to a file.

1036

## 3.4 Additional File Access Control Mechanism

1037

An implementation-defined mechanism that is layered upon the access control mechanisms defined here, but which do not grant permissions beyond those defined herein, although they may further restrict them.

1038

1039

1040

**Note:** File Access Permissions are defined in detail in [Section 4.7](#) (on page 97).

1041

## 3.5 Address Space

1042

The memory locations that can be referenced by a process or the threads of a process.

1043

## 3.6 Advisory Information

1044

An interface that advises the implementation on (portable) application behavior so that it can optimize the system.

1045

### 1046 3.7 Affirmative Response

1047 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category  
1048 keyword **yesexpr**, matching an extended regular expression in the current locale.

1049 **Note:** The *LC\_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 159).

### 1050 3.8 Alert

1051 To cause the user's terminal to give some audible or visual indication that an error or some other  
1052 event has occurred. When the standard output is directed to a terminal device, the method for  
1053 alerting the terminal user is unspecified. When the standard output is not directed to a terminal  
1054 device, the alert is accomplished by writing the alert to standard output (unless the utility  
1055 description indicates that the use of standard output produces undefined results in this case).

### 1056 3.9 Alert Character (<alert>)

1057 A character that in the output stream should cause a terminal to alert its user via a visual or  
1058 audible notification. It is the character designated by '\a' in the C language. It is unspecified  
1059 whether this character is the exact sequence transmitted to an output device by the system to  
1060 accomplish the alert function.

### 1061 3.10 Alias Name

1062 In the shell command language, a word consisting solely of alphabetic and digits from the  
1063 portable character set and any of the following characters: '!', '%', ',', '-', '@', '\_'.

1064 Implementations may allow other characters within alias names as an extension.

1065 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 117).

### 1066 3.11 Alignment

1067 A requirement that objects of a particular type be located on storage boundaries with addresses  
1068 that are particular multiples of a byte address.

1069 **Note:** See also the ISO C standard, Section 6.2.8.

### 1070 3.12 Alternate File Access Control Mechanism

1071 An implementation-defined mechanism that is independent of the access control mechanisms  
1072 defined herein, and which if enabled on a file may either restrict or extend the permissions of a  
1073 given user. POSIX.1-2024 defines when such mechanisms can be enabled and when they are  
1074 disabled.

1075 **Note:** File Access Permissions are defined in detail in [Section 4.7](#) (on page 97).



### 1076 **3.13 Alternate Signal Stack**

1077 Memory associated with a thread, established upon request by the implementation for a thread,  
1078 separate from the thread signal stack, in which signal handlers responding to signals sent to that  
1079 thread may be executed.

### 1080 **3.14 Ancillary Data**

1081 Protocol-specific, local system-specific, or optional information. The information can be both  
1082 local or end-to-end significant, header information, part of a data portion, protocol-specific, and  
1083 implementation or system-specific.

### 1084 **3.15 Angle Brackets**

1085 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase  
1086 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed,  
1087 and '`>`' immediately follows it. When describing these characters in the portable character set,  
1088 the names `<less-than-sign>` and `<greater-than-sign>` are used.

### 1089 **3.16 Anonymous Memory Object**

1090 An object that represents memory not associated with any other memory objects.

### 1091 **3.17 Apostrophe Character (<apostrophe>)**

1092 The character designated by '`\'`' in the C language, also known as the single-quote character.

### 1093 **3.18 Application**

1094 A computer program that performs some desired function.

1095 When the User Portability Utilities option is supported, requirements placed on applications  
1096 relating to the use of standard utilities shall also apply to the actions of a user who is entering  
1097 shell command language statements into an interactive shell.

### 1098 **3.19 Application Address**

1099 Endpoint address of a specific application.

### 1100 **3.20 Application Program Interface (API)**

1101 The definition of syntax and semantics for providing computer system services.

### 1102 3.21 Appropriate Privileges

1103 An implementation-defined means of associating privileges with a process with regard to the  
1104 function calls, function call options, and the commands that need special privileges. There may  
1105 be zero or more such means. These means (or lack thereof) are described in the conformance  
1106 document.

1107 **Note:** Function calls are defined in the System Interfaces volume of POSIX.1-2024, and commands are  
1108 defined in the Shell and Utilities volume of POSIX.1-2024.

### 1109 3.22 Argument

1110 In the shell command language, a parameter passed to a utility as the equivalent of a single  
1111 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,  
1112 option-arguments, or operands following the command name.

1113 **Note:** The Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213) and XCU [Section](#)  
1114 [2.9.1.4](#) (on page 2502).

1115 In the C language, an expression in a function call expression or a sequence of preprocessing  
1116 tokens in a function-like macro invocation.

### 1117 3.23 Arm (a Timer)

1118 To start a timer measuring the passage of time, enabling notifying a process when the specified  
1119 time or time interval has passed.

### 1120 3.24 Asterisk Character (<asterisk>)

1121 The character ' \* '.

### 1122 3.25 Async-Cancel-Safe Function

1123 A function that may be safely invoked by an application while the asynchronous form of  
1124 cancellation is enabled. No function is async-cancel-safe unless explicitly described as such.

### 1125 3.26 Asynchronous Events

1126 Events that occur independently of the execution of the application.

### 1127 3.27 Asynchronous Input and Output

1128 A functionality enhancement to allow an application process to queue data input and output  
1129 commands with asynchronous notification of completion.

### 1130 3.28 Async-Signal-Safe Function

1131 A function that can be called, without restriction, from signal-catching functions. Note that,  
1132 although there is no restriction on the calls themselves, for certain functions there are restrictions  
1133 on subsequent behavior after the function is called from a signal-catching function. No function  
1134 is async-signal-safe unless explicitly described as such.

1135 **Note:** Async-signal-safety is defined in detail in XSH [Section 2.4.3](#) (on page 516).

### 1136 3.29 Asynchronously-Generated Signal

1137 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals  
1138 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a  
1139 property of how the signal was generated and not a property of the signal number. All signals  
1140 may be generated asynchronously.

1141 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of POSIX.1-2024.

### 1142 3.30 Asynchronous I/O Completion

1143 For an asynchronous read or write operation, when a corresponding synchronous read or write  
1144 would have completed and when any associated status fields have been updated.

### 1145 3.31 Asynchronous I/O Operation

1146 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from  
1147 further use of the processor.

1148 This implies that the process and the I/O operation may be running concurrently.

### 1149 3.32 Atomic Operation

1150 An operation that cannot be broken up into smaller parts that could be performed separately. An  
1151 atomic operation is guaranteed to complete either fully or not at all. In the context of the  
1152 functionality provided by the `<stdatomic.h>` header, there are different types of atomic  
1153 operation that are defined in detail in [Section 4.15.1](#) (on page 100).

### 1154 3.33 Authentication

1155 The process of validating a user or process to verify that the user or process is not a counterfeit.

### 1156 3.34 Authorization

1157 The process of verifying that a user or process has permission to use a resource in the manner  
1158 requested.

1159 To assure security, the user or process would also need to be authenticated before granting  
1160 access.

### 1161 3.35 Background Job

1162 In the context of the System Interfaces volume of POSIX.1-2024, a background process group  
1163 (see [Section 3.37](#)).

1164 In the context of the shell, a job that the shell is not waiting for before it executes further  
1165 commands or, if interactive, prompts for further commands. A background job can be a job-  
1166 control background job or a non-job-control background job. A job-control background job is a  
1167 job that started execution (either in the background or the foreground) while job control was  
1168 enabled and is currently in the background. A non-job-control background job is an  
1169 asynchronous AND-OR list that started execution while job control was disabled and was  
1170 assigned a job number. An implementation need not support non-job-control background jobs;  
1171 that is, the shell may, but need not, assign job numbers to asynchronous AND-OR lists that start  
1172 execution while job control is disabled.

1173 **Note:** Asynchronous AND-OR lists are defined in detail in XCU [Section 2.9.3.1](#) (on page 2506).

1174 **Note:** See also [Section 3.158](#) (on page 54), [Section 3.180](#) (on page 57), [Section 3.181](#) (on page 57), and  
1175 [Section 3.362](#) (on page 84).

### 1176 3.36 Background Process

1177 A process that is a member of a background process group.

### 1178 3.37 Background Process Group

1179 Any process group, other than a foreground process group, that is a member of a session that  
1180 has established a connection with a controlling terminal.

1181 **Note:** See also [Section 3.35](#).

### 1182 3.38 Backquote Character

1183 The character ' ` ', also known as <grave-accent>.

### 1184 3.39 Backslash Character (<backslash>)

1185 The character designated by ' \\ ' in the C language, also known as reverse solidus.

### 1186 **3.40 Backspace Character (<backspace>)**

1187 A character that, in the output stream, should cause printing (or displaying) to occur one  
1188 column position previous to the position about to be printed. If the position about to be printed  
1189 is at the beginning of the current line, the behavior is unspecified. It is the character designated  
1190 by '\b' in the C language. It is unspecified whether this character is the exact sequence  
1191 transmitted to an output device by the system to accomplish the backspace function. The  
1192 backspace defined here is not necessarily the ERASE special character.

1193 **Note:** Special Characters are defined in detail in [Section 11.1.9](#) (on page 203).

### 1194 **3.41 Barrier**

1195 A synchronization object that allows multiple threads to synchronize at a particular point in  
1196 their execution.

### 1197 **3.42 Basename**

1198 For pathnames containing at least one filename: the final, or only, filename in the pathname. For  
1199 pathnames consisting only of <slash> characters: either '/' or '/' if the pathname consists of  
1200 exactly two <slash> characters, and '/' otherwise.

### 1201 **3.43 Basic Regular Expression (BRE)**

1202 A regular expression (see [Section 3.308](#), on page 76) used by the majority of utilities that select  
1203 strings from a set of character strings.

1204 **Note:** Basic Regular Expressions are described in detail in [Section 9.3](#) (on page 181).

### 1205 **3.44 Bind**

1206 The process of assigning a network address to an endpoint.

### 1207 **3.45 Blank Character (<blank>)**

1208 One of the characters that belong to the **blank** character class as defined via the *LC\_CTYPE*  
1209 category in the current locale. In the POSIX locale, a <blank> character is either a <tab> or a  
1210 <space>.

### 1211 **3.46 Blank Line**

1212 A line consisting solely of zero or more <blank> characters terminated by a <newline>; see also  
1213 [Section 3.120](#) (on page 48).

### 1214 **3.47 Blocked Process (or Thread)**

1215 A process (or thread) that is waiting for some condition (other than the availability of a

1216 processor) to be satisfied before it can continue execution.

### 1217 **3.48 Blocking**

1218 A property of an open file description that causes function calls associated with it to wait for the  
1219 requested action to be performed before returning.

### 1220 **3.49 Block-Mode Terminal**

1221 A terminal device operating in a mode incapable of the character-at-a-time input and output  
1222 operations described by some of the standard utilities.

1223 **Note:** Output Devices and Terminal Types are defined in detail in [Section 10.2](#) (on page 197).

### 1224 **3.50 Block Special File**

1225 A file that refers to a device. A block special file is normally distinguished from a character  
1226 special file by providing access to the device in a manner such that the hardware characteristics  
1227 of the device are not visible.

### 1228 **3.51 Braces**

1229 The characters '{' (left-curly-bracket) and '}' (right-curly-bracket). When used in the phrase  
1230 "enclosed in (curly) braces" the symbol '{' immediately precedes the object to be enclosed, and  
1231 '}' immediately follows it. When describing these characters in the portable character set, the  
1232 names <left-curly-bracket> and <left-brace> are used for '{', and <right-curly-bracket> and  
1233 <right-brace> are used for '}'.

### 1234 **3.52 Brackets**

1235 The characters '[' (left-square-bracket) and ']' (right-square-bracket). When used in the  
1236 phrase "enclosed in (square) brackets" the symbol '[' immediately precedes the object to be  
1237 enclosed, and ']' immediately follows it. When describing these characters in the portable  
1238 character set, the names <left-square-bracket> and <right-square-bracket> are used.

### 1239 **3.53 Broadcast**

1240 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and  
1241 RFC 922.

### 1242 3.54 Built-In Utility (or Built-In)

1243 A utility implemented within a shell. There are two main types of built-in utilities: special built-  
 1244 ins and regular built-ins. Unless qualified, the term “built-in” includes both types. The utilities  
 1245 referred to as special built-ins have special qualities. Regular built-ins are not required to be  
 1246 actually built into the shell on the implementation, but they usually have special command-  
 1247 search qualities, or affect the current execution environment.

1248 **Note:** Special Built-In Utilities are defined in detail in XCU [Section 2.15](#) (on page 2526).

1249 Regular Built-In Utilities are defined in detail in XCU [Section 1.6](#) (on page 2470).

### 1250 3.55 Byte

1251 An individually addressable unit of data storage that is exactly an octet, used to store a character  
 1252 or a portion of a character; see also [Section 3.58](#). A byte is composed of a contiguous sequence of  
 1253 8 bits. The least significant bit is called the “low-order” bit; the most significant is called the  
 1254 “high-order” bit.

1255 **Note:** The definition of byte from the ISO C standard is broader than the above and might  
 1256 accommodate hardware architectures with different sized addressable units than octets.

### 1257 3.56 Byte Input/Output Functions

1258 The functions that perform byte-oriented input from streams or byte-oriented output to streams:  
 1259 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *getdelim()*,  
 1260 *getline()*, *printf()*, *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1261 **Note:** Functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

### 1262 3.57 Carriage-Return Character (<carriage-return>)

1263 A character that in the output stream indicates that printing should start at the beginning of the  
 1264 same physical line in which the carriage-return occurred. It is the character designated by '\r'  
 1265 in the C language. It is unspecified whether this character is the exact sequence transmitted to an  
 1266 output device by the system to accomplish the movement to the beginning of the line.

### 1267 3.58 Character

1268 A sequence of one or more bytes representing a member of a character set.

1269 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte  
 1270 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,  
 1271 *character* here has no necessary relationship with storage space, and *byte* is used when storage  
 1272 space is discussed.

1273 See the definition of the portable character set in [Section 6.1](#) (on page 117) for a further  
 1274 explanation of the graphical representations of (abstract) characters, as opposed to character  
 1275 encodings.

**1276 3.59 Character Array**

1277 An array of elements of type **char**.

**1278 3.60 Character Class**

1279 A named set of characters sharing an attribute associated with the name of the class. The classes  
1280 and the characters that they contain are dependent on the value of the *LC\_CTYPE* category in  
1281 the current locale.

1282 **Note:** The *LC\_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 131).

**1283 3.61 Character Set**

1284 A finite set of different characters used for the representation, organization, or control of data.

**1285 3.62 Character Special File**

1286 A file that refers to a device (such as a terminal device file) or that has special properties (such as  
1287 */dev/null*).

1288 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

**1289 3.63 Character String**

1290 A contiguous sequence of characters terminated by and including the first null byte.

**1291 3.64 Child Process**

1292 A new process created (by *fork()*, *posix\_spawn()*, or *posix\_spawnp()*) by a given process. A child  
1293 process remains the child of the creating process as long as both processes continue to exist.

1294 **Note:** The *fork()*, *posix\_spawn()*, and *posix\_spawnp()* functions are defined in detail in the System  
1295 Interfaces volume of POSIX.1-2024.

**1296 3.65 Circumflex Character (<circumflex>)**

1297 The character '^'.



**1298 3.66 Clock**

1299 A software or hardware object that can be used to measure the apparent or actual passage of  
1300 time.

1301 The current value of the time measured by a clock can be queried and, possibly, set to a value  
1302 within the legal range of the clock.

**1303 3.67 Clock Jump**

1304 The difference between two successive distinct values of a clock, as observed from the  
1305 application via one of the ``get time'' operations.

**1306 3.68 Clock Tick**

1307 An interval of time; an implementation-defined number of these occur each second. Clock ticks  
1308 are one of the units that may be used to express a value found in type `clock_t`.

**1309 3.69 Code Block**

1310 In the context of the System Interfaces volume of POSIX.1-2024, a *block* as defined in the ISO C  
1311 standard.

**1312 3.70 Coded Character Set**

1313 A set of unambiguous rules that establishes a character set and the one-to-one relationship  
1314 between each character of the set and its bit representation.

**1315 3.71 Codeset**

1316 The result of applying rules that map a numeric code value to each element of a character set.  
1317 An element of a character set may be related to more than one numeric code value but the  
1318 reverse is not true. However, for state-dependent encodings the relationship between numeric  
1319 code values and elements of a character set may be further controlled by state information. The  
1320 character set may contain fewer elements than the total number of possible numeric code values;  
1321 that is, some code values may be unassigned.

1322 **Note:** Character Encoding is defined in detail in [Section 6.2](#) (on page 120).

**1323 3.72 Collating Element**

1324 The smallest entity used to determine the logical ordering of character or wide-character strings;  
1325 see also [Section 3.74](#) (on page 42). A collating element consists of either a single character, or  
1326 two or more characters collating as a single entity. The value of the `LC_COLLATE` category in the  
1327 current locale determines the current set of collating elements.

### 1328 3.73 Collation

1329 The logical ordering of character or wide-character strings according to defined precedence  
1330 rules. These rules identify a collation sequence between the collating elements, and such  
1331 additional rules that can be used to order strings consisting of multiple collating elements.

### 1332 3.74 Collation Sequence

1333 The relative order of collating elements as determined by the setting of the *LC\_COLLATE*  
1334 category in the current locale. The collation sequence is used for sorting and is determined from  
1335 the collating weights assigned to each collating element. In the absence of weights, the collation  
1336 sequence is the order in which collating elements are specified between **order\_start** and  
1337 **order\_end** keywords in the *LC\_COLLATE* category.

1338 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to  
1339 the limit {*COLL\_WEIGHTS\_MAX*}. On each level, elements may be given the same weight (at  
1340 the primary level, called an equivalence class; see also [Section 3.126](#), on page 49) or be omitted  
1341 from the sequence. Strings that collate equally using the first assigned weight (primary ordering)  
1342 are then compared using the next assigned weight (secondary ordering), and so on.

1343 **Note:** {*COLL\_WEIGHTS\_MAX*} is defined in detail in [<limits.h>](#).

### 1344 3.75 Column Position

1345 A unit of horizontal measure related to characters in a line.

1346 It is assumed that each character in a character set has an intrinsic column width independent of  
1347 any output device. Each printable character in the portable character set has a column width of  
1348 one. The standard utilities, when used as described in POSIX.1-2024, assume that all characters  
1349 have integral column widths. The column width of a character is not necessarily related to the  
1350 internal representation of the character (numbers of bits or bytes).

1351 The column position of a character in a line is defined as one plus the sum of the column widths  
1352 of the preceding characters in the line. Column positions are numbered starting from 1.

### 1353 3.76 Command

1354 A directive to the shell to perform a particular task.

1355 **Note:** Shell Commands are defined in detail in XCU [Section 2.9](#) (on page 2499).

### 1356 3.77 Command Language Interpreter

1357 An interface that interprets sequences of text input as commands. It may operate on an input  
1358 stream or it may interactively prompt and read commands from a terminal. It is possible for  
1359 applications to invoke utilities through a number of interfaces, which are collectively considered  
1360 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*  
1361 function, although *popen()* and the various forms of *exec* may also be considered to behave as  
1362 interpreters.

1363           **Note:**     The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2024.  
1364                     The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume  
1365                     of POSIX.1-2024.

### 1366   **3.78   Composite Graphic Symbol**

1367           A graphic symbol consisting of a combination of two or more other graphic symbols in a single  
1368           character position, such as a diacritical mark and a base character.

### 1369   **3.79   Condition Variable**

1370           A synchronization object which allows a thread to suspend execution, repeatedly, until some  
1371           associated predicate becomes true. A thread whose execution is suspended on a condition  
1372           variable is said to be blocked on the condition variable.

1373           There are two types of condition variable: those of type **pthread\_cond\_t** which are initialized  
1374           using *pthread\_cond\_init()* and those of type **cond\_t** which are initialized using *cond\_init()*. If an  
1375           application attempts to use the two types interchangeably (that is, pass a condition variable of  
1376           type **pthread\_cond\_t** to a function that takes a **cond\_t**, or vice versa), the behavior is undefined.

1377           **Note:**     The *pthread\_cond\_init()* and *cond\_init()* functions are defined in detail in the System Interfaces  
1378                     volume of POSIX.1-2024.

### 1379   **3.80   Connected Socket**

1380           A connection-mode socket for which a connection has been established, or a connectionless-  
1381           mode socket for which a peer address has been set. See also [Section 3.81](#), [Section 3.82](#), [Section](#)  
1382           [3.83](#), and [Section 3.342](#) (on page 81).

### 1383   **3.81   Connection**

1384           An association established between two or more endpoints for the transfer of data

### 1385   **3.82   Connection Mode**

1386           The transfer of data in the context of a connection; see also [Section 3.83](#).

### 1387   **3.83   Connectionless Mode**

1388           The transfer of data other than in the context of a connection; see also [Section 3.82](#) and [Section](#)  
1389           [3.96](#) (on page 45).

### 1390   **3.84   Control Character**

1391           A character, other than a graphic character, that affects the recording, processing, transmission,  
1392           or interpretation of text.

### 1393 **3.85 Control Operator**

1394 In the shell command language, a token that performs a control function. It is one of the  
1395 following symbols:

1396 & && ( ) ; ;; ;& newline | ||

1397 The end-of-input indicator used internally by the shell is also considered a control operator.

1398 **Note:** Token Recognition is defined in detail in XCU [Section 2.3](#) (on page 2475).

### 1399 **3.86 Controlling Process**

1400 The session leader that established the connection to the controlling terminal. If the terminal  
1401 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be  
1402 the controlling process.

### 1403 **3.87 Controlling Terminal**

1404 A terminal that is associated with a session. Each session may have at most one controlling  
1405 terminal associated with it, and a controlling terminal is associated with exactly one session.  
1406 Certain input sequences from the controlling terminal cause signals to be sent to all processes in  
1407 the foreground process group associated with the controlling terminal.

1408 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

### 1409 **3.88 Conversion Descriptor**

1410 A per-process unique value used to identify an open codeset conversion.

### 1411 **3.89 Core Image**

1412 An unspecified object of unspecified format that may be generated when a process terminates  
1413 abnormally.

### 1414 **3.90 CPU Time (Execution Time)**

1415 The time spent executing a process or thread, including the time spent executing system services  
1416 on behalf of that process or thread. The value of the CPU-time clock for a process is  
1417 implementation-defined. With this definition the sum of all the execution times of all the threads  
1418 in a process might not equal the process execution time, even in a single-threaded process,  
1419 because implementations may differ in how they account for time during context switches or for  
1420 other reasons.

### 1421 **3.91 CPU-Time Clock**

1422 A clock that measures the execution time of a particular process or thread.

**1423 3.92 CPU-Time Timer**

1424 A timer attached to a CPU-time clock.

**1425 3.93 Current Job**

1426 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There  
1427 is at most one current job; see also [Section 3.182](#) (on page 57).

**1428 3.94 Current Working Directory**

1429 See *Working Directory* in [Section 3.421](#) (on page 93).

**1430 3.95 Cursor Position**

1431 The line and column position on the screen denoted by the terminal's cursor.

**1432 3.96 Datagram**

1433 A unit of data transferred from one endpoint to another in connectionless mode service.

**1434 3.97 Data Race**

1435 A situation in which there are two conflicting actions in different threads, at least one of which is  
1436 not atomic, and neither ``happens before'' the other, where the ``happens before'' relation is  
1437 defined formally in [Section 4.15.1](#) (on page 100).

**1438 3.98 Data Segment**

1439 Memory associated with a process, that can contain dynamically allocated data.

**1440 3.99 Decimal-Point Character**

1441 See *Radix Character* in [Section 3.294](#) (on page 74).

**1442 3.100 Declaration Utility**

1443 A utility which can take arguments that cause variable assignments (of the form *varname=value*)  
1444 which will persist in the current shell environment. When the shell recognizes a declaration  
1445 utility as the command name, subsequent arguments that would be a valid variable assignment  
1446 in isolation are subject to different expansion rules (field splitting and pathname expansion are  
1447 suppressed, and tilde expansion occurs after the <equals-sign> and any unquoted <colon>).  
1448 Arguments which are not a valid variable assignment in isolation are processed according to  
1449 normal argument expansion rules.

1450 The following standard utilities are declaration utilities: *export*, *readonly*, and, under certain

1451 conditions, *command*. An implementation may provide other declaration utilities.

### 1452 **3.101 Device**

1453 A computer peripheral or an object that appears to the application as such.

### 1454 **3.102 Device ID**

1455 A non-negative integer used to identify a device.

### 1456 **3.103 Directory**

1457 A file that contains directory entries. No two directory entries in the same directory have the  
1458 same name.

### 1459 **3.104 Directory Entry (or Hard Link)**

1460 An object that associates a filename with a file. Several directory entries can associate names  
1461 with the same file.

### 1462 **3.105 Directory Stream**

1463 A sequence of all the directory entries in a particular directory. An open directory stream may be  
1464 implemented using a file descriptor.

### 1465 **3.106 Disarm (a Timer)**

1466 To stop a timer from measuring the passage of time, disabling any future process notifications  
1467 (until the timer is armed again).

### 1468 **3.107 Display**

1469 To output to the user's terminal. If the output is not directed to a terminal, the results are  
1470 undefined.

### 1471 **3.108 Display Line**

1472 A line of text on a physical device or an emulation thereof. Such a line has a maximum number  
1473 of characters which can be presented.

1474 **Note:** This may also be written as "line on the display".

### 1475 **3.109 Dollar-Sign Character (<dollar-sign>)**

1476 The character '\$'.

### 1477 3.110 Dot

1478 In the context of naming files, the filename consisting of a single <period> character ( ' . ' ).

1479 **Note:** In the context of shell special built-in utilities, see *dot* in XCU [Section 2.15](#) (on page 2526).

1480 Pathname Resolution is defined in detail in [Section 4.16](#) (on page 105).

### 1481 3.111 Dot-Dot

1482 The filename consisting solely of two <period> characters ( " . . " ).

1483 **Note:** Pathname Resolution is defined in detail in [Section 4.16](#) (on page 105).

### 1484 3.112 Dot-Po File

1485 See *Portable Messages Object Source File* in [Section 3.266](#) (on page 70).

### 1486 3.113 Double-Quote Character

1487 The character ' " ', also known as <quotation-mark>.

1488 **Note:** The “double” adjective in this term refers to the two strokes in the character glyph.  
1489 POSIX.1-2024 never uses the term “double-quote” to refer to two apostrophes or quotation-  
1490 marks.

### 1491 3.114 Downshifting

1492 The conversion of an uppercase character that has a single-character lowercase representation  
1493 into this lowercase representation.

### 1494 3.115 Driver

1495 A module that controls data transferred to and received from devices.

1496 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are  
1497 frequently written separately from the writing of the implementation. A driver may contain  
1498 processor-specific code, and therefore be non-portable.

### 1499 3.116 Effective Group ID

1500 An attribute of a process that is used in determining various permissions, including file access  
1501 permissions; see also [Section 3.165](#) (on page 55).

### 1502 3.117 Effective User ID

1503 An attribute of a process that is used in determining various permissions, including file access  
1504 permissions; see also [Section 3.408](#) (on page 91).

1505 **3.118 Eight-Bit Transparency**

1506 The ability of a software component to process 8-bit characters without modifying or utilizing  
1507 any part of the character in a way that is inconsistent with the rules of the current coded  
1508 character set.

1509 **3.119 Empty Directory**

1510 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one hard  
1511 link to it other than its own dot entry (if one exists), in dot-dot. No other hard links to the  
1512 directory can exist. It is unspecified whether an implementation can ever consider the root  
1513 directory to be empty.

1514 **3.120 Empty Line**

1515 A line consisting of only a <newline>; see also [Section 3.46](#) (on page 37).

1516 **3.121 Empty String (or Null String)**

1517 A string whose first byte is a null byte.

1518 **3.122 Empty Wide-Character String**

1519 A wide-character string whose first element is a null wide-character code.

1520 **3.123 Encoding Rule**

1521 The rules used to convert between wide-character codes and multi-byte character codes.

1522 **Note:** Stream Orientation and Encoding Rules are defined in detail in [XSH Section 2.5.2](#) (on page 524).

1523 **3.124 Entire Regular Expression**

1524 The concatenated set of one or more basic regular expressions or extended regular expressions  
1525 that make up the pattern specified for string selection.

1526 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 179).

1527 **3.125 Epoch**

1528 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time  
1529 (UTC).

1530 **Note:** See also *Seconds Since the Epoch* defined in [Section 4.19](#) (on page 107).



### 1531 **3.126 Equivalence Class**

1532 A set of collating elements with the same primary collation weight.

1533 Elements in an equivalence class are typically elements that naturally group together, such as all  
1534 accented letters based on the same base letter.

1535 The collation order of elements within an equivalence class is determined by the weights  
1536 assigned on any subsequent levels after the primary weight.

### 1537 **3.127 Era**

1538 A locale-specific method for counting and displaying years.

1539 **Note:** The *LC\_TIME* category is defined in detail in [Section 7.3.5](#) (on page 152).

### 1540 **3.128 Event Management**

1541 The mechanism that enables applications to register for and be made aware of external events  
1542 such as data becoming available for reading.

### 1543 **3.129 Executable File**

1544 A regular file acceptable as a new process image file by the equivalent of the *exec* family of  
1545 functions, and thus usable as one form of a utility. The standard utilities described as compilers  
1546 can produce executable files, but other unspecified methods of producing executable files may  
1547 also be provided. The internal format of an executable file is unspecified, but a conforming  
1548 application cannot assume an executable file is a text file.

### 1549 **3.130 Execute**

1550 To perform command search and execution actions, as defined in the Shell and Utilities volume  
1551 of POSIX.1-2024; see also [Section 3.179](#) (on page 57).

1552 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.4](#) (on page 2502).

### 1553 **3.131 Execution Time**

1554 See *CPU Time* in [Section 3.90](#) (on page 44).

### 1555 **3.132 Execution Time Monitoring**

1556 A set of execution time monitoring primitives that allow online measuring of thread and process  
1557 execution times.

### 1558 3.133 Expand

1559 In the shell command language, when not qualified, the act of applying word expansions.

1560 **Note:** Word Expansions are defined in detail in XCU [Section 2.6](#) (on page 2483).

### 1561 3.134 Extended Regular Expression (ERE)

1562 A regular expression (see also [Section 3.308](#), on page 76) that is an alternative to the Basic  
1563 Regular Expression using a more extensive syntax, occasionally used by some utilities.

1564 **Note:** Extended Regular Expressions are described in detail in [Section 9.4](#) (on page 187).

### 1565 3.135 Extended Security Controls

1566 Implementation-defined security controls allowed by the file access permission and appropriate  
1567 privileges (see also [Section 3.21](#), on page 34) mechanisms, through which an implementation can  
1568 support different security policies from those described in POSIX.1-2024.

1569 **Note:** See also *Extended Security Controls* defined in [Section 4.6](#) (on page 96).

1570 File Access Permissions are defined in detail in [Section 4.7](#) (on page 97).

### 1571 3.136 Feature Test Macro

1572 A macro used to determine whether a particular set of features is included from a header.

1573 **Note:** See also XSH [Section 2.2](#) (on page 496).

### 1574 3.137 Field

1575 In the shell command language, a unit of text that is the result of parameter expansion,  
1576 arithmetic expansion, command substitution, or field splitting. During command processing, the  
1577 resulting fields are used as the command name and its arguments.

1578 **Note:** Parameter Expansion is defined in detail in XCU [Section 2.6.2](#) (on page 2485).

1579 Arithmetic Expansion is defined in detail in XCU [Section 2.6.4](#) (on page 2490).

1580 Command Substitution is defined in detail in XCU [Section 2.6.3](#) (on page 2489).

1581 Field Splitting is defined in detail in XCU [Section 2.6.5](#) (on page 2491).

1582 For further information on command processing, see XCU [Section 2.9.1](#) (on page 2500).

### 1583 3.138 FIFO Special File (or FIFO)

1584 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

1585 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of POSIX.1-2024,  
1586 *lseek()*, *open()*, *read()*, and *write()*.

### 1587 3.139 File

1588 An object that can be written to, or read from, or both. A file has certain attributes, including  
1589 access permissions and type. File types include regular file, character special file, block special  
1590 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported  
1591 by the implementation.

### 1592 3.140 File Description

1593 See *Open File Description* in [Section 3.241](#) (on page 66).

### 1594 3.141 File Descriptor

1595 A per-process unique, non-negative integer used to identify an open file for the purpose of file  
1596 access. The values 0, 1, and 2 have special meaning and conventional uses, and are referred to as  
1597 *standard input*, *standard output*, and *standard error*, respectively. Programs usually take their input  
1598 from standard input, and write output on standard output. Diagnostic messages are usually  
1599 written on standard error. The value of a newly-created file descriptor is from zero to  
1600 {OPEN\_MAX}-1. A file descriptor can have a value greater than or equal to {OPEN\_MAX} if the  
1601 value of {OPEN\_MAX} has decreased (see *sysconf()*) since the file descriptor was opened. File  
1602 descriptors may also be used to implement message catalog descriptors and directory streams;  
1603 see also [Section 3.241](#) (on page 66).

1604 **Note:** {OPEN\_MAX} is defined in detail in [<limits.h>](#).

### 1605 3.142 File Group Class

1606 The property of a file indicating access permissions for a process related to the group  
1607 identification of a process. A process is in the file group class of a file if the process is not in the  
1608 file owner class and if the effective group ID or one of the supplementary group IDs of the  
1609 process matches the group ID associated with the file. Other members of the class may be  
1610 implementation-defined.

### 1611 3.143 File Lock

1612 Any advisory lock, including a record lock (see [Section 3.302](#), on page 76), obtained on a file for  
1613 the purpose of coordinating transactions among cooperating processes accessing the same file  
1614 with the same lock type. See also [Section 3.237](#) (on page 66) and [Section 3.289](#) (on page 74).

1615 **Note:** All file locks created by interfaces defined in this standard are record locks; however,  
1616 implementations commonly also support a file lock extension interface named *flock()*, which  
1617 creates non-record locks (that is, a file lock that can only be held on the whole file).

1618 **Note:** Advisory locks do not prevent a process with sufficient access permissions from modifying the  
1619 file without taking locks.

### 1620 **3.144 File Mode**

1621 An object containing the file mode bits and some information about the file type of a file.

1622 **Note:** File mode bits and file types are defined in detail in [<sys/stat.h>](#).

### 1623 **3.145 File Mode Bits**

1624 A file's file permission bits, set-user-ID-on-execution bit (S\_ISUID), set-group-ID-on-execution  
1625 bit (S\_ISGID), and, on directories, the restricted deletion flag bit (S\_ISVTX).

1626 **Note:** File Mode Bits are defined in detail in [<sys/stat.h>](#).

### 1627 **3.146 Filename**

1628 A sequence of bytes consisting of 1 to {NAME\_MAX} bytes used to name a file. The bytes  
1629 composing the name shall not contain the <NUL> or <slash> characters. In the context of a  
1630 pathname, each filename shall be followed by a <slash> or a <NUL> character; elsewhere, a  
1631 filename followed by a <NUL> character forms a string (but not necessarily a character string).  
1632 The filenames **dot** and **dot-dot** have special meaning. A filename is sometimes referred to as a  
1633 ``pathname component''. See also [Section 3.254](#) (on page 68).

1634 **Note:** Pathname Resolution is defined in detail in [Section 4.16](#) (on page 105).

### 1635 **3.147 Filename String**

1636 A string consisting of a filename followed by a <NUL> character.

### 1637 **3.148 File Offset**

1638 The byte position in the file where the next I/O operation begins. Each open file description  
1639 associated with a regular file, block special file, or directory has a file offset. A character special  
1640 file that does not refer to a terminal device may have a file offset. There is no file offset specified  
1641 for a pipe or FIFO.

### 1642 **3.149 File Other Class**

1643 The property of a file indicating access permissions for a process related to the user and group  
1644 identification of a process. A process is in the file other class of a file if the process is not in the  
1645 file owner class or file group class.

### 1646 **3.150 File Owner Class**

1647 The property of a file indicating access permissions for a process related to the user

1648 identification of a process. A process is in the file owner class of a file if the effective user ID of  
1649 the process matches the user ID of the file.

### 1650 **3.151 File Permission Bits**

1651 Information about a file that is used, along with other information, to determine whether a  
1652 process has read, write, or execute/search permission to a file. The bits are divided into three  
1653 parts: owner, group, and other. Each part is used with the corresponding file class of processes.  
1654 These bits are contained in the file mode.

1655 **Note:** File modes are defined in detail in [<sys/stat.h>](#).

1656 File Access Permissions are defined in detail in [Section 4.7](#) (on page 97).

### 1657 **3.152 File Serial Number**

1658 A per-file system unique identifier for a file.

### 1659 **3.153 File System**

1660 A collection of files and certain of their attributes. It provides a name space for file serial  
1661 numbers referring to those files.

### 1662 **3.154 File Type**

1663 See *File* in [Section 3.139](#) (on page 51).

### 1664 **3.155 Filter**

1665 A command whose operation consists of reading data from standard input or a list of input files  
1666 and writing data to standard output. Typically, its function is to perform some transformation  
1667 on the data stream.

### 1668 **3.156 First Open (of a File)**

1669 When a process opens a file that is not currently an open file within any process.

### 1670 **3.157 Flow Control**

1671 The mechanism employed by a communications provider that constrains a sending entity to  
1672 wait until the receiving entities can safely receive additional data without loss.

### 1673 3.158 Foreground Job

1674 In the context of the System Interfaces volume of POSIX.1-2024, a foreground process group (see  
1675 [Section 3.160](#)).

1676 In the context of the shell, a job that the shell is waiting for before it executes further commands  
1677 or, if interactive, prompts for further commands.

1678 **Note:** See also [Section 3.35](#) (on page 36), [Section 3.180](#) (on page 57), and [Section 3.362](#) (on page 84).

### 1679 3.159 Foreground Process

1680 A process that is a member of a foreground process group.

### 1681 3.160 Foreground Process Group

1682 A process group whose member processes have certain privileges, denied to processes in  
1683 background process groups, when accessing their controlling terminal. Each session that has  
1684 established a connection with a controlling terminal has at most one process group of the session  
1685 as the foreground process group of that controlling terminal.

1686 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#).

1687 **Note:** See also [Section 3.158](#).

### 1688 3.161 Foreground Process Group ID

1689 The process group ID of the foreground process group.

### 1690 3.162 Form-Feed Character (<form-feed>)

1691 A character that in the output stream indicates that printing should start on the next page of an  
1692 output device. It is the character designated by '\f' in the C language. If the form-feed is not  
1693 the first character of an output line, the result is unspecified. It is unspecified whether this  
1694 character is the exact sequence transmitted to an output device by the system to accomplish the  
1695 movement to the next page.

### 1696 3.163 Graphic Character

1697 A member of the **graph** character class of the current locale.

1698 **Note:** The **graph** character class is defined in detail in [Section 7.3.1](#) (on page 131).

### 1699 3.164 Group Database

1700 A system database that contains at least the following information for each group ID:

- 1701 • Group name
- 1702 • Numerical group ID
- 1703 • List of users allowed in the group

1704 The list of users allowed in the group is used by the *newgrp* utility.

1705 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2024.

### 1706 3.165 Group ID

1707 A non-negative integer, which can be contained in an object of type `gid_t`, that is used to identify  
1708 a group of system users. Each system user is a member of at least one group. When the identity  
1709 of a group is associated with a process, a group ID value is referred to as a real group ID, an  
1710 effective group ID, one of the supplementary group IDs, or a saved set-group-ID. The value  
1711 (`gid_t`)-1 shall not be a valid group ID, but does have a defined use in some interfaces defined in  
1712 this standard.

### 1713 3.166 Group Name

1714 A string that is used to identify a group; see also [Section 3.164](#). To be portable across conforming  
1715 systems, the value is composed of characters from the portable filename character set. The  
1716 <hyphen-minus> should not be used as the first character of a portable group name.

### 1717 3.167 Hard Limit

1718 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.  
1719 A non-privileged process is restricted to only lowering its hard limit.

### 1720 3.168 Hard Link

1721 See Directory Entry in [Section 3.104](#) (on page 46). A file can have multiple hard links as a result  
1722 of an execution of the *ln* utility (without the `-s` option) or the *link()* function. This term is  
1723 contrasted against symbolic link; see also [Section 3.364](#) (on page 85).

### 1724 3.169 Hole

1725 A contiguous region of bytes within a file, all having the value of zero. Not all bytes with the  
1726 value zero need belong to a hole; however, all seekable files shall have a virtual hole starting at  
1727 the current size of the file. A hole is typically created via *truncate()*, or if an *lseek()* call has been  
1728 made to position beyond the end of a file and data subsequently written at that point, although  
1729 it is up to the implementation to define when sparse files can be created and with what  
1730 granularity for the size of holes.

1731 **3.170 Home Directory**

1732 The directory specified by the *HOME* environment variable.

1733 **3.171 Host Byte Order**

1734 The arrangement of bytes in any integer type when using a specific machine architecture.

1735 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian is a format  
1736 for storage of binary data in which the most significant byte is placed first, with the rest in  
1737 descending order. Little-endian is a format for storage or transmission of binary data in which  
1738 the least significant byte is placed first, with the rest in ascending order. See also [Section 4.13](#) (on  
1739 page 99).

1740 **3.172 Incomplete Line**

1741 A sequence of one or more non-`<newline>` characters at the end of the file.

1742 **3.173 Inf**

1743 A value representing +infinity or a value representing -infinity that can be stored in a floating  
1744 type. Not all systems support the Inf values.

1745 **3.174 Interactive Device**

1746 A terminal device.

1747 **Note:** This definition is intended to align with the ISO C standard's use of "interactive device".

1748 **3.175 Interactive Shell**

1749 A processing mode of the shell that is suitable for direct user interaction.

1750 **3.176 Internationalization**

1751 The provision within a computer program of the capability of making itself adaptable to the  
1752 requirements of different native languages, local customs, and coded character sets.

1753 **3.177 Interprocess Communication**

1754 A functionality enhancement to add a high-performance, deterministic interprocess  
1755 communication facility for local communication.



### 1756 3.178 Intrinsic Utility

1757 A utility that is not subject to a *PATH* search during command search, usually implemented as a  
1758 regular built-in utility.

1759 **Note:** Intrinsic Utilities are defined in detail in XCU [Section 1.7](#) (on page 2470).

### 1760 3.179 Invoke

1761 To perform command search and execution actions, except that searching for shell functions and  
1762 special built-in utilities is suppressed; see also [Section 3.130](#) (on page 49).

1763 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.4](#) (on page 2502).

### 1764 3.180 Job

1765 A background job, a foreground job, or a suspended job.

1766 In the context of the shell, jobs are created when a list (see XCU [Section 2.9.3](#), on page 2505) is  
1767 executed while job control is enabled, and may be created when an asynchronous AND-OR list  
1768 is executed while job control is disabled.

1769 **Note:** Job control in the shell is defined in detail in XCU [Section 2.11](#) (on page 2518).

1770 **Note:** See also [Section 3.35](#) (on page 36), [Section 3.158](#) (on page 54), and [Section 3.362](#) (on page 84).

### 1771 3.181 Job Control

1772 A facility that allows users selectively to stop (suspend) the execution of processes and continue  
1773 (resume) their execution at a later point. The user typically employs this facility via the  
1774 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

1775 The term is also used in connection with system interfaces that can be used by a command  
1776 interpreter to implement job control (see for example *setpgid()*).

1777 **Note:** Job control in the shell is defined in detail in XCU [Section 2.11](#) (on page 2518).

### 1778 3.182 Job ID

1779 A handle that is used to refer to a job. The job ID can be any of the forms shown in the following  
1780 table:

1781

Table 3-1 Job ID Formats

1782

1783

1784

1785

1786

1787

1788

Job ID	Meaning
%%	Current job.
%+	Current job.
%-	Previous job.
%n	Job number <i>n</i> .
%string	Job whose command begins with <i>string</i> .
%?string	Job whose command contains <i>string</i> .

1789

### 3.183 Joinable Thread

1790

1791

1792

1793

A thread that was created either using `pthread_create()` with the `detachstate` attribute not set to `PTHREAD_CREATE_DETACHED` or using `thrd_create()`, and for which neither `pthread_detach()` nor `pthread_join()` has been called and returned zero, and neither `thrd_detach()` nor `thrd_join()` has been called and returned `thrd_success`.

1794

1795

1796

**Note:** The `pthread_attr_setdetachstate()`, `pthread_create()`, `pthread_detach()`, `pthread_join()`, `thrd_create()`, `thrd_detach()`, and `thrd_join()` functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

1797

### 3.184 Last Close (of a File)

1798

When a process closes a file, resulting in the file not being an open file within any process.

1799

### 3.185 Line

1800

A sequence of zero or more non-`<newline>` characters plus a terminating `<newline>` character.

1801

### 3.186 Linger

1802

The period of time before terminating a connection, to allow outstanding data to be transferred.

1803

### 3.187 Link

1804

In the context of the file hierarchy, either a hard link or a symbolic link.

1805

In the context of the `c17` utility, the action performed by the link editor (or linker).

1806

**Note:** The `c17` utility is defined in detail in the Shell and Utilities volume of POSIX.1-2024.

1807

### 3.188 Link Count

1808

The number of directory entries that refer to a particular file.

**1809 3.189 Live Process**

1810 An address space with one or more threads executing within that address space, and the  
1811 required system resources for those threads.

1812 **Note:** Many of the system resources defined by POSIX.1-2024 are shared among all of the threads  
1813 within a process. These include the process ID, the parent process ID, process group ID, session  
1814 membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-ID,  
1815 supplementary group IDs, current working directory, root directory, file mode creation mask,  
1816 and file descriptors.

**1817 3.190 Live Thread**

1818 A single flow of control within a process. Each thread has its own thread ID, scheduling priority  
1819 and policy, *errno* value, floating point environment, thread-specific key/value bindings, and the  
1820 required system resources to support a flow of control. Anything whose address can be  
1821 determined by a thread, including but not limited to static variables, storage obtained via  
1822 *malloc()*, directly addressable storage obtained through implementation-defined functions, and  
1823 automatic variables, are accessible to all live threads in the same process.

1824 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of POSIX.1-2024.

**1825 3.191 Local Customs**

1826 The conventions of a geographical area or territory for such things as date, time, and currency  
1827 formats.

**1828 3.192 Local Interprocess Communication (Local IPC)**

1829 The transfer of data between processes in the same system.

**1830 3.193 Locale**

1831 The definition of the subset of a user's environment that depends on language and cultural  
1832 conventions.

1833 **Note:** Locales are defined in detail in [Chapter 7](#) (on page 127).

**1834 3.194 Localization**

1835 The process of establishing information within a computer system specific to the operation of  
1836 particular native languages, local customs, and coded character sets.

**1837 3.195 Lock-Free Operation**

1838 An operation that does not require the use of a lock such as a mutex in order to avoid data races.

**1839 3.196 Login**

1840 The unspecified activity by which a user gains access to the system. Each login is associated  
1841 with exactly one login name.

**1842 3.197 Login Name**

1843 A user name that is associated with a login.

**1844 3.198 Map**

1845 To create an association between a page-aligned range of the address space of a process and  
1846 some memory object, such that a reference to an address in that range of the address space  
1847 results in a reference to the associated memory object. The mapped memory object is not  
1848 necessarily memory-resident.

**1849 3.199 Matched**

1850 A state applying to a sequence of zero or more characters when the characters in the sequence  
1851 correspond to a sequence of characters defined by a basic regular expression or extended regular  
1852 expression pattern.

1853 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 179).

**1854 3.200 Memory Mapped Files**

1855 A facility to allow applications to access files as part of the address space.

**1856 3.201 Memory Object**

1857 One of:

- 1858 • A file (see [Section 3.139](#), on page 51)
- 1859 • An anonymous memory object (see [Section 3.16](#), on page 33)
- 1860 • A shared memory object (see [Section 3.332](#), on page 80)
- 1861 • A typed memory object (see [Section 3.401](#), on page 90)

1862 When used in conjunction with `mmap()`, a memory object appears in the address space of the  
1863 calling process.

1864 **Note:** The `mmap()` function is defined in detail in the System Interfaces volume of POSIX.1-2024.

**1865 3.202 Memory-Resident**

1866 The process of managing the implementation in such a way as to provide an upper bound on  
1867 memory access times.

### 1868 3.203 Message

1869 In the context of programmatic message passing, information that can be transferred between  
1870 processes or threads by being added to and removed from a message queue. A message consists  
1871 of a fixed-size message buffer.

### 1872 3.204 Message Catalog

1873 In the context of providing natural language messages to the user, a file or storage area  
1874 containing program messages, command prompts, and responses to prompts for a particular  
1875 native language, territory, and codeset.

### 1876 3.205 Message Catalog Descriptor

1877 In the context of providing natural language messages to the user, a per-process unique value  
1878 used to identify an open message catalog. A message catalog descriptor may be implemented  
1879 using a file descriptor.

### 1880 3.206 Message Queue

1881 In the context of programmatic message passing, an object to which messages can be added and  
1882 removed. Messages may be removed in the order in which they were added or in priority order.

### 1883 3.207 Messages Object

1884 A file containing message identifiers and translations in an unspecified format. Used by the  
1885 *gettext* family of functions and the *gettext* and *ngettext* utilities for internationalization and  
1886 localization of programs and scripts. Messages objects have the filename suffix **.mo**, and can be  
1887 created by the *msgfmt* utility.

1888 See also [Section 3.386](#) (on page 88).

### 1889 3.208 Mode

1890 A collection of attributes that specifies a file's type and its access permissions.

1891 **Note:** File Access Permissions are defined in detail in [Section 4.7](#) (on page 97).

### 1892 3.209 Monotonic Clock

1893 A clock measuring real time, whose value cannot be set via *clock\_settime()* and which cannot  
1894 have negative clock jumps.

### 1895 3.210 Mount Point

1896 Either the system root directory or a directory for which the *st\_dev* field of structure **stat** differs  
1897 from that of its parent directory.

1898 **Note:** The **stat** structure is defined in detail in [<sys/stat.h>](#).

### 1899 3.211 Multi-Character Collating Element

1900 A sequence of two or more characters that collate as an entity. For example, in some coded  
1901 character sets, an accented character is represented by a non-spacing accent, followed by the  
1902 letter. Other examples are the Spanish elements *ch* and *ll*.

### 1903 3.212 Multi-Threaded Library

1904 A library containing object files that were produced by compiling with *c17* using the flags  
1905 output by *getconf* POSIX\_V8\_THREADS\_CFLAGS, or by compiling using a non-standard utility  
1906 with equivalent flags, and which makes use of interfaces that are only made available by *c17*  
1907 when the **-l pthread** option is used or makes use of SIGEV\_THREAD notifications.

### 1908 3.213 Multi-Threaded Process

1909 A process that contains more than one thread.

### 1910 3.214 Multi-Threaded Program

1911 A program whose executable file was produced by compiling with *c17* using the flags output by  
1912 *getconf* POSIX\_V8\_THREADS\_CFLAGS, and linking with *c17* using the flags output by *getconf*  
1913 POSIX\_V8\_THREADS\_LDFLAGS and the **-l pthread** option, or by compiling and linking using  
1914 a non-standard utility with equivalent flags. Execution of a multi-threaded program initially  
1915 creates a single-threaded process; the process can create additional threads using  
1916 *pthread\_create()*, *thrd\_create()*, or SIGEV\_THREAD notifications.

### 1917 3.215 Mutex

1918 A synchronization object used to allow multiple threads to serialize their access to shared data.  
1919 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has  
1920 locked a mutex becomes its owner and remains the owner until that same thread unlocks the  
1921 mutex.

1922 There are two types of mutex: those of type **pthread\_mutex\_t** which are initialized using  
1923 *pthread\_mutex\_init()* and those of type **mtx\_t** which are initialized using *mtx\_init()*. If an  
1924 application attempts to use the two types interchangeably (that is, pass a mutex of type  
1925 **pthread\_mutex\_t** to a function that takes a **mtx\_t**, or vice versa), the behavior is undefined.

1926 **Note:** The *pthread\_mutex\_init()* and *mtx\_init()* functions are defined in detail in the System Interfaces  
1927 volume of POSIX.1-2024.

**1928 3.216 Name**

1929 In the shell command language, a word consisting solely of underscores, digits, and alphabets  
1930 from the portable character set. The first character of a name is not a digit.

1931 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 117).

**1932 3.217 NaN (Not a Number)**

1933 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-  
1934 point numbers. Not all systems support NaN values.

**1935 3.218 Native Language**

1936 A computer user's spoken or written language, such as American English, British English,  
1937 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

**1938 3.219 Negative**

1939 When describing a value (not a sign), less than zero. Note that in the phrase "negative zero" it  
1940 describes a sign, and therefore negative zero (also represented as -0.0) is not a negative value.

**1941 3.220 Negative Response**

1942 An input string that matches one of the responses acceptable to the *LC\_MESSAGES* category  
1943 keyword **noexpr**, matching an extended regular expression in the current locale.

1944 **Note:** The *LC\_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 159).

**1945 3.221 Network**

1946 A collection of interconnected hosts.

1947 **Note:** The term "network" in POSIX.1-2024 is used to refer to the network of hosts. The term "batch  
1948 system" is used to refer to the network of batch servers.

**1949 3.222 Network Address**

1950 A network-visible identifier used to designate specific endpoints in a network. Specific  
1951 endpoints on host systems have addresses, and host systems may also have addresses.

### 1952 3.223 Network Byte Order

1953 The way of representing any integer type such that, when transmitted over a network via a  
1954 network endpoint, the `int` type is transmitted as an appropriate number of octets with the most  
1955 significant octet first, followed by any other octets in descending order of significance.

1956 **Note:** This order is more commonly known as big-endian ordering. See also [Section 4.13](#) (on page 99).

### 1957 3.224 Newline Character (<newline>)

1958 A character that in the output stream indicates that printing should start at the beginning of the  
1959 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this  
1960 character is the exact sequence transmitted to an output device by the system to accomplish the  
1961 movement to the next line.

### 1962 3.225 Nice Value

1963 A number used as advice to the system to alter process scheduling. Numerically smaller values  
1964 give a process additional preference when scheduling a process to run. Numerically larger  
1965 values reduce the preference and make a process less likely to run. Typically, a process with a  
1966 smaller nice value runs to completion more quickly than an equivalent process with a higher  
1967 nice value. The symbol `{NZERO}` specifies the default nice value of the system.

### 1968 3.226 Non-Blocking

1969 A property of an open file description that causes function calls involving it to return without  
1970 delay when it is detected that the requested action associated with the function call cannot be  
1971 completed without unknown delay.

1972 **Note:** The exact semantics are dependent on the type of file associated with the open file description.  
1973 For data reads from devices such as ttys and FIFOs, this property causes the read to return  
1974 immediately when no data was available. Similarly, for writes, it causes the call to return  
1975 immediately when the thread would otherwise be delayed in the write operation; for example,  
1976 because no space was available. For networking, it causes functions not to await protocol events  
1977 (for example, acknowledgements) to occur. See also [XSH Section 2.10.7](#) (on page 551).

### 1978 3.227 Non-Spacing Characters

1979 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:2001  
1980 standard coded graphic character set, which is used in combination with other characters to  
1981 form composite graphic symbols.

### 1982 3.228 NUL

1983 A character with all bits set to zero.



**1984 3.229 Null Byte**

1985 A byte with all bits set to zero.

**1986 3.230 Null Pointer**

1987 A pointer obtained by converting an integer constant expression with the value 0, or such an  
1988 expression cast to type **void \***, to a pointer type; for example, **(char \*)0**. The C language  
1989 guarantees that a null pointer compares unequal to a pointer to any object or function, so it is  
1990 used by many functions that return pointers to indicate an error. POSIX.1-2024 additionally  
1991 guarantees that any pointer object whose representation has all bits set to zero, perhaps by  
1992 *memset()* to 0 or by *calloc()*, is interpreted as a null pointer.

**1993 3.231 Null String**

1994 See *Empty String* in [Section 3.121](#) (on page 48).

**1995 3.232 Null Terminator**

1996 A term used for the null byte when used as a terminator for a string.

**1997 3.233 Null Wide-Character Code**

1998 A wide-character code with all bits set to zero.

**1999 3.234 Number-Sign Character (<number-sign>)**

2000 The character '#', also known as hash sign.

**2001 3.235 Object File**

2002 A regular file containing the output of a compiler, formatted as input to a linkage editor for  
2003 linking with other object files into an executable form. The methods of linking are unspecified  
2004 and may involve the dynamic linking of objects at runtime. The internal format of an object file  
2005 is unspecified, but a conforming application cannot assume an object file is a text file.

**2006 3.236 Octet**

2007 Unit of data representation that consists of eight contiguous bits.

**2008 3.237 OFD-Owned File Lock**

2009 A record lock owned by an open file description. OFD-owned file locks are obtained through the  
2010 use of *fcntl()* with *F\_OFD\_SETLK* or *F\_OFD\_SETLKW*. Whenever a file descriptor associated  
2011 with the owning open file description is inherited these locks remain in effect. OFD-owned file  
2012 locks are automatically released on the last close of the open file description. These locks are  
2013 only shared among file descriptors associated with the same open file description. Thus, a multi-  
2014 threaded process can use multiple open file descriptions (such as by *open()*) to create  
2015 independent OFD-owned locks that can then be used to coordinate access patterns to the same  
2016 file, while multiple file descriptors associated with the same open file description (such as by  
2017 *dup()*) share lock actions among all other descriptors associated with the same open file  
2018 description.

**2019 3.238 Offset Maximum**

2020 An attribute of an open file description representing the largest value that can be used as a file  
2021 offset.

**2022 3.239 Opaque Address**

2023 An address such that the entity making use of it requires no details about its contents or format.

**2024 3.240 Open File**

2025 A file that is currently associated with a file descriptor.

**2026 3.241 Open File Description**

2027 A record of how a process or group of processes is accessing a file. Each file descriptor refers to  
2028 exactly one open file description, but an open file description can be referred to by more than  
2029 one file descriptor. The file offset, file status, and file access modes are attributes of an open file  
2030 description.

**2031 3.242 Operand**

2032 An argument to a command that is generally used as an object supplying information to a utility  
2033 necessary to complete its processing. Operands generally follow the options in a command line.

2034 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

**2035 3.243 Operator**

2036 In the shell command language, either a control operator or a redirection operator.

### 2037 **3.244 Option**

2038 An argument to a command that is generally used to specify changes in the utility's default  
2039 behavior.

2040 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

### 2041 **3.245 Option-Argument**

2042 A parameter that follows certain options. In some cases an option-argument immediately  
2043 follows the option character within the same argument string as the option; otherwise the  
2044 option-argument is the next argument string.

2045 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

### 2046 **3.246 Orientation**

2047 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2048 **Note:** For further information, see XSH [Section 2.5.2](#) (on page 524).

### 2049 **3.247 Orphaned Process Group**

2050 A process group in which the parent of every member is either itself a member of the group or is  
2051 not a member of the group's session.

### 2052 **3.248 Page**

2053 The granularity of process memory mapping or locking.

2054 Physical memory and memory objects can be mapped into the address space of a process on  
2055 page boundaries and in integral multiples of pages. Process address space can be locked into  
2056 memory (made memory-resident) on page boundaries and in integral multiples of pages.

### 2057 **3.249 Page Size**

2058 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On  
2059 systems that have segment rather than page-based memory architectures, the term "page"  
2060 means a segment.

### 2061 3.250 Parameter

2062 In the shell command language, an entity that stores values. There are three types of parameters:  
 2063 variables (named parameters), positional parameters, and special parameters. Parameter  
 2064 expansion is accomplished by introducing a parameter with the '\$' character.

2065 **Note:** See also XCU [Section 2.5](#) (on page 2478).

2066 In the C language, an object declared as part of a function declaration or definition that acquires  
 2067 a value on entry to the function, or an identifier following the macro name in a function-like  
 2068 macro definition.

### 2069 3.251 Parent Directory

2070 When discussing a given directory, the directory that both contains a directory entry for the  
 2071 given directory and is represented by the pathname dot-dot in the given directory.

2072 When discussing other types of files, a directory containing a directory entry for the file under  
 2073 discussion.

2074 This concept does not apply to dot and dot-dot.

### 2075 3.252 Parent Process

2076 The process which created (or inherited) the process under discussion.

### 2077 3.253 Parent Process ID

2078 An attribute of a new process identifying the parent of the process. The parent process ID of a  
 2079 process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime  
 2080 has ended, the parent process ID is the process ID of an implementation-defined system process.

### 2081 3.254 Pathname

2082 A string that is used to identify a file. In the context of POSIX.1-2024, a pathname may be limited  
 2083 to {PATH\_MAX} bytes, including the terminating null byte. It has optional beginning <slash>  
 2084 characters, followed by zero or more filenames separated by <slash> characters. A pathname  
 2085 can optionally contain one or more trailing <slash> characters. Multiple successive <slash>  
 2086 characters are considered to be the same as one <slash>, except it is implementation-defined  
 2087 whether the case of exactly two leading <slash> characters is treated specially.

2088 **Note:** If a pathname consists of only bytes corresponding to characters from the portable filename  
 2089 character set (see [Section 3.265](#), on page 70), <slash> characters, and a single terminating  
 2090 <NUL> character, the pathname will be usable as a character string in all supported locales;  
 2091 otherwise, the pathname might only be a string (rather than a character string). Additionally,  
 2092 since the single-byte encoding of the <slash> character is required to be the same across all  
 2093 locales and to not occur within a multi-byte character, references to a <slash> character within a  
 2094 pathname are well-defined even when the pathname is not a character string. However, this  
 2095 property does not necessarily hold for the remaining characters within the portable filename  
 2096 character set.

2097 Pathname Resolution is defined in detail in [Section 4.16](#) (on page 105).

### 2098 **3.255 Pathname Component**

2099 See *Filename* in [Section 3.146](#) (on page 52).

### 2100 **3.256 Path Prefix**

2101 The part of a pathname up to, but not including, the last component and any trailing <slash>  
2102 characters, unless the pathname consists entirely of <slash> characters, in which case the path  
2103 prefix is '/' for a pathname containing either a single <slash> or three or more <slash>  
2104 characters, and '// ' for the pathname //. The path prefix of a pathname containing no <slash>  
2105 characters is empty, but is treated as referring to the current working directory.

2106 **Note:** The term is used both in the sense of identifying part of a pathname that forms the prefix and of  
2107 joining a non-empty path prefix to a filename to form a pathname. In the latter case, the path  
2108 prefix need not have a trailing <slash> (in which case the joining is done with a <slash>  
2109 character).

### 2110 **3.257 Pattern**

2111 A sequence of characters used either with regular expression notation or with shell pattern  
2112 matching notation.

2113 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 179).

2114 Shell pattern matching notation is defined in detail in [Section 2.14](#) (on page 2523).

2115 The syntaxes of the two types of patterns are similar, but not identical; POSIX.1-2024 always  
2116 indicates the type of pattern being referred to in the immediate context of the use of the term.

### 2117 **3.258 Period Character (<period>)**

2118 The character '.'. The term "period" is contrasted with dot (see also [Section 3.110](#), on page 47),  
2119 which is used to describe a specific directory entry.

### 2120 **3.259 Permissions**

2121 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2122 **Note:** File Access Permissions are defined in detail in [Section 4.7](#) (on page 97).

### 2123 **3.260 Persistence**

2124 A mode for semaphores, shared memory, and message queues requiring that the object and its  
2125 state (including data, if any) are preserved after the object is no longer referenced by any  
2126 process.

2127 Persistence of an object does not imply that the state of the object is maintained across a system  
2128 crash or a system reboot.

### 2129 3.261 Pipe

2130 An object identical to a FIFO which has no links in the file hierarchy.

2131 **Note:** The `pipe()` function is defined in detail in the System Interfaces volume of POSIX.1-2024.

### 2132 3.262 Polling

2133 A scheduling scheme whereby the local process periodically checks until the pre-specified  
2134 events (for example, read, write) have occurred.

### 2135 3.263 Portable Character Set

2136 The collection of characters that are required to be present in all locales supported by  
2137 conforming systems.

2138 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 117).

2139 This term is contrasted against the smaller portable filename character set; see also [Section 3.265](#).

### 2140 3.264 Portable Filename

2141 A filename consisting only of characters from the portable filename character set.

2142 **Note:** Applications should avoid using filenames that have the <hyphen-minus> character as the first  
2143 character since this may cause problems when filenames are passed as command line  
2144 arguments.

### 2145 3.265 Portable Filename Character Set

2146 The set of characters from which portable filenames are constructed.

2147 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
2148 a b c d e f g h i j k l m n o p q r s t u v w x y z  
2149 0 1 2 3 4 5 6 7 8 9 . \_ -

2150 The last three characters are the <period>, <underscore>, and <hyphen-minus> characters,  
2151 respectively. See also [Section 3.254](#) (on page 68).

### 2152 3.266 Portable Messages Object Source File (or Dot-Po File)

2153 A text file containing messages and directives. A portable messages object source file can be  
2154 compiled into a messages object by the `msgfmt` utility.

2155 **Note:** By convention, portable messages object source files have filenames ending with the `.po` suffix.  
2156 Utility descriptions in this standard frequently use dot-po file as a shorthand for portable  
2157 messages object source file (even though the `.po` suffix need not be included in the filename).  
2158 Template portable messages object source files can be created from C-language source files by  
2159 the `xgettext` utility.

### 2160 3.267 Positional Parameter

2161 In the shell command language, a parameter denoted by a decimal representation of a positive  
2162 integer.

2163 **Note:** For further information, see XCU [Section 2.5.1](#) (on page 2479).

### 2164 3.268 Positive

2165 When describing a value (not a sign), greater than zero. Note that in the common phrase  
2166 “positive zero” (which is not used in this standard, although the representation +0.0 is) it  
2167 describes a sign, and therefore positive zero (+0.0) is not a positive value.

### 2168 3.269 Preallocation

2169 The reservation of resources in a system for a particular use.

2170 Preallocation does not imply that the resources are immediately allocated to that use, but merely  
2171 indicates that they are guaranteed to be available in bounded time when needed.

### 2172 3.270 Preempted Process (or Thread)

2173 A running thread whose execution is suspended due to another thread becoming runnable at a  
2174 higher priority.

### 2175 3.271 Previous Job

2176 In the context of job control, the job used as the default for the *fg* or *bg* utilities if the current job  
2177 exits. There is at most one previous job; see also [Section 3.182](#) (on page 57).

### 2178 3.272 Printable Character

2179 One of the characters included in the **print** character classification of the *LC\_CTYPE* category in  
2180 the current locale.

2181 **Note:** The *LC\_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 131).

### 2182 3.273 Printable File

2183 A text file consisting only of the characters included in the **print** and **space** character  
2184 classifications of the *LC\_CTYPE* category and the <backspace>, all in the current locale.

2185 **Note:** The *LC\_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 131).

**2186 3.274 Priority**

2187 A non-negative integer associated with processes or threads whose value is constrained to a  
2188 range defined by the applicable scheduling policy. Numerically higher values represent higher  
2189 priorities.

**2190 3.275 Priority Inversion**

2191 A condition in which a thread that is not voluntarily suspended (waiting for an event or time  
2192 delay) is not running while a lower priority thread is running. Such blocking of the higher  
2193 priority thread is often caused by contention for a shared resource.

**2194 3.276 Priority Scheduling**

2195 A performance and determinism improvement facility to allow applications to determine the  
2196 order in which threads that are ready to run are granted access to processor resources.

**2197 3.277 Priority-Based Scheduling**

2198 Scheduling in which the selection of a running thread is determined by the priorities of the  
2199 runnable processes or threads.

**2200 3.278 Privilege**

2201 See *Appropriate Privileges* in [Section 3.21](#) (on page 34).

**2202 3.279 Process**

2203 A live process (see [Section 3.189](#), on page 59) or a zombie process (see [Section 3.426](#), on page 94).  
2204 The lifetime of a process is described in [Section 3.285](#) (on page 73).

**2205 3.280 Process Group**

2206 A collection of processes that permits the signaling of related processes. Each process in the  
2207 system is a member of a process group that is identified by a process group ID. A newly created  
2208 process joins the process group of its creator.

**2209 3.281 Process Group ID**

2210 The unique positive integer identifier representing a process group during its lifetime.

2211 **Note:** See also *Process Group ID Reuse* defined in [Section 4.17](#) (on page 106).

**2212 3.282 Process Group Leader**

2213 A process whose process ID is the same as its process group ID.



### 2214 3.283 Process Group Lifetime

2215 The period of time that begins when a process group is created and ends when the last  
2216 remaining process in the group leaves the group, due either to the end of the lifetime of the last  
2217 process or to the last remaining process calling the *setsid()* or *setpgid()* functions.

2218 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of  
2219 POSIX.1-2024.

### 2220 3.284 Process ID

2221 The unique positive integer identifier representing a process during its lifetime.

2222 **Note:** See also *Process ID Reuse* defined in [Section 4.17](#) (on page 106).

### 2223 3.285 Process Lifetime

2224 The period of time that begins when a process is created and ends when its process ID is  
2225 returned to the system.

2226 See also [Section 3.189](#) (on page 59), [Section 3.287](#), and [Section 3.426](#) (on page 94).

2227 **Note:** Process creation is defined in detail in the descriptions of the *fork()*, *posix\_spawn()*, and  
2228 *posix\_spawnp()* functions in the System Interfaces volume of POSIX.1-2024.

### 2229 3.286 Process Memory Locking

2230 A performance improvement facility to bind application programs into the high-performance  
2231 random access memory of a computer system. This avoids potential latencies introduced by the  
2232 operating system in storing parts of a program that were not recently referenced on secondary  
2233 memory devices.

### 2234 3.287 Process Termination

2235 There are two kinds of process termination:

- 2236 1. Normal termination occurs by a return from *main()*, when requested with the *exit()*,  
2237 *\_exit()*, or *\_Exit()* functions; or when the last thread in the process terminates by  
2238 returning from its start function, by calling the *pthread\_exit()* or *thrd\_exit()* function, or  
2239 through cancellation.
- 2240 2. Abnormal termination occurs when requested by the *abort()* function or when some  
2241 signals are received.

2242 **Note:** The consequences of process termination can be found in the description of the *\_Exit()* function  
2243 in the System Interfaces volume of POSIX.1-2024. The *\_exit()*, *\_Exit()*, *abort()*, and *exit()*  
2244 functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

**2245 3.288 Process Virtual Time**

2246 The measurement of time in units elapsed by the system clock while a process is executing.

**2247 3.289 Process-Owned File Lock**

2248 A record lock owned by a process. Process-owned file locks are obtained through the use of  
2249 *fcntl()* with *F\_SETLK* or *F\_SETLKW*, or the use of *lockf()*. Process-owned file locks are not  
2250 inherited by child processes, but are preserved across the *exec* family of functions. A process-  
2251 owned file lock is released when the process exits, or when any file descriptor in the process  
2252 referring to the same file is closed (even if via a different open file description). These locks are  
2253 shared among all open file descriptions referring to the same file in the process, making the use  
2254 of process-owned file locks unsuitable for use for coordination of record access among multiple  
2255 threads in a process.

**2256 3.290 Process-To-Process Communication**

2257 The transfer of data between processes.

**2258 3.291 Program**

2259 A prepared sequence of instructions to the system to accomplish a defined task. The term  
2260 “program” in POSIX.1-2024 encompasses applications written in the Shell Command Language,  
2261 complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level languages.

**2262 3.292 Protocol**

2263 A set of semantic and syntactic rules for exchanging information.

**2264 3.293 Pseudo-Terminal**

2265 A facility that provides an interface that is identical to the terminal subsystem, except where  
2266 noted otherwise in POSIX.1-2024. A pseudo-terminal is composed of two devices: the “manager  
2267 device” and a “subsidiary device”. The subsidiary device provides processes with an interface  
2268 that is identical to the terminal interface, although there need not be hardware behind that  
2269 interface. Anything written on the manager device is presented to the subsidiary as an input and  
2270 anything written on the subsidiary device is presented as an input on the manager side.

**2271 3.294 Radix Character (or Decimal-Point Character)**

2272 The character that separates the integer part of a number from the fractional part.

**2273 3.295 Read-Only File System**

2274 A file system that has implementation-defined characteristics restricting modifications.

2275 **Note:** File Times Update is described in detail in [Section 4.12](#) (on page 98).

**2276 3.296 Read-Write Lock**

2277 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous  
2278 read-only access to data while allowing only one thread to have write access at any given time.  
2279 They are typically used to protect data that is read-only more frequently than it is changed.

2280 Read-write locks can be used to synchronize threads in the current process and other processes if  
2281 they are allocated in memory that is writable and shared among the cooperating processes and  
2282 have been initialized for this behavior.

**2283 3.297 Real Group ID**

2284 The attribute of a process that, at the time of process creation, identifies the group of the user  
2285 who created the process; see also [Section 3.165](#) (on page 55).

**2286 3.298 Real Time**

2287 Time measured as total units elapsed by the system clock without regard to which thread is  
2288 executing.

**2289 3.299 Realtime Signal Extension**

2290 A determinism improvement facility to enable asynchronous signal notifications to an  
2291 application to be queued without impacting compatibility with the existing signal functions.

**2292 3.300 Real User ID**

2293 The attribute of a process that, at the time of process creation, identifies the user who created the  
2294 process; see also [Section 3.408](#) (on page 91).

**2295 3.301 Record**

2296 A collection of related data units or words which is treated as a unit.

### 2297 **3.302 Record Lock**

2298 A file lock held on a record within a file. A record lock can be used to lock a whole file by  
 2299 specifying a special record with starting offset zero and length zero. (This special record extends  
 2300 to any future end-of-file, not just the current end-of-file.) This includes an OFD-owned file lock  
 2301 (see [Section 3.237](#), on page 66) or a process-owned file lock (see [Section 3.289](#), on page 74). It is  
 2302 unspecified whether an implementation will detect and prevent deadlocks caused by two  
 2303 competing lock owners holding separate locks where each tries to obtain a lock that is blocked  
 2304 by the other's lock.

### 2305 **3.303 Redirection**

2306 In the shell command language, a method of associating files with the input or output of  
 2307 commands.

2308 **Note:** For further information, see XCU [Section 2.7](#) (on page 2493).

### 2309 **3.304 Redirection Operator**

2310 In the shell command language, a token that performs a redirection function. It is one of the  
 2311 following symbols:

2312 < > >| << >> <& >& <<- <>

### 2313 **3.305 Referenced Shared Memory Object**

2314 A shared memory object that is open or has one or more mappings defined on it.

### 2315 **3.306 Refresh**

2316 Make the information on the user's terminal screen up-to-date.

### 2317 **3.307 Regular Built-In Utility (or Regular Built-In)**

2318 See *Built-In Utility* in [Section 3.54](#) (on page 39).

### 2319 **3.308 Regular Expression**

2320 A pattern that selects specific strings from a set of character strings.

2321 **Note:** Regular Expressions are described in detail in [Chapter 9](#) (on page 179).

### 2322 **3.309 Region**

2323 In the context of the address space of a process, a sequence of addresses.

2324 In the context of a file, a sequence of offsets.

**2325 3.310 Regular File**

2326 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the  
2327 system.

**2328 3.311 Relative Pathname**

2329 A pathname not beginning with a <slash> character.

2330 **Note:** Pathname Resolution is defined in detail in [Section 4.16](#) (on page 105).

**2331 3.312 Relocatable File**

2332 A file holding code or data suitable for linking with other object files to create an executable or a  
2333 shared object file.

**2334 3.313 Relocation**

2335 The process of connecting symbolic references with symbolic definitions. For example, when a  
2336 program calls a function, the associated call instruction transfers control to the proper  
2337 destination address at execution.

**2338 3.314 (Time) Resolution**

2339 The minimum time interval that a clock can measure or whose passage a timer can detect.

**2340 3.315 Robust Mutex**

2341 A mutex with the *robust* attribute set.

2342 **Note:** The *robust* attribute is defined in detail by the `pthread_mutexattr_getrobust()` function.

**2343 3.316 Root Directory**

2344 A directory, associated with a process, that is used in pathname resolution for pathnames that  
2345 begin with a <slash> character.

**2346 3.317 Runnable Process (or Thread)**

2347 A thread that is capable of being a running thread, but for which no processor is available.

**2348 3.318 Running Process (or Thread)**

2349 A thread currently executing on a processor. On multi-processor systems there may be more  
2350 than one such thread in a system at a time.

### 2351 **3.319 Saved Resource Limits**

2352 An attribute of a process that provides some flexibility in the handling of unrepresentable  
2353 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2354 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of  
2355 POSIX.1-2024.

### 2356 **3.320 Saved Set-Group-ID**

2357 An attribute of a process that allows some flexibility in the assignment of the effective group ID  
2358 attribute, as described in the *exec* family of functions and *setgid()*.

2359 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of  
2360 POSIX.1-2024.

### 2361 **3.321 Saved Set-User-ID**

2362 An attribute of a process that allows some flexibility in the assignment of the effective user ID  
2363 attribute, as described in the *exec* family of functions and *setuid()*.

2364 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of  
2365 POSIX.1-2024.

### 2366 **3.322 Scheduling**

2367 The application of a policy to select a runnable process or thread to become a running process or  
2368 thread, or to alter one or more of the thread lists.

### 2369 **3.323 Scheduling Allocation Domain**

2370 The set of processors on which an individual thread can be scheduled at any given time.

### 2371 **3.324 Scheduling Contention Scope**

2372 A property of a thread that defines the set of threads against which that thread competes for  
2373 resources.

2374 For example, in a scheduling decision, threads sharing scheduling contention scope compete for  
2375 processor resources. In POSIX.1-2024, a thread has scheduling contention scope of either  
2376 PTHREAD\_SCOPE\_SYSTEM or PTHREAD\_SCOPE\_PROCESS.

### 2377 **3.325 Scheduling Policy**

2378 A set of rules that is used to determine the order of execution of processes or threads to achieve  
2379 some goal.

2380 **Note:** Scheduling Policy is defined in detail in [Section 4.18](#) (on page 107).

### 2381 **3.326 Screen**

2382 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a  
2383 physical display device or may occupy the entire physical area of the display device.

### 2384 **3.327 Scroll**

2385 To move the representation of data vertically or horizontally relative to the terminal screen.  
2386 There are two types of scrolling:

- 2387 1. The cursor moves with the data.
- 2388 2. The cursor remains stationary while the data moves.

### 2389 **3.328 Semaphore**

2390 A minimum synchronization primitive to serve as a basis for more complex synchronization  
2391 mechanisms to be defined by the application program.

2392 **Note:** Semaphores are defined in detail in [Section 4.20](#) (on page 108).

### 2393 **3.329 Session**

2394 A collection of process groups established for job control purposes. Each process group is a  
2395 member of a session. A process is considered to be a member of the session of which its process  
2396 group is a member. A newly created process joins the session of its creator. A process can alter  
2397 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2398 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of POSIX.1-2024.

### 2399 **3.330 Session Leader**

2400 A process that has created a session.

2401 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of  
2402 POSIX.1-2024.

### 2403 **3.331 Session Lifetime**

2404 The period between when a session is created and the end of the lifetime of all the process  
2405 groups that remain as members of the session.

**2406 3.332 Shared Memory Object**

2407 An object that represents memory that can be mapped concurrently into the address space of  
2408 more than one process.

**2409 3.333 Shell**

2410 A program that interprets sequences of text input as commands. It may operate on an input  
2411 stream or it may interactively prompt and read commands from a terminal.

**2412 3.334 Shell, the**

2413 The Shell Command Language Interpreter; a specific instance of a shell.

2414 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of  
2415 POSIX.1-2024.

**2416 3.335 Shell Script**

2417 A file containing shell commands. If the file is made executable, it can be executed by specifying  
2418 its name as a simple command. Execution of a shell script causes a shell to execute the  
2419 commands within the script. Alternatively, a shell can be requested to execute the commands in  
2420 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2421 **Note:** Simple Commands are defined in detail in XCU [Section 2.9.1](#) (on page 2500).

2422 The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2024.

**2423 3.336 Signal**

2424 A mechanism by which a process or thread may be notified of, or affected by, an event occurring  
2425 in the system. Examples of such events include hardware exceptions and specific actions by  
2426 processes. The term signal is also used to refer to the event itself.

**2427 3.337 Signal Stack**

2428 Memory established for a thread, in which signal handlers catching signals sent to that thread  
2429 are executed.

**2430 3.338 Single-Quote Character**

2431 The character designated by ' \ ' ' in the C language, also known as <apostrophe>.

**2432 3.339 Single-Threaded Process**

2433 A process that contains a single thread.



### 2434 3.340 Single-Threaded Program

2435 A program whose executable file was produced by compiling with *c17* without using the flags  
 2436 output by *getconf* `POSIX_V8_THREADS_CFLAGS` and linking with *c17* using neither the flags  
 2437 output by *getconf* `POSIX_V8_THREADS_LDFLAGS` nor the `-l pthread` option, or by compiling  
 2438 and linking using a non-standard utility with equivalent flags. Execution of a single-threaded  
 2439 program creates a single-threaded process; if the process attempts to create additional threads  
 2440 using *pthread\_create()*, *thrd\_create()*, or `SIGEV_THREAD` notifications, the behavior is undefined.  
 2441 If the process uses *dlopen()* to load a multi-threaded library, the behavior is undefined.

### 2442 3.341 Slash Character (<slash>)

2443 The character ' / ', also known as solidus.

### 2444 3.342 Socket

2445 A file of a particular type that is used as a communications endpoint for process-to-process  
 2446 communication as described in the System Interfaces volume of POSIX.1-2024.

### 2447 3.343 Socket Address

2448 An address associated with a socket or remote endpoint, including an address family identifier  
 2449 and addressing information specific to that address family. The address may include multiple  
 2450 parts, such as a network address associated with a host system and an identifier for a specific  
 2451 endpoint.

### 2452 3.344 Soft Limit

2453 A resource limitation established for each process that the process may set to any value less than  
 2454 or equal to the hard limit.

### 2455 3.345 Source Code

2456 When dealing with the Shell Command Language, input to the command language interpreter.  
 2457 The term “shell script” is synonymous with this meaning.

2458 When dealing with an ISO/IEC-conforming programming language, source code is input to a  
 2459 compiler conforming to that ISO/IEC standard.

2460 Source code also refers to the input statements prepared for the following standard utilities: *awk*,  
 2461 *bc*, *ed*, *ex*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

2462 Source code can also refer to a collection of sources meeting any or all of these meanings.

2463 **Note:** The *awk*, *bc*, *ed*, *ex*, *lex*, *localedef*, *make*, *sed*, and *yacc* utilities are defined in detail in the Shell and  
 2464 Utilities volume of POSIX.1-2024.

2465 **3.346 Space Character (<space>)**

2466 The character defined in the portable character set as <space>. The <space> character is a  
 2467 member of the **space** character class of the current locale, but represents the single character, and  
 2468 not all of the possible members of the class; see also [Section 3.413](#) (on page 92).

2469 **3.347 Sparse File**

2470 A file that contains more holes than just the virtual hole at the end of the file.

2471 **3.348 Spawn**

2472 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient  
 2473 replacement for *fork()/exec*.

2474 **3.349 Special Built-In Utility (or Special Built-In)**

2475 See *Built-In Utility* in [Section 3.54](#) (on page 39).

2476 **3.350 Special Parameter**

2477 In the shell command language, a parameter named by a single character from the following list:

2478 \* @ # ? ! - \$ 0

2479 **Note:** For further information, see XCU [Section 2.5.2](#) (on page 2479).

2480 **3.351 Spin Lock**

2481 A synchronization object used to allow multiple threads to serialize their access to shared data.

2482 **3.352 Sporadic Server**

2483 A scheduling policy for threads and processes that reserves a certain amount of execution  
 2484 capacity for processing aperiodic events at a given priority level.

2485 **3.353 Standard Error**

2486 In the context of file descriptors (see [Section 3.141](#), on page 51), file descriptor number 2.

2487 In the context of standard I/O streams (see XSH [Section 2.5](#), on page 521), an output stream  
 2488 usually intended to be used for diagnostic messages, and accessed using the global variable  
 2489 *stderr*.

2490 **Note:** The file descriptor underlying *stderr* is initially 2, but it can be changed by *freopen()* to 0 or 1  
 2491 (and implementations may have extensions that allow it to be changed to other numbers).  
 2492 Therefore, writing to the standard error stream does not always produce output on the standard  
 2493 error file descriptor.

### 2494 3.354 Standard Input

2495 In the context of file descriptors (see [Section 3.141](#), on page 51), file descriptor number 0.

2496 In the context of standard I/O streams (see XSH [Section 2.5](#), on page 521), an input stream  
2497 usually intended to be used for primary data input, and accessed using the global variable *stdin*.

2498 **Note:** The file descriptor underlying *stdin* is initially 0; this cannot change through the use of  
2499 interfaces defined in this standard, but implementations may have extensions that allow it to be  
2500 changed. Therefore, in conforming applications using extensions, reading from the standard  
2501 input stream does not always obtain input from the standard input file descriptor.

### 2502 3.355 Standard Output

2503 In the context of file descriptors (see [Section 3.141](#), on page 51), file descriptor number 1.

2504 In the context of standard I/O streams (see XSH [Section 2.5](#), on page 521), an output stream  
2505 usually intended to be used for primary data output, and accessed using the global variable  
2506 *stdout*.

2507 **Note:** The file descriptor underlying *stdout* is initially 1, but it can be changed by *freopen()* to 0 (and  
2508 implementations may have extensions that allow it to be changed to other numbers). Therefore,  
2509 writing to the standard output stream does not always produce output on the standard output  
2510 file descriptor.

### 2511 3.356 Standard Utilities

2512 The utilities described in the Shell and Utilities volume of POSIX.1-2024.

### 2513 3.357 Stream

2514 Appearing in lowercase, a stream is an ordered sequence of bytes, as described by the ISO C  
2515 standard.

2516 In the shell command language, each stream is associated with a file descriptor. These can be  
2517 opened using redirection operators.

2518 **Note:** Redirection is defined in detail in XCU [Section 2.7](#) (on page 2493).

2519 In the C language, each stream is accessed via a file access object and is either a stream  
2520 associated with a file descriptor or a memory stream. A file access object associated with a file  
2521 descriptor can be created by the *fdopen()*, *fopen()*, or *popen()* functions. A file access object for a  
2522 memory stream can be created by the *fmemopen()* or *open\_memstream()* functions. A stream  
2523 provides the additional services of user-selectable buffering and formatted input and output.

2524 **Note:** For further information, see XSH [Section 2.5](#) (on page 521).

2525 The *fdopen()*, *fmemopen()*, *fopen()*, *open\_memstream()*, and *popen()* functions are defined in detail  
2526 in the System Interfaces volume of POSIX.1-2024.

### 2527 3.358 String

2528 A contiguous sequence of bytes terminated by and including the first null byte.

### 2529 3.359 Subshell

2530 A shell execution environment, distinguished from the main or current shell execution  
2531 environment.

2532 **Note:** For further information, see XCU [Section 2.13](#) (on page 2522).

### 2533 3.360 Successfully Transferred

2534 For a write operation to a regular file, when the system ensures that all data written is readable  
2535 on any subsequent open of the file (even one that follows a system or power failure) in the  
2536 absence of a failure of the physical storage medium.

2537 For a read operation, when an image of the data on the physical storage medium is available to  
2538 the requesting process.

### 2539 3.361 Supplementary Group ID

2540 An attribute of a process used in determining file access permissions. A process has up to  
2541 {NGROUPS\_MAX} supplementary group IDs in addition to the effective group ID. The  
2542 supplementary group IDs of a process are set to the supplementary group IDs of the parent  
2543 process when the process is created.

### 2544 3.362 Suspended Job

2545 In the context of the System Interfaces volume of POSIX.1-2024, a job that has received a  
2546 SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the process group to stop.

2547 In the context of the shell, a job, other than a non-job-control background job, that became  
2548 suspended when a process returned a wait status to the shell indicating that the process was  
2549 stopped by a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal.

2550 A suspended job is a job-control background job, but a job-control background job is not  
2551 necessarily a suspended job. A non-job-control background job is never a suspended job, even if  
2552 it includes processes that have been stopped by a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU  
2553 signal.

2554 **Note:** See also [Section 3.35](#) (on page 36), [Section 3.180](#) (on page 57), and [Section 3.158](#) (on page 54).

### 2555 3.363 Symbolic Constant

2556 An object-like macro defined with a constant value.

2557 Unless stated otherwise, the following shall apply to every symbolic constant:

- 2558 • It expands to a compile-time constant expression with an integer type.
- 2559 • It may be defined as another type of constant—e.g., an enumeration constant—as well as  
2560 being a macro.
- 2561 • It need not be usable in `#if` preprocessing directives.

### 2562 **3.364 Symbolic Link**

2563 A type of file with the property that when the file is encountered during pathname resolution, a  
2564 string stored by the file is used to modify the pathname resolution. The stored string has a  
2565 length of {SYMLINK\_MAX} bytes or fewer.

2566 **Note:** Pathname Resolution is defined in detail in [Section 4.16](#) (on page 105).

### 2567 **3.365 Synchronization Operation**

2568 An operation that synchronizes memory. See [Section 4.15](#) (on page 100).

### 2569 **3.366 Synchronized Input and Output**

2570 A determinism and robustness improvement mechanism to enhance the data input and output  
2571 mechanisms, so that an application can be assured that the data being manipulated is physically  
2572 present on secondary mass storage devices.

### 2573 **3.367 Synchronized I/O Completion**

2574 The state of an I/O operation that has either been successfully transferred or diagnosed as  
2575 unsuccessful.

### 2576 **3.368 Synchronized I/O Data Integrity Completion**

2577 For read, when the operation has been completed or diagnosed if unsuccessful. The read is  
2578 complete only when an image of the data has been successfully transferred to the requesting  
2579 process. If there were any pending write requests affecting the data to be read at the time that  
2580 the synchronized read operation was requested, these write requests are successfully transferred  
2581 prior to reading the data.

2582 For write, when the operation has been completed or diagnosed if unsuccessful. The write is  
2583 complete only when the data specified in the write request is successfully transferred and all file  
2584 system information required to retrieve the data is successfully transferred.

2585 For the purpose of this definition, an operation that reads or searches a directory is considered to  
2586 be a read operation, an operation that modifies a directory is considered to be a write operation,  
2587 and a directory's entries are considered to be the data read or written.

2588 This standard provides no way to synchronize the contents or attributes of a symbolic link.

2589 File attributes that are not necessary for data retrieval (access time, modification time, status  
2590 change time) need not be successfully transferred prior to returning to the calling process.

### 2591 **3.369 Synchronized I/O File Integrity Completion**

2592 Identical to a synchronized I/O data integrity completion with the addition that all file  
2593 attributes relative to the I/O operation (including access time, modification time, status change  
2594 time) are successfully transferred prior to returning to the calling process.

**2595 3.370 Synchronized I/O Operation**

2596 An I/O operation performed on a file that provides the application assurance of the integrity of  
2597 its data and files.

**2598 3.371 Synchronous I/O Operation**

2599 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the  
2600 processor until that I/O operation completes.

2601 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or  
2602 synchronized I/O file integrity completion.

**2603 3.372 Synchronously-Generated Signal**

2604 A signal that is attributable to a specific thread.

2605 For example, a thread executing an illegal instruction or touching invalid memory causes a  
2606 synchronously-generated signal. Being synchronous is a property of how the signal was  
2607 generated and not a property of the signal number.

**2608 3.373 System**

2609 An implementation of POSIX.1-2024.

**2610 3.374 System Boot**

2611 An unspecified sequence of events that may result in the loss of transitory data; that is, data that  
2612 is not saved in permanent storage. For example, message queues, shared memory, semaphores,  
2613 and processes.

**2614 3.375 System Clock**

2615 A clock with at least one second resolution that contains seconds since the Epoch.

**2616 3.376 System Console**

2617 A device that receives messages sent by the *syslog()* function, and the *fntmsg()* function when  
2618 the MM\_CONSOLE flag is set.

2619 **Note:** The *syslog()* and *fntmsg()* functions are defined in detail in the System Interfaces volume of  
2620 POSIX.1-2024.

**2621 3.377 System Crash**

2622 An interval initiated by an unspecified circumstance that causes all processes (possibly other  
2623 than special system processes) to be terminated in an undefined manner, after which any

2624 changes to the state and contents of files created or written to by an application prior to the  
2625 interval are undefined, except as required elsewhere in POSIX.1-2024.

### 2626 **3.378 System Databases**

2627 An implementation provides two system databases: the ``group database'' (see also [Section](#)  
2628 [3.164](#), on page 55) and the ``user database'' (see also [Section 3.407](#), on page 91).

### 2629 **3.379 System Documentation**

2630 All documentation provided with an implementation except for the conformance document.  
2631 Electronically distributed documents for an implementation are considered part of the system  
2632 documentation.

### 2633 **3.380 System Process**

2634 An object other than a process executing an application, that is provided by the system and has a  
2635 process ID.

### 2636 **3.381 System Reboot**

2637 See *System Boot* defined in [Section 3.374](#) (on page 86).

### 2638 **3.382 System-Wide**

2639 Pertaining to events occurring in all processes existing in an implementation at a given point in  
2640 time.

### 2641 **3.383 Tab Character (<tab>)**

2642 A character that in the output stream indicates that printing or displaying should start at the  
2643 next horizontal tabulation position on the current line. It is the character designated by '`\t`' in  
2644 the C language. If the current position is at or past the last defined horizontal tabulation  
2645 position, the behavior is unspecified. It is unspecified whether this character is the exact  
2646 sequence transmitted to an output device by the system to accomplish the tabulation.

### 2647 **3.384 Terminal (or Terminal Device)**

2648 A character special file that obeys the specifications of the general terminal interface.

2649 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

### 2650 **3.385 Text Column**

2651 A roughly rectangular block of characters capable of being laid out side-by-side next to other  
2652 text columns on an output page or terminal screen. The widths of text columns are measured in

2653 column positions.

### 2654 3.386 Text Domain

2655 A named collection of messages objects (one messages object per supported language) for  
2656 internationalization and localization purposes. A text domain is often named after the  
2657 application or library that provides the collection, but may have a more general name if it is  
2658 intended to be shared by multiple applications or libraries.

2659 **Note:** The use of text domains is defined in detail in the descriptions of the *bindtextdomain()* and  
2660 *gettext* family of functions in the System Interfaces volume of POSIX.1-2024.

### 2661 3.387 Text File

2662 A file that contains characters organized into zero or more lines. The lines do not contain NUL  
2663 characters and none can exceed {LINE\_MAX} bytes in length, including the <newline>  
2664 character. Although POSIX.1-2024 does not distinguish between text files and binary files (see  
2665 the ISO C standard), many utilities only produce predictable or meaningful output when  
2666 operating on text files. The standard utilities that have such restrictions always specify ``text  
2667 files'' in their STDIN or INPUT FILES sections.

### 2668 3.388 Thread

2669 A live thread (see [Section 3.190](#), on page 59) or a zombie thread (see [Section 3.427](#), on page 94).  
2670 The lifetime of a thread is described in [Section 3.390](#) (on page 89).

### 2671 3.389 Thread ID

2672 A value that uniquely identifies each thread in a process during the thread's lifetime. The value  
2673 shall be unique across all threads in a process, regardless of whether the thread is:

- 2674 • The initial thread
- 2675 • A thread created using *pthread\_create()*
- 2676 • A thread created using *thrd\_create()*
- 2677 • A thread created via a SIGEV\_THREAD notification

2678 **Note:** Since *pthread\_create()* returns an ID of type **pthread\_t** and *thrd\_create()* returns an ID of type  
2679 **thrd\_t**, this uniqueness requirement necessitates that these two types are defined as the same  
2680 underlying type because calls to *pthread\_self()* and *thrd\_current()* from the initial thread need to  
2681 return the same thread ID. The *pthread\_create()*, *pthread\_self()*, *thrd\_create()*, and *thrd\_current()*  
2682 functions and SIGEV\_THREAD notifications are defined in detail in the System Interfaces  
2683 volume of POSIX.1-2024.



### 2684 3.390 Thread Lifetime

2685 The period of time that begins when a thread is created and ends when its thread ID is returned  
2686 to the process.

2687 See also *Live Thread* in [Section 3.190](#) (on page 59), *Thread Termination* in [Section 3.392](#), and *Zombie*  
2688 *Thread* in [Section 3.427](#) (on page 94).

2689 **Note:** Thread creation is defined in detail in the descriptions of the `pthread_create()` and `thrd_create()`  
2690 functions in the System Interfaces volume of POSIX.1-2024.

### 2691 3.391 Thread List

2692 An ordered set of runnable threads that all have the same ordinal value for their priority.

2693 The ordering of threads on the list is determined by a scheduling policy or policies. The set of  
2694 thread lists includes all runnable threads in the system.

### 2695 3.392 Thread Termination

2696 Thread termination occurs when a thread executes `pthread_exit()` or `thrd_exit()`, when it returns  
2697 from the `start_routine` function passed to `pthread_create()` or from the `func` function passed to  
2698 `thrd_create()`, or when it acts on a cancellation request initiated by `pthread_cancel()`.

2699 **Note:** The `pthread_cancel()`, `pthread_create()`, `pthread_exit()`, `thrd_create()`, and `thrd_exit()` functions are  
2700 defined in detail in the System Interfaces volume of POSIX.1-2024.

### 2701 3.393 Thread-Safe

2702 A thread-safe function shall avoid data races with other calls to the same function, and with calls  
2703 to any other thread-safe functions, by multiple threads. Each function defined in the System  
2704 Interfaces volume of POSIX.1-2024 is thread-safe unless explicitly stated otherwise. Examples  
2705 are any “pure” function, a function which holds a mutex locked while it is accessing static  
2706 storage, or objects shared among threads.

2707 A function that is not required to be thread-safe need not avoid data races with other calls to the  
2708 same function, nor with calls to any other function (including thread-safe functions), by multiple  
2709 threads, unless explicitly stated otherwise.

### 2710 3.394 Thread-Specific Data Key

2711 A process global handle which is used for naming thread-specific data. There are two types of  
2712 key: those of type `pthread_key_t` which are created using `pthread_key_create()` and those of type  
2713 `tss_t` which are created using `tss_create()`. If an application attempts to use the two types of key  
2714 interchangeably (that is, pass a key of type `pthread_key_t` to a function that takes a `tss_t`, or vice  
2715 versa), the behavior is undefined.

2716 Although the same key value can be used by different threads, the values bound to the key by  
2717 `pthread_setspecific()` for keys of type `pthread_key_t`, and by `tss_set()` for keys of type `tss_t`, are  
2718 maintained on a per-thread basis and persist for the life of the calling thread.

2719 **Note:** The `pthread_getspecific()`, `pthread_setspecific()`, `tss_create()`, and `tss_set()` functions are defined in  
2720 detail in the System Interfaces volume of POSIX.1-2024.

### 2721 **3.395 Tilde Character (<tilde>)**

2722 The character '~'.

### 2723 **3.396 Timeouts**

2724 A method of limiting the length of time an interface will block; see also [Section 3.47](#) (on page 37).

### 2725 **3.397 Timer**

2726 A mechanism that can notify a thread when the time as measured by a particular clock has  
2727 reached or passed a specified value, or when a specified amount of time has passed.

### 2728 **3.398 Timer Overrun**

2729 A condition that occurs each time a timer, for which there is already an expiration signal queued  
2730 to the process, expires.

### 2731 **3.399 Token**

2732 In the shell command language, a sequence of characters that the shell considers as a single unit  
2733 when reading input. A token is either an operator or a word.

2734 **Note:** The rules for reading input are defined in detail in XCU [Section 2.3](#) (on page 2475).

### 2735 **3.400 Typed Memory Name Space**

2736 A system-wide name space that contains the names of the typed memory objects present in the  
2737 system. It is configurable for a given implementation.

### 2738 **3.401 Typed Memory Object**

2739 A combination of a typed memory pool and a typed memory port. The entire contents of the  
2740 pool are accessible from the port. The typed memory object is identified through a name that  
2741 belongs to the typed memory name space.

### 2742 **3.402 Typed Memory Pool**

2743 An extent of memory with the same operational characteristics. Typed memory pools may be  
2744 contained within each other.

**2745 3.403 Typed Memory Port**

2746 A hardware access path to one or more typed memory pools.

**2747 3.404 Unbind**

2748 Remove the association between a network address and an endpoint.

**2749 3.405 Unit Data**

2750 See *Datagram* in [Section 3.96](#) (on page 45).

**2751 3.406 Upshifting**

2752 The conversion of a lowercase character that has a single-character uppercase representation into  
2753 this uppercase representation.

**2754 3.407 User Database**

2755 A system database that contains at least the following information for each user ID:

- 2756 • User name
- 2757 • Numerical user ID
- 2758 • Initial numerical group ID
- 2759 • Initial working directory
- 2760 • Initial user program

2761 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under  
2762 which the initial values are operative are implementation-defined.

2763 If the initial user program field is null, an implementation-defined program is used.

2764 If the initial working directory field is null, the interpretation of that field is implementation-  
2765 defined.

2766 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2024.

**2767 3.408 User ID**

2768 A non-negative integer that is used to identify a system user. When the identity of a user is  
2769 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or  
2770 a saved set-user-ID. The value (**uid\_t**)-1 shall not be a valid user ID, but does have a defined use  
2771 in some interfaces defined in this standard.

**2772 3.409 User Name**

2773 A string that is used to identify a user; see also [Section 3.407](#). To be portable across systems  
2774 conforming to POSIX.1-2024, the value is composed of characters from the portable filename

2775 character set. The <hyphen-minus> character should not be used as the first character of a  
2776 portable user name.

### 2777 **3.410 Utility**

2778 A program, excluding special built-in utilities provided as part of the Shell Command Language,  
2779 that can be called by name from a shell to perform a specific task, or related set of tasks.

2780 **Note:** For further information on special built-in utilities, see XCU [Section 2.15](#) (on page 2526).

### 2781 **3.411 Variable**

2782 In the shell command language, a named parameter.

2783 **Note:** For further information, see XCU [Section 2.5](#) (on page 2478).

### 2784 **3.412 Vertical-Tab Character (<vertical-tab>)**

2785 A character that in the output stream indicates that printing should start at the next vertical  
2786 tabulation position. It is the character designated by '\v' in the C language. If the current  
2787 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is  
2788 unspecified whether this character is the exact sequence transmitted to an output device by the  
2789 system to accomplish the tabulation.

### 2790 **3.413 White Space**

2791 A sequence of one or more characters that belong to the **space** character class as defined via the  
2792 *LC\_CTYPE* category in the current locale or a specified locale.

2793 In the POSIX locale, white space consists of one or more <blank> (<space> and <tab>  
2794 characters), <newline>, <carriage-return>, <form-feed>, and <vertical-tab> characters.

### 2795 **3.414 White-Space Byte**

2796 A single-byte white-space character; that is, a character for which the *isspace()* or *isspace\_1()*  
2797 function returns a non-zero value.

### 2798 **3.415 White-Space Character**

2799 A character that belongs to the **space** character class as defined via the *LC\_CTYPE* category in  
2800 the current locale or a specified locale.

### 2801 **3.416 White-Space Wide Character**

2802 A wide-character code that belongs to the **space** character class as defined via the *LC\_CTYPE*  
2803 category in the current locale or a specified locale.

### 2804 3.417 Wide-Character Code (C Language)

2805 An integer value corresponding to a single graphic symbol or control code.

2806 **Note:** C Language Wide-Character Codes are defined in detail in [Section 6.3](#) (on page 120).

### 2807 3.418 Wide-Character Input/Output Functions

2808 The functions that perform wide-oriented input from streams or wide-oriented output to  
2809 streams: *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *putwc()*,  
2810 *putwchar()*, *ungetwc()*, *vwprintf()*, *vwscanf()*, *vwpprintf()*, *vwscanf()*, *wprintf()*, and *wscanf()*.

2811 **Note:** These functions are defined in detail in the System Interfaces volume of POSIX.1-2024.

### 2812 3.419 Wide-Character String

2813 A contiguous sequence of wide-character codes terminated by and including the first null wide-  
2814 character code.

### 2815 3.420 Word

2816 In the shell command language, a token other than an operator. In some cases a word is also a  
2817 portion of a word token: in the various forms of parameter expansion, such as *\${name-word}*,  
2818 and variable assignment, such as *name=word*, the word is the portion of the token depicted by  
2819 *word*. The concept of a word is no longer applicable following word expansions—only fields  
2820 remain.

2821 **Note:** For further information, see XCU [Section 2.6.2](#) (on page 2485) and [Section 2.6](#) (on page 2483).

### 2822 3.421 Working Directory (or Current Working Directory)

2823 A directory, associated with a process, that is used in pathname resolution for pathnames that do  
2824 not begin with a <slash> character.

### 2825 3.422 Worldwide Portability Interface

2826 Functions for handling characters in a codeset-independent manner.

### 2827 3.423 Write

2828 To output characters to a file, such as standard output or standard error. Unless otherwise stated,  
2829 standard output is the default output destination for all uses of the term “write”; see the  
2830 distinction between display and write in [Section 3.107](#) (on page 46).

### 2831 3.424 XSI

2832 The X/Open System Interfaces (XSI) option is the core application programming interface for C

2833 and *sh* programming for systems conforming to the Single UNIX Specification. This is a  
2834 superset of the mandatory requirements for conformance to POSIX.1-2024.

### 2835 **3.425 XSI-Conformant**

2836 A system which allows an application to be built using a set of services that are consistent across  
2837 all systems that conform to POSIX.1-2024 and that support the XSI option.

2838 **Note:** See also [Chapter 2](#) (on page 15).

### 2839 **3.426 Zombie Process**

2840 The remains of a live process (see [Section 3.189](#), on page 59) after it terminates (see [Section 3.287](#),  
2841 on page 73) and before its status information (see [XSH Section 2.12](#), on page 563) is consumed by  
2842 its parent process.

### 2843 **3.427 Zombie Thread**

2844 The remains of a joinable live thread (see [Section 3.183](#) (on page 58) and [Section 3.190](#), on page  
2845 59) after it terminates (see [Section 3.392](#), on page 89) and before it has been joined with  
2846 *pthread\_join()* or *thrd\_join()* or detached with *pthread\_detach()* or *thrd\_detach()*.

2847 **Note:** The *pthread\_detach()*, *pthread\_join()*, *thrd\_detach()*, and *thrd\_join()* functions are defined in detail  
2848 in the System Interfaces volume of POSIX.1-2024.

### 2849 **3.428 ±0**

2850 The algebraic sign provides additional information about any variable that has the value zero  
2851 when the representation allows the sign to be determined.

2852

2853

# General Concepts

2854

For the purposes of POSIX.1-2024, the general concepts given in [Chapter 4](#) apply.

2855

**Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and definitions given in this chapter are used elsewhere in text related to extensions and options, they are shaded as appropriate.

2856

2857

2858

## 4.1 Case Insensitive Comparisons

2859

When a standard utility or function that uses regular expressions or pattern matching specifies that matching shall be case insensitive, then if a string would match the regular expression or pattern when doing a case-sensitive match, the same string with any of its characters replaced with their case counterparts, as defined by the **toupper** and **tolower** character mappings (see [Section 7.3.1](#), on page 131), shall also match when doing a case-insensitive match.

2860

2861

2862

2863

2864

This definition of case-insensitive processing is intended to allow matching of multi-character collating elements as well as characters, as each character in the string is matched using both its cases. For example, in a locale with a "Ch" multi-character collating element (see [Section 9.3.2](#), on page 139), the bracket expression "[[.Ch.]]" (see [Section 9.3.5](#) (on page 182) item 4) matches the strings "ch", "Ch", "cH", and "CH" when matching without regard to case.

2865

2866

2867

2868

2869

## 4.2 Concurrent Execution

2870

Functions that suspend the execution of the calling thread shall not cause the execution of other threads to be indefinitely suspended.

2871

2872

## 4.3 Default Initialization

2873

Default initialization causes an object to be initialized according to these rules:

2874

- If it has pointer type, it is initialized to a null pointer.
- If it has arithmetic type, it is initialized to (positive or unsigned) zero.
- If it is an aggregate, every member is initialized (recursively) according to these rules.
- If it is a union, the first named member is initialized (recursively) according to these rules.

2875

2876

2877

2878

For an object of aggregate type with an explicit initializer, the initialization shall occur in initializer list order, each initializer provided for a particular subobject overriding any previously listed initializer for the same subobject; all subobjects that are not initialized explicitly shall be initialized implicitly according to the rules for default initialization.

2879

2880

2881

2882

Objects with static storage duration but no explicit initializer shall be initialized implicitly according to the rules for default initialization.

2883

2884

An explicit initializer of { 0 } works to perform explicit default initialization for any object of scalar or aggregate type, and for any storage duration.

2885

2886 **Notes:**

- 2887
- 2888
- 2889
- 2890
- 2891
- 2892
- 2893
- 2894
- 2895
- 2896
1. The ISO C standard does not require a compiler to set any field alignment padding bits in a structure or array definition to a particular value. Because of this, a structure initialized using `{ 0 }` might not *memcmp()* as equal to the same structure initialized using *memset()* to zero. For consistent results, portable applications comparing structures should test each field individually.
  2. If an implementation treats the all-zero bit pattern of a floating-point object as equivalent to positive 0, then *memset()* to zero and *calloc()* have the same effects as default initialization for all named members of a structure. Implementations that define `__STDC_IEC_559__` guarantee that the all-zero bit pattern of a floating-point object represents 0.0.

## 2897 4.4 Directory Operations

2898 All file system operations that read or search a directory or that modify the contents of a  
 2899 directory (for example creating, unlinking, or renaming a file) shall operate atomically. That is,  
 2900 each operation shall either have its entire effect and succeed, or shall not affect the file system  
 2901 and shall fail. Furthermore, these operations shall be serializable; that is, the state of the file  
 2902 system and of the results of each operation shall always be values that would be obtained if the  
 2903 operations were executed one after the other.

## 2904 4.5 Directory Protection

2905 If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove  
 2906 or rename files within that directory only if one or more of the following is true:

- 2907
- 2908
- 2909
- 2910
- 2911
- The effective user ID of the process is the same as that of the owner ID of the file.
  - The effective user ID of the process is the same as that of the owner ID of the directory.
  - The process has appropriate privileges.
  - Optionally, the file is writable by the process. Whether or not files that are writable by the process can be removed or renamed is implementation-defined.

2912 If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

## 2913 4.6 Extended Security Controls

2914 An implementation may provide implementation-defined extended security controls (see  
 2915 [Section 3.135](#), on page 50). These permit an implementation to provide security mechanisms to  
 2916 implement different security policies than those described in POSIX.1-2024. These mechanisms  
 2917 shall not alter or override the defined semantics of any of the interfaces in POSIX.1-2024.



## 2918 4.7 File Access Permissions

2919 The standard file access control mechanism uses the file permission bits, as described below.

2920 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An  
2921 additional access control mechanism shall only further restrict the access permissions defined by  
2922 the file permission bits. An alternate file access control mechanism shall:

- 2923 • Specify file permission bits for the file owner class, file group class, and file other class of  
2924 that file, corresponding to the access permissions.
- 2925 • Be enabled only by explicit user action, on a per-file basis by the file owner or a user with  
2926 appropriate privileges.
- 2927 • Be disabled for a file after the file permission bits are changed for that file with *chmod()*.  
2928 The disabling of the alternate mechanism need not disable any additional mechanisms  
2929 supported by an implementation.

2930 Whenever a process requests file access permission for read, write, or execute/search, if no  
2931 additional mechanism denies access, access shall be determined as follows:

- 2932 • If a process has appropriate privileges:
  - 2933 — If read, write, or directory search permission is requested, access shall be granted.
  - 2934 — If execute permission is requested, access shall be granted if execute permission is  
2935 granted to at least one user by the file permission bits or by an alternate access  
2936 control mechanism; otherwise, access shall be denied.
- 2937 • Otherwise:
  - 2938 — The file permission bits of a file contain read, write, and execute/search permissions  
2939 for the file owner class, file group class, and file other class.
  - 2940 — Access shall be granted if an alternate access control mechanism is not enabled and  
2941 the requested access permission bit is set for the class (file owner class, file group  
2942 class, or file other class) to which the process belongs, or if an alternate access control  
2943 mechanism is enabled and it allows the requested access; otherwise, access shall be  
2944 denied.

## 2945 4.8 File Hierarchy

2946 Files in the system are organized in a hierarchical structure in which all of the non-terminal  
2947 nodes are directories and all of the terminal nodes are any other type of file. Since multiple  
2948 directory entries may refer to the same file, the hierarchy is properly described as a “directed  
2949 graph”.

## 2950 4.9 Filenames

2951 Uppercase and lowercase letters shall retain their unique identities between conforming  
2952 implementations.

## 2953 4.10 Filename Portability

2954 For a filename to be portable across implementations conforming to POSIX.1-2024, it shall  
2955 consist only of the portable filename character set as defined in [Section 3.265](#) (on page 70).

2956 **Note:** Applications should avoid using filenames that have the <hyphen-minus> character as the first  
2957 character since this may cause problems when filenames are passed as command line  
2958 arguments.

## 2959 4.11 File System Cache

2960 If the file system is accessed via a memory cache, file-related requirements stated in the rest of  
2961 this standard shall apply to the cache, except where explicitly stated otherwise: this includes  
2962 directory atomicity and serializability requirements (see [Section 4.4](#)), file times update  
2963 requirements (see [Section 4.12](#)), and read-write serializability requirements (see [write\(\)](#)). Cache  
2964 entries shall be transferred to the underlying storage as the result of successful calls to  
2965 [fdatasync\(\)](#), [fsync\(\)](#), or [aio\\_fsync\(\)](#), and may be transferred to storage automatically at other  
2966 times. Such transfers shall be atomic, with minimum units being directory entries (for directory  
2967 contents), aligned data blocks of the fundamental file system block size (for regular-file contents;  
2968 see [<sys/statvfs.h>](#)), and all attributes of a single file (for file attributes).

2969 **Note:** If the system crashes before the cache is fully transferred, later operations' effects may be  
2970 present in storage with earlier effects missing.

2971 **Note:** Operations that create or modify multiple directory entries, aligned data blocks, or file  
2972 attributes (e.g., [mkdir\(\)](#), [rename\(\)](#), [write\(\)](#) with large buffer size, [open\(\)](#) with `O_CREAT`) may  
2973 have only part of their effects transferred to storage, and after a crash these operations may  
2974 appear to have been only partly done, with the parts not necessarily done in any order. For  
2975 example, only the second half of a [write\(\)](#) may be transferred; or `rename("a", "b")` may  
2976 result in `b` being created without `a` being removed.

2977 **Note:** Although conforming file systems are required to perform all caching as described above, some  
2978 file systems may support non-conforming configurations (for example via mount options) for  
2979 which this is not the case. Applications that are used on non-conforming file systems cannot  
2980 rely on files being synchronized properly.

## 2981 4.12 File Times Update

2982 Many operations have requirements to update file timestamps. These requirements do not apply  
2983 to streams that have no underlying file description (for example, memory streams created by  
2984 [open\\_memstream\(\)](#) have no underlying file description).

2985 Each file has three distinct associated timestamps: the time of last data access, the time of last  
2986 data modification, and the time the file status last changed. These values are returned in the file  
2987 characteristics structure `struct stat`, as described in [<sys/stat.h>](#) (on page 414).

2988 Each function or utility in POSIX.1-2024 that reads or writes data (even if the data does not  
2989 change) or performs an operation to change file status (even if the file status does not change)  
2990 indicates which of the appropriate timestamps shall be marked for update. If an implementation  
2991 of such a function or utility marks for update one of these timestamps in a place or time not  
2992 specified by POSIX.1-2024, this shall be documented, except that any changes caused by  
2993 pathname resolution need not be documented. For the other functions or utilities in  
2994 POSIX.1-2024 (those that are not explicitly required to read or write file data or change file  
2995 status, but that in some implementations happen to do so), the effect is unspecified.

2996 An implementation may update timestamps that are marked for update immediately, or it may  
2997 update such timestamps periodically. At the point in time when an update occurs, any marked  
2998 timestamps shall be set to the current time and the update marks shall be cleared. All  
2999 timestamps that are marked for update shall be updated when the file ceases to be open by any  
3000 process or before a *fstat()*, *fstatat()*, *fsync()*, *futimens()*, *lstat()*, *stat()*, *utimensat()*, or *utimes()*  
3001 is successfully performed on the file. Other times at which updates are done are unspecified.  
3002 Marks for update, and updates themselves, shall not be done for files on read-only file systems;  
3003 see [Section 3.295](#) (on page 75).

3004 The resolution of timestamps of files in a file system is implementation-defined, but shall be no  
3005 coarser than one-second resolution. The three timestamps shall always have values that are  
3006 supported by the file system. Whenever any of a file's timestamps are to be set to a value *V*  
3007 according to the rules of the preceding paragraphs of this section, the implementation shall  
3008 immediately set the timestamp to the greatest value supported by the file system that is not  
3009 greater than *V*.

### 3010 4.13 Host and Network Byte Orders

3011 When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned  
3012 values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be  
3013 stored in several octets. The convention is that all such values are stored with 8 bits in each octet,  
3014 and with the first (lowest-addressed) octet holding the most-significant bits. This is called  
3015 "network byte order".

3016 Network byte order may not be convenient for processing actual values. For this, it is more  
3017 sensible for values to be stored as ordinary integers. This is known as "host byte order". In host  
3018 byte order:

- 3019 • The most significant bit might not be stored in the first byte in address order.
- 3020 • Bits might not be allocated to bytes in any obvious order at all.

3021 8-bit values stored in **uint8\_t** objects do not require conversion to or from host byte order, as  
3022 they have the same representation. 16 and 32-bit values can be converted using the *htonl()*,  
3023 *htons()*, *ntohl()*, and *ntohs()* functions. When reading data that is to be converted to host byte  
3024 order, it should either be received directly into a **uint16\_t** or **uint32\_t** object or should be copied  
3025 from an array of bytes using *memcpy()* or similar. Passing the data through other types could  
3026 cause the byte order to be changed. Similar considerations apply when sending data.

### 3027 4.14 Measurement of Execution Time

3028 The mechanism used to measure execution time shall be implementation-defined. The  
3029 implementation shall also define to whom the CPU time that is consumed by interrupt handlers  
3030 and system services on behalf of the operating system will be charged. See [Section 3.90](#) (on page  
3031 44).

## 3032 4.15 Memory Ordering and Synchronization

### 3033 4.15.1 Memory Ordering

#### 3034 4.15.1.1 Data Races

3035 The value of an object visible to a thread *T* at a particular point is the initial value of the object, a  
3036 value stored in the object by *T*, or a value stored in the object by another thread, according to the  
3037 rules below.

3038 Two expression evaluations *conflict* if one of them modifies a memory location and the other one  
3039 reads or modifies the same memory location.

3040 This standard defines a number of atomic operations (see `<stdatomic.h>`) and operations on  
3041 mutexes (see `<threads.h>`) that are specially identified as synchronization operations. These  
3042 operations play a special role in making assignments in one thread visible to another. A  
3043 synchronization operation on one or more memory locations is either an *acquire operation*, a  
3044 *release operation*, both an acquire and release operation, or a *consume operation*. A synchronization  
3045 operation without an associated memory location is a *fence* and can be either an acquire fence, a  
3046 release fence, or both an acquire and release fence. In addition, there are *relaxed atomic operations*,  
3047 which are not synchronization operations, and atomic *read-modify-write operations*, which have  
3048 special characteristics.

3049 **Note:** For example, a call that acquires a mutex will perform an acquire operation on the locations  
3050 composing the mutex. Correspondingly, a call that releases the same mutex will perform a  
3051 release operation on those same locations. Informally, performing a release operation on *A*  
3052 forces prior side effects on other memory locations to become visible to other threads that later  
3053 perform an acquire or consume operation on *A*. Relaxed atomic operations are not included as  
3054 synchronization operations although, like synchronization operations, they cannot contribute to  
3055 data races.

3056 All modifications to a particular atomic object *M* occur in some particular total order, called the  
3057 modification order of *M*. If *A* and *B* are modifications of an atomic object *M*, and *A* happens  
3058 before *B*, then *A* shall precede *B* in the modification order of *M*, which is defined below.

3059 **Note:** This states that the modification orders must respect the “happens before” relation.

3060 **Note:** There is a separate order for each atomic object. There is no requirement that these can be  
3061 combined into a single total order for all objects. In general this will be impossible since  
3062 different threads may observe modifications to different variables in inconsistent orders.

3063 A *release sequence* headed by a release operation *A* on an atomic object *M* is a maximal  
3064 contiguous sub-sequence of side effects in the modification order of *M*, where the first operation  
3065 is *A* and every subsequent operation either is performed by the same thread that performed the  
3066 release or is an atomic read-modify-write operation.

3067 Certain system interfaces *synchronize with* other system interfaces performed by another thread.  
3068 In particular, an atomic operation *A* that performs a release operation on an object *M* shall  
3069 synchronize with an atomic operation *B* that performs an acquire operation on *M* and reads a  
3070 value written by any side effect in the release sequence headed by *A*.

3071 **Note:** Except in the specified cases, reading a later value does not necessarily ensure visibility as  
3072 described below. Such a requirement would sometimes interfere with efficient implementation.

3073 **Note:** The specifications of the synchronization operations define when one reads the value written by  
 3074 another. For atomic variables, the definition is clear. All operations on a given mutex occur in a  
 3075 single total order. Each mutex acquisition “reads the value written” by the last mutex release.

3076 An evaluation *A* carries a dependency to an evaluation *B* if:

- 3077 • the value of *A* is used as an operand of *B*, unless:
  - 3078 — *B* is an invocation of the `kill_dependency()` macro,
  - 3079 — *A* is the left operand of a `&&` or `||` operator,
  - 3080 — *A* is the left operand of a `?:` operator, or
  - 3081 — *A* is the left operand of a `,` (comma) operator; or
- 3082 • *A* writes a scalar object or bit-field *M*, *B* reads from *M* the value written by *A*, and *A* is  
 3083 sequenced before *B*, or
- 3084 • for some evaluation *X*, *A* carries a dependency to *X* and *X* carries a dependency to *B*.

3085 An evaluation *A* is *dependency-ordered before* an evaluation *B* if:

- 3086 • *A* performs a release operation on an atomic object *M*, and, in another thread, *B* performs a  
 3087 consume operation on *M* and reads a value written by any side effect in the release  
 3088 sequence headed by *A*, or
- 3089 • for some evaluation *X*, *A* is dependency-ordered before *X* and *X* carries a dependency to *B*.

3090 An evaluation *A* *inter-thread happens before* an evaluation *B* if *A* synchronizes with *B*, *A* is  
 3091 dependency-ordered before *B*, or, for some evaluation *X*:

- 3092 • *A* synchronizes with *X* and *X* is sequenced before *B*,
- 3093 • *A* is sequenced before *X* and *X* inter-thread happens before *B*, or
- 3094 • *A* inter-thread happens before *X* and *X* inter-thread happens before *B*.

3095 **Note:** The “inter-thread happens before” relation describes arbitrary concatenations of “sequenced  
 3096 before”, “synchronizes with”, and “dependency-ordered before” relationships, with two  
 3097 exceptions. The first exception is that a concatenation is not permitted to end with  
 3098 “dependency-ordered before” followed by “sequenced before”. The reason for this limitation is  
 3099 that a consume operation participating in a “dependency-ordered before” relationship provides  
 3100 ordering only with respect to operations to which this consume operation actually carries a  
 3101 dependency. The reason that this limitation applies only to the end of such a concatenation is  
 3102 that any subsequent release operation will provide the required ordering for a prior consume  
 3103 operation. The second exception is that a concatenation is not permitted to consist entirely of  
 3104 “sequenced before”. The reasons for this limitation are (1) to permit “inter-thread happens  
 3105 before” to be transitively closed and (2) the “happens before” relation, defined below, provides  
 3106 for relationships consisting entirely of “sequenced before”.

3107 An evaluation *A* *happens before* an evaluation *B* if *A* is sequenced before *B* or *A* inter-thread  
 3108 happens before *B*. The implementation shall ensure that a cycle in the “happens before” relation  
 3109 never occurs.

3110 **Note:** This cycle would otherwise be possible only through the use of consume operations.

3111 A *visible side effect* *A* on an object *M* with respect to a value computation *B* of *M* satisfies the  
 3112 conditions:

- 3113 • *A* happens before *B*, and
- 3114 • there is no other side effect *X* to *M* such that *A* happens before *X* and *X* happens before *B*.

3115 The value of a non-atomic scalar object *M*, as determined by evaluation *B*, shall be the value

3116 stored by the visible side effect *A*.

3117 **Note:** If there is ambiguity about which side effect to a non-atomic object is visible, then there is a data  
3118 race and the behavior is undefined.

3119 **Note:** This states that operations on ordinary variables are not visibly reordered. This is not actually  
3120 detectable without data races, but it is necessary to ensure that data races, as defined here, and  
3121 with suitable restrictions on the use of atomics, correspond to data races in a simple interleaved  
3122 (sequentially consistent) execution.

3123 The value of an atomic object *M*, as determined by evaluation *B*, shall be the value stored by  
3124 some side effect *A* that modifies *M*, where *B* does not happen before *A*.

3125 **Note:** The set of side effects from which a given evaluation might take its value is also restricted by  
3126 the rest of the rules described here, and in particular, by the coherence requirements below.

3127 If an operation *A* that modifies an atomic object *M* happens before an operation *B* that modifies  
3128 *M*, then *A* shall be earlier than *B* in the modification order of *M*. (This is known as “write-write  
3129 coherence”.)

3130 If a value computation *A* of an atomic object *M* happens before a value computation *B* of *M*, and  
3131 *A* takes its value from a side effect *X* on *M*, then the value computed by *B* shall either be the  
3132 value stored by *X* or the value stored by a side effect *Y* on *M*, where *Y* follows *X* in the  
3133 modification order of *M*. (This is known as “read-read coherence”.)

3134 If a value computation *A* of an atomic object *M* happens before an operation *B* on *M*, then *A*  
3135 shall take its value from a side effect *X* on *M*, where *X* precedes *B* in the modification order of  
3136 *M*. (This is known as “read-write coherence”.)

3137 If a side effect *X* on an atomic object *M* happens before a value computation *B* of *M*, then the  
3138 evaluation *B* shall take its value from *X* or from a side effect *Y* that follows *X* in the modification  
3139 order of *M*. (This is known as “write-read coherence”.)

3140 **Note:** This effectively disallows implementation reordering of atomic operations to a single object,  
3141 even if both operations are “relaxed” loads. By doing so, it effectively makes the “cache  
3142 coherence” guarantee provided by most hardware available to POSIX atomic operations.

3143 **Note:** The value observed by a load of an atomic object depends on the “happens before” relation,  
3144 which in turn depends on the values observed by loads of atomic objects. The intended reading  
3145 is that there must exist an association of atomic loads with modifications they observe that,  
3146 together with suitably chosen modification orders and the “happens before” relation derived as  
3147 described above, satisfy the resulting constraints as imposed here.

3148 An application contains a data race if it contains two conflicting actions in different threads, at  
3149 least one of which is not atomic, and neither happens before the other. Any such data race  
3150 results in undefined behavior.

#### 3151 4.15.1.2 Memory Order and Consistency

3152 The enumerated type **memory\_order**, defined in `<stdatomic.h>` (if supported), specifies the  
3153 detailed regular (non-atomic) memory synchronization operations as defined in [Section 4.15.1.1](#)  
3154 (on page 100) and may provide for operation ordering. Its enumeration constants specify  
3155 memory order as follows:

3156 For `memory_order_relaxed`, no operation orders memory.

3157 For `memory_order_release`, `memory_order_acq_rel`, and `memory_order_seq_cst`, a  
3158 store operation performs a release operation on the affected memory location.

3159 For `memory_order_acquire`, `memory_order_acq_rel`, and `memory_order_seq_cst`, a  
3160 load operation performs an acquire operation on the affected memory location.

3161 For `memory_order_consume`, a load operation performs a consume operation on the affected  
3162 memory location.

3163 There shall be a single total order  $S$  on all `memory_order_seq_cst` operations, consistent with  
3164 the “happens before” order and modification orders for all affected locations, such that each  
3165 `memory_order_seq_cst` operation  $B$  that loads a value from an atomic object  $M$  observes one  
3166 of the following values:

- 3167 • the result of the last modification  $A$  of  $M$  that precedes  $B$  in  $S$ , if it exists, or
- 3168 • if  $A$  exists, the result of some modification of  $M$  that is not `memory_order_seq_cst` and  
3169 that does not happen before  $A$ , or
- 3170 • if  $A$  does not exist, the result of some modification of  $M$  that is not  
3171 `memory_order_seq_cst`.

3172 **Note:** Although it is not explicitly required that  $S$  include lock operations, it can always be extended  
3173 to an order that does include lock and unlock operations, since the ordering between those is  
3174 already included in the “happens before” ordering.

3175 **Note:** Atomic operations specifying `memory_order_relaxed` are relaxed only with respect to  
3176 memory ordering. Implementations must still guarantee that any given atomic access to a  
3177 particular atomic object be indivisible with respect to all other atomic accesses to that object.

3178 For an atomic operation  $B$  that reads the value of an atomic object  $M$ , if there is a  
3179 `memory_order_seq_cst` fence  $X$  sequenced before  $B$ , then  $B$  observes either the last  
3180 `memory_order_seq_cst` modification of  $M$  preceding  $X$  in the total order  $S$  or a later  
3181 modification of  $M$  in its modification order.

3182 For atomic operations  $A$  and  $B$  on an atomic object  $M$ , where  $A$  modifies  $M$  and  $B$  takes its value,  
3183 if there is a `memory_order_seq_cst` fence  $X$  such that  $A$  is sequenced before  $X$  and  $B$  follows  
3184  $X$  in  $S$ , then  $B$  observes either the effects of  $A$  or a later modification of  $M$  in its modification  
3185 order.

3186 For atomic modifications  $A$  and  $B$  of an atomic object  $M$ ,  $B$  occurs later than  $A$  in the  
3187 modification order of  $M$  if:

- 3188 • there is a `memory_order_seq_cst` fence  $X$  such that  $A$  is sequenced before  $X$ , and  $X$   
3189 precedes  $B$  in  $S$ , or
- 3190 • there is a `memory_order_seq_cst` fence  $Y$  such that  $Y$  is sequenced before  $B$ , and  $A$   
3191 precedes  $Y$  in  $S$ , or
- 3192 • there are `memory_order_seq_cst` fences  $X$  and  $Y$  such that  $A$  is sequenced before  $X$ ,  $Y$  is  
3193 sequenced before  $B$ , and  $X$  precedes  $Y$  in  $S$ .

3194 Atomic read-modify-write operations shall always read the last value (in the modification order)  
3195 stored before the write associated with the read-modify-write operation.

3196 An atomic store shall only store a value that has been computed from constants and input values  
3197 by a finite sequence of evaluations, such that each evaluation observes the values of variables as  
3198 computed by the last prior assignment in the sequence. The ordering of evaluations in this  
3199 sequence shall be such that:

- 3200 • If an evaluation  $B$  observes a value computed by  $A$  in a different thread, then  $B$  does not  
3201 happen before  $A$ .
- 3202 • If an evaluation  $A$  is included in the sequence, then all evaluations that assign to the same  
3203 variable and happen before  $A$  are also included.

3204 **Note:** The second requirement disallows “out-of-thin-air”, or “speculative” stores of atomics when  
 3205 relaxed atomics are used. Since unordered operations are involved, evaluations can appear in  
 3206 this sequence out of thread order.

## 3207 4.15.2 Memory Synchronization

3208 In order to avoid data races, applications shall ensure that non-lock-free access to any memory  
 3209 location by more than one thread of control (threads or processes) is restricted such that no  
 3210 thread of control can read or modify a memory location while another thread of control might be  
 3211 modifying it. Such access can be restricted using functions that synchronize thread execution  
 3212 and also synchronize memory with respect to other threads. The following functions shall  
 3213 synchronize memory with respect to other threads on all successful calls:

3214	<i>cond_broadcast()</i>	<i>pthread_rwlock_clockrdlock()</i>	<i>sem_timedwait()</i>
3215	<i>cond_signal()</i>	<i>pthread_rwlock_clockwrlock()</i>	<i>sem_trywait()</i>
3216	<i>fork()</i>	<i>pthread_rwlock_rdlock()</i>	<i>sem_wait()</i>
3217	<i>pthread_barrier_wait()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>semctl()</i>
3218	<i>pthread_cond_broadcast()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>semop()</i>
3219	<i>pthread_cond_signal()</i>	<i>pthread_rwlock_tryrdlock()</i>	<i>thr_create()</i>
3220	<i>pthread_create()</i>	<i>pthread_rwlock_trywrlock()</i>	<i>thr_join()</i>
3221	<i>pthread_join()</i>	<i>pthread_rwlock_unlock()</i>	<i>wait()</i>
3222	<i>pthread_spin_lock()</i>	<i>pthread_rwlock_wrlock()</i>	<i>waitid()</i>
3223	<i>pthread_spin_trylock()</i>	<i>sem_clockwait()</i>	<i>waitpid()</i>
3224	<i>pthread_spin_unlock()</i>	<i>sem_post()</i>	

3225 The *pthread\_once()* and *call\_once()* functions shall synchronize memory for the first successful  
 3226 call in each thread for a given **pthread\_once\_t** or **once\_flag** object, respectively. If the *init\_routine*  
 3227 called by *pthread\_once()* or *call\_once()* is a cancellation point and is canceled, a successful call to  
 3228 *pthread\_once()* for the same **pthread\_once\_t** object or to *call\_once()* for the same **once\_flag** object,  
 3229 made from a cancellation cleanup handler shall also synchronize memory.

3230 RPP|TPP The *pthread\_mutex\_clocklock()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_setprioceiling()*,  
 3231 *pthread\_mutex\_timedlock()*, and *pthread\_mutex\_trylock()* functions shall synchronize memory on  
 3232 all calls that acquire the mutex, including those that return [EOWNERDEAD]. The  
 3233 *pthread\_mutex\_unlock()* function shall synchronize memory on all calls that release the mutex.

3234 **Note:** If the mutex type is PTHREAD\_MUTEX\_RECURSIVE, calls to the locking functions do not  
 3235 acquire the mutex if the calling thread already owns it, and calls to *pthread\_mutex\_unlock()* do  
 3236 not release the mutex if it has a lock count greater than one.

3237 The *pthread\_cond\_clockwait()*, *pthread\_cond\_wait()*, and *pthread\_cond\_timedwait()* functions shall  
 3238 synchronize memory on all calls that release and re-acquire the specified mutex, including calls  
 3239 that return [EOWNERDEAD], both when the mutex is released and when it is re-acquired.

3240 **Note:** If the mutex type is PTHREAD\_MUTEX\_RECURSIVE, calls to *pthread\_cond\_clockwait()*,  
 3241 *pthread\_cond\_wait()*, and *pthread\_cond\_timedwait()* do not release and re-acquire the mutex if it  
 3242 has a lock count greater than one.

3243 The *mtx\_lock()*, *mtx\_timedlock()*, and *mtx\_trylock()* functions shall synchronize memory on all  
 3244 calls that acquire the mutex. The *mtx\_unlock()* function shall synchronize memory on all calls  
 3245 that release the mutex.

3246 **Note:** If the mutex is a recursive mutex, calls to the locking functions do not acquire the mutex if the  
 3247 calling thread already owns it, and calls to *mtx\_unlock()* do not release the mutex if it has a lock  
 3248 count greater than one.

3249 The *cond\_wait()* and *cond\_timedwait()* functions shall synchronize memory on all calls that release



3250 and re-acquire the specified mutex, both when the mutex is released and when it is re-acquired.

3251 **Note:** If the mutex is a recursive mutex, calls to `cond_wait()` and `cond_timedwait()` do not release and re-  
3252 acquire the mutex if it has a lock count greater than one.

3253 Unless explicitly stated otherwise, if one of the functions named in this section returns an error,  
3254 it is unspecified whether the invocation causes memory to be synchronized.

3255 Applications can allow more than one thread of control to read a memory location  
3256 simultaneously.

3257 For purposes of determining the existence of a data race, all lock and unlock operations on a  
3258 particular synchronization object that synchronize memory shall behave as atomic operations,  
3259 and they shall occur in some particular total order (see [Section 4.15.1](#), on page 100).

## 3260 4.16 Pathname Resolution

3261 Pathname resolution is performed for a process to resolve a pathname to a particular directory  
3262 entry for a file in the file hierarchy. There may be multiple pathnames that resolve to the same  
3263 directory entry, and multiple directory entries for the same file. When a process resolves a  
3264 pathname of an existing directory entry, the entire pathname shall be resolved as described  
3265 below. When a process resolves a pathname of a directory entry that is to be created immediately  
3266 after the pathname is resolved, pathname resolution terminates when all components of the path  
3267 prefix of the last component have been resolved. It is then the responsibility of the process to  
3268 create the final component.

3269 Each filename in the pathname is located in the directory specified by its predecessor (for  
3270 example, in the pathname fragment **a/b**, file **b** is located in the directory specified by **a**).  
3271 Pathname resolution shall fail if this cannot be accomplished. If the pathname begins with a  
3272 `<slash>`, the predecessor of the first filename in the pathname shall be taken to be the root  
3273 directory of the process (such pathnames are referred to as “absolute pathnames”). If the  
3274 pathname does not begin with a `<slash>`, the predecessor of the first filename of the pathname  
3275 shall be taken to be either the current working directory of the process or for certain interfaces  
3276 the directory identified by a file descriptor passed to the interface (such pathnames are referred  
3277 to as “relative pathnames”).

3278 The interpretation of a pathname component is dependent on the value of `{NAME_MAX}` and  
3279 `_POSIX_NO_TRUNC` associated with the path prefix of that component. If any pathname  
3280 component is longer than `{NAME_MAX}`, the implementation shall consider this an error.

3281 A pathname that contains at least one non-`<slash>` character and that ends with one or more  
3282 trailing `<slash>` characters shall not be resolved successfully unless the last pathname  
3283 component before the trailing `<slash>` characters resolves (with symbolic links followed—see  
3284 below) to an existing directory or a directory entry that is to be created for a directory  
3285 immediately after the pathname is resolved. Interfaces using pathname resolution may specify  
3286 additional constraints<sup>7</sup> when a pathname that does not name an existing directory contains at  
3287 least one non-`<slash>` character and contains one or more trailing `<slash>` characters.

3288 If a symbolic link is encountered during pathname resolution, the behavior shall depend on  
3289 whether the pathname component is at the end of the pathname and on the function being  
3290 performed. If all of the following are true, then pathname resolution is complete:

---

3291 7. The only interfaces that further constrain pathnames in POSIX.1-2024 are the `rename()` and `renameat()` functions (see XSH [rename\(\)](#))  
3292 and the `mv` utility (see XCU [mv](#)).

- 3293 1. This is the last pathname component of the pathname.
- 3294 2. The pathname has no trailing `<slash>`.
- 3295 3. The function is required to act on the symbolic link itself, or certain arguments direct that
- 3296 the function act on the symbolic link itself.

3297 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the  
3298 symbolic link, except that if the contents of the symbolic link is the empty string, then either  
3299 pathname resolution shall fail with functions reporting an [ENOENT] error and utilities writing  
3300 an equivalent diagnostic message, or the pathname of the directory containing the symbolic link  
3301 shall be used in place of the contents of the symbolic link. If the contents of the symbolic link  
3302 consist solely of `<slash>` characters, then all leading `<slash>` characters of the remaining  
3303 pathname shall be omitted from the resulting combined pathname, leaving only the leading  
3304 `<slash>` characters from the symbolic link contents. In the cases where prefixing occurs, if the  
3305 combined length exceeds {PATH\_MAX}, and the implementation considers this to be an error,  
3306 pathname resolution shall fail with functions reporting an [ENAMETOOLONG] error and  
3307 utilities writing an equivalent diagnostic message. Otherwise, the resolved pathname shall be  
3308 the resolution of the pathname just created. If the resulting pathname does not begin with a  
3309 `<slash>`, the predecessor of the first filename of the pathname is taken to be the directory  
3310 containing the symbolic link.

3311 If the system detects a loop in the pathname resolution process, pathname resolution shall fail  
3312 with functions reporting an [ELOOP] error and utilities writing an equivalent diagnostic  
3313 message. The same may happen if during the resolution process more symbolic links were  
3314 followed than the implementation allows. This implementation-defined limit shall not be  
3315 smaller than {SYMLOOP\_MAX}.

3316 The special filename dot shall refer to the directory specified by its predecessor. The special  
3317 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case,  
3318 in the root directory, dot-dot may refer to the root directory itself.

3319 A pathname consisting of a single `<slash>` shall resolve to the root directory of the process. A  
3320 null pathname shall not be successfully resolved. If a pathname begins with two successive  
3321 `<slash>` characters, the first component following the leading `<slash>` characters may be  
3322 interpreted in an implementation-defined manner, although more than two leading `<slash>`  
3323 characters shall be treated as a single `<slash>` character.

3324 Pathname resolution for a given pathname shall yield the same results when used by any  
3325 interface in POSIX.1-2024 as long as there are no changes to any files evaluated during pathname  
3326 resolution for the given pathname between resolutions.

## 3327 4.17 Process ID Reuse

3328 A process group ID shall not be reused by the system until the process group lifetime ends.

3329 A process ID shall not be reused by the system until the process lifetime ends. In addition, if  
3330 there exists a process group whose process group ID is equal to that process ID, the process ID  
3331 shall not be reused by the system until the process group lifetime ends. A process that is not a  
3332 system process shall not have a process ID of 1.

## 3333 4.18 Scheduling Policy

3334 A scheduling policy affects process or thread ordering:

- 3335 • When a process or thread is a running thread and it becomes a blocked thread
- 3336 • When a process or thread is a running thread and it becomes a preempted thread
- 3337 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3338 • When a running thread calls a function that can change the priority or scheduling policy of
- 3339 a process or thread
- 3340 • In other scheduling policy-defined circumstances

3341 Conforming implementations shall define the manner in which each of the scheduling policies  
 3342 may modify the priorities or otherwise affect the ordering of processes or threads at each of the  
 3343 occurrences listed above. Additionally, conforming implementations shall define in what other  
 3344 circumstances and in what manner each scheduling policy may modify the priorities or affect  
 3345 the ordering of processes or threads.

## 3346 4.19 Seconds Since the Epoch

3347 A value that approximates the number of seconds that have elapsed since the Epoch. A  
 3348 Coordinated Universal Time name (specified in terms of seconds (*tm\_sec*), minutes (*tm\_min*),  
 3349 hours (*tm\_hour*), days since January 1 of the year (*tm\_yday*), and calendar year minus 1900  
 3350 (*tm\_year*)) is related to a time represented as seconds since the Epoch, according to the  
 3351 expression below.

3352 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and  
 3353 the value is non-negative, the value is related to a Coordinated Universal Time name according  
 3354 to the C-language expression, where *tm\_sec*, *tm\_min*, *tm\_hour*, *tm\_yday*, and *tm\_year* are all  
 3355 integer types:

$$\begin{aligned}
 &tm\_sec + tm\_min*60 + tm\_hour*3600 + tm\_yday*86400 + \\
 &\quad (tm\_year-70)*31536000 + ((tm\_year-69)/4)*86400 - \\
 &\quad ((tm\_year-1)/100)*86400 + ((tm\_year+299)/400)*86400
 \end{aligned}$$

3359 The relationship between the actual date and time in Coordinated Universal Time, as  
 3360 determined by the International Earth Rotation Service, and the system's current value for  
 3361 seconds since the Epoch is unspecified.

3362 How any changes to the value of seconds since the Epoch are made to align to a desired  
 3363 relationship with the current actual time is implementation-defined. As represented in seconds  
 3364 since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

3365 **Note:** The last three terms of the expression add in a day for each year that follows a leap year starting  
 3366 with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973,  
 3367 the second subtracts a day back out every 100 years starting in 2001, and the third adds a day  
 3368 back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that  
 3369 is, the remainder is discarded leaving only the integer quotient.

## 3370 4.20 Semaphore

3371 A minimum synchronization primitive to serve as a basis for more complex synchronization  
3372 mechanisms to be defined by the application program.

3373 For the mandatory semaphores (those not associated with the X/Open System Interfaces (XSI)  
3374 option), a semaphore is represented as a shareable resource that has a non-negative integer  
3375 value. When the value is zero, there is a (possibly empty) set of threads awaiting the availability  
3376 of the semaphore.

3377 For the semaphores associated with the X/Open System Interfaces (XSI) option, a semaphore is  
3378 an integer with minimum value 0 and an implementation-defined maximum value which shall  
3379 be at least 32767. The *semget()* function can be called to create a set or array of semaphores. A  
3380 semaphore set can contain one or more semaphores up to an implementation-defined value.

### 3381 Semaphore Lock Operation

3382 An operation that is applied to a semaphore. If, prior to the operation, the value of the  
3383 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and  
3384 added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

### 3385 Semaphore Unlock Operation

3386 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in  
3387 the set of threads awaiting the semaphore, then some thread from that set shall be removed from  
3388 the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

## 3389 4.21 Special Device Drivers

3390 Some devices require control operations, other than the operations that are common to most  
3391 devices (such as *read()*, *write()*, *open()*, and *close()*), but because the device belongs to a class that  
3392 is not present in the majority of systems, standardization of a device-specific application  
3393 program interface (API) for controlling it has not been practical. The driver for such a device  
3394 may respond to the *write()* function to transfer data to the device or the *read()* function to collect  
3395 information from the device. The interpretation of the information is defined by the  
3396 implementor of the driver.

3397 The term *special device* refers to hardware, or an object that appears to the application as such;  
3398 access to the driver for this hardware uses the file abstraction *character special file*.  
3399 Implementations supporting the Device Control option shall provide the means to integrate a  
3400 device driver into the system. The means available to integrate drivers into the system and the  
3401 way character special files that refer to them are created are implementation defined. Character  
3402 special files that have no structure defined by this standard can be accessed using the  
3403 *posix\_devctl()* function defined in the System Interfaces volume of POSIX.1-2024.

## 3404 4.22 Thread-Safety

3405 Refer to XSH [Section 2.9](#) (on page 537).

## 3406 4.23 Treatment of Error Conditions for Mathematical Functions

3407 For all the functions in the `<math.h>` header, an application wishing to check for error situations  
3408 should set `errno` to 0 and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On  
3409 return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |`  
3410 `FE_UNDERFLOW)` is non-zero, an error has occurred.

3411 On implementations that support the IEC 60559 Floating-Point option, whether or when  
3412 functions in the `<math.h>` header raise an undeserved underflow floating-point exception is  
3413 unspecified. Otherwise, as implied by XSH `feraiseexcept()`, the `<math.h>` functions do not raise  
3414 spurious floating-point exceptions (detectable by the user), other than the inexact floating-point  
3415 exception.

3416 The error conditions defined for all functions in the `<math.h>` header are domain, pole and  
3417 range errors, described below. If a domain, pole, or range error occurs and the integer expression  
3418 `(math_errhandling & MATH_ERRNO)` is zero, then `errno` shall either be set to the value  
3419 corresponding to the error, as specified below, or be left unmodified. If no such error occurs,  
3420 `errno` shall be left unmodified regardless of the setting of `math_errhandling`.

### 3421 4.23.1 Domain Error

3422 A “domain error” shall occur if an input argument is outside the domain over which the  
3423 mathematical function is defined. The description of each function lists any required domain  
3424 errors; an implementation may define additional domain errors, provided that such errors are  
3425 consistent with the mathematical definition of the function.

3426 On a domain error, the function shall return an implementation-defined value; if the integer  
3427 expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to [EDOM]; if  
3428 the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “invalid”  
3429 floating-point exception shall be raised.

### 3430 4.23.2 Pole Error

3431 A “pole error” shall occur if the mathematical result of the function has an exact infinite result as  
3432 the finite input argument(s) are approached in the limit (for example, `log(0.0)`). The  
3433 description of each function lists any required pole errors; an implementation may define  
3434 additional pole errors, provided that such errors are consistent with the mathematical definition  
3435 of the function.

3436 On a pole error, the function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or  
3437 `HUGE_VALL` according to the return type, with the same sign as the correct value of the  
3438 function; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall  
3439 be set to [ERANGE]; if the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-  
3440 zero, the “divide-by-zero” floating-point exception shall be raised.

3441 **4.23.3 Range Error**

3442 A “range error” shall occur if the finite mathematical result of the function cannot be  
 3443 represented in an object of the specified type, due to extreme magnitude. The description of  
 3444 each function lists any required range errors; an implementation may define additional range  
 3445 errors, provided that such errors are consistent with the mathematical definition of the function  
 3446 and are the result of either overflow or underflow.

3447 **4.23.3.1 Result Overflows**

3448 A floating result overflows if the magnitude of the mathematical result is finite but so large that  
 3449 the mathematical result cannot be represented without extraordinary roundoff error in an object  
 3450 of the specified type. If a floating result overflows and default rounding is in effect, then the  
 3451 function shall return the value of the macro HUGE\_VAL, HUGE\_VALF, or HUGE\_VALL  
 3452 according to the return type, with the same sign as the correct value of the function; if the integer  
 3453 expression (math\_errhandling & MATH\_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if  
 3454 the integer expression (math\_errhandling & MATH\_ERREXCEPT) is non-zero, the “overflow”  
 3455 floating-point exception shall be raised.

3456 **4.23.3.2 Result Underflows**

3457 The result underflows if the magnitude of the mathematical result is so small that the  
 3458 mathematical result cannot be represented, without extraordinary roundoff error, in an object of  
 3459 the specified type. If the result underflows, the function shall return an implementation-defined  
 3460 value whose magnitude is no greater than the smallest normalized positive number in the  
 3461 specified type; if the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero,  
 3462 whether *errno* is set to [ERANGE] is implementation-defined; if the integer expression  
 3463 (math\_errhandling & MATH\_ERREXCEPT) is non-zero, whether the “underflow” floating-point  
 3464 exception is raised is implementation-defined.

3465 **4.24 Treatment of NaN Arguments for the Mathematical Functions**

3466 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,  
 3467 except where stated otherwise.

3468 If a function with one or more NaN arguments returns a NaN result, the result should be the  
 3469 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3470 On implementations that support the IEC 60559:1989 standard floating point, functions with  
 3471 signaling NaN argument(s) shall be treated as if the function were called with an argument that  
 3472 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3473 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are  
 3474 evaluated during the function call.

3475 On implementations that support the IEC 60559:1989 standard floating point, for those  
 3476 functions that do not have a documented domain error, the following shall apply:

3477 These functions shall fail if:

3478 Domain Error Any argument is a signaling NaN.

3479 Either, the integer expression (math\_errhandling & MATH\_ERRNO) is non-zero and *errno*  
 3480 shall be set to [EDOM], or the integer expression (math\_errhandling &  
 3481 MATH\_ERREXCEPT) is non-zero and the invalid floating-point exception shall be raised.

## 3482 4.25 Utility

3483 A utility program shall be either an executable file, such as might be produced by a compiler or  
3484 linker system from computer source code, or a file of shell source code, directly interpreted by  
3485 the shell. The program may have been produced by the user, provided by the system  
3486 implementor, or acquired from an independent distributor.

3487 The system may implement certain utilities as shell functions (see XCU [Section 2.9.5](#), on page  
3488 2511) or built-in utilities, but only an application that is aware of the command search order (as  
3489 described in XCU [Section 2.9.1.4](#), on page 2502) or of performance characteristics can discern  
3490 differences between the behavior of such a function or built-in utility and that of an executable  
3491 file.

## 3492 4.26 Variable Assignment

3493 In the shell command language, a word consisting of the following parts:

3494 *varname=value*

3495 When used in a context where assignment is defined to occur and at no other time, the *value*  
3496 (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.  
3497 Assignment context occurs in the *cmd\_prefix* portion of a shell simple command, as well as in  
3498 arguments of a recognized declaration utility.

3499 **Note:** For further information, see XCU [Section 2.9.1](#) (on page 2500).

3500 The *varname* and *value* parts shall meet the requirements for a name and a word, respectively,  
3501 except that they are delimited by the embedded unquoted <equals-sign>, in addition to other  
3502 delimiters.

3503 **Note:** Additional delimiters are described in XCU [Section 2.3](#) (on page 2475).

3504 When a variable assignment is done, the variable shall be created if it did not already exist. If  
3505 *value* is not specified, the variable shall be given a null value.

3506 **Note:** An alternative form of variable assignment:

3507 *symbol=value*

3508 (where *symbol* is a valid word delimited by an <equals-sign>, but not a valid name) produces  
3509 unspecified results. The form *symbol=value* is used by the KornShell *name[expression]=value*  
3510 syntax.





## File Format Notation

The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility descriptions use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces volume of POSIX.1-2024 `printf()` function, as described in this chapter. When used in STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the System Interfaces volume of POSIX.1-2024 `scanf()` function to read the input file.

The description of an individual record is as follows:

```
"<format>", [<arg1>, <arg2>, ..., <argn>]
```

The *format* is a character string that contains three types of objects defined below:

1. *Characters* that are not “escape sequences” or “conversion specifications”, as described below, shall be copied to the output.
2. *Escape Sequences* represent non-graphic characters and the escape character (<backslash>).
3. *Conversion Specifications* specify the output format of each argument; see below.

The following characters have the following special meaning in the format string:

' ' (An empty character position.) Represents one or more <blank> characters from the portable character set.

Δ Represents exactly one <space> character.

Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

3532

**Table 5-1** Escape Sequences and Associated Actions

3533

3534

3535

3536

3537

3538

3539

3540

3541

3542

3543

3544

3545

3546

3547

3548

Escape Sequence	Represents Character	Terminal Action
\\	<backslash>	Print the <backslash> character.
\a	<alert>	Attempt to alert the user through audible or visible notification.
\b	<backspace>	Move the printing position to one column before the current position, unless the current position is the start of a line.
\f	<form-feed>	Move the printing position to the initial printing position of the next logical page.
\n	<newline>	Move the printing position to the start of the next line.
\r	<carriage-return>	Move the printing position to the start of the current line.
\t	<tab>	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
\v	<vertical-tab>	Move the printing position to the start of the next <vertical-tab> position. If there are no more <vertical-tab> positions left on the page, the behavior is undefined.

3549 Each conversion specification is introduced by the <percent-sign> character ('%'). After the  
3550 character '%', the following shall appear in sequence:

3551 *flags* Zero or more *flags*, in any order, that modify the meaning of the conversion  
3552 specification.

3553 *field width* An optional string of decimal digits to specify a minimum field width. For an  
3554 output field, if the converted value has fewer bytes than the field width, it shall be  
3555 padded on the left (or right, if the left-adjustment flag ('-'), described below, has  
3556 been given) to the field width.

3557 *precision* Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversion  
3558 specifiers (the field is padded with leading zeros), the number of digits to appear  
3559 after the radix character for the *e* and *f* conversion specifiers, the maximum  
3560 number of significant digits for the *g* conversion specifier; or the maximum  
3561 number of bytes to be written from a string in the *s* conversion specifier. The  
3562 precision shall take the form of a <period> ('.') followed by a decimal digit  
3563 string; a null digit string is treated as zero.

3564 *conversion specifier characters*  
3565 A conversion specifier character (see below) that indicates the type of conversion  
3566 to be applied.

3567 The *flag* characters and their meanings are:

3568 - The result of the conversion shall be left-justified within the field.

3569 + The result of a signed conversion shall always begin with a sign ('+' or '-').

3570 <space> If the first character of a signed conversion is not a sign, a <space> shall be  
3571 prefixed to the result. This means that if the <space> and '+' flags both appear,  
3572 the <space> flag shall be ignored.

3573 # The value shall be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s*  
3574 conversion specifiers, the behavior is undefined. For the *o* conversion specifier, it  
3575 shall increase the precision to force the first digit of the result to be a zero. For *x* or  
3576 *X* conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively.  
3577 For *a*, *A*, *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers, the result shall always contain a  
3578 radix character, even if no digits follow the radix character. For *g* and *G* conversion  
3579 specifiers, trailing zeros shall not be removed from the result as they usually are.

3580 0 For *a*, *A*, *d*, *e*, *E*, *f*, *F*, *g*, *G*, *i*, *o*, *u*, *x*, and *X* conversion specifiers, leading zeros  
3581 (following any indication of sign or base) shall be used to pad to the field width  
3582 rather than performing space padding, except when converting an infinity or NaN.  
3583 If the '0' and '-' flags both appear, the '0' flag shall be ignored. For *d*, *i*, *o*, *u*,  
3584 *x*, and *X* conversion specifiers, if a precision is specified, the '0' flag shall be  
3585 ignored. For other conversion specifiers, the behavior is undefined.

3586 Each conversion specifier character shall result in fetching zero or more arguments. The results  
3587 are undefined if there are insufficient arguments for the format. If the format is exhausted while  
3588 arguments remain, the excess arguments shall be ignored.

3589 The conversion specifiers and their meanings are:

3590 *a,A* The floating-point number argument representing a floating-point number shall be  
3591 converted in the style "[−]0xh.hhhhp±d", where there is one hexadecimal digit  
3592 (which shall be non-zero if the argument is a normalized floating-point number  
3593 and is otherwise unspecified) before the decimal-point character and the number  
3594 of hexadecimal digits after it is equal to the precision; if the precision is missing

3595		and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact
3596		representation of the value; if the precision is missing and FLT_RADIX is not a
3597		power of 2, then the precision shall be sufficient to distinguish different floating-
3598		point values in the internal representation used by the utility, except that trailing
3599		zeros may be omitted; if the precision is zero and the # flag is not specified, no
3600		decimal-point character shall appear. The letters "abcdef" shall be used for a
3601		conversion and the letters "ABCDEF" for A conversion. The A conversion specifier
3602		produces a number with X and P instead of x and p. The exponent shall always
3603		contain at least one digit, and only as many more digits as necessary to represent
3604		the decimal exponent of 2. If the value is zero, the exponent shall be zero. A
3605		floating-point number argument representing an infinity or NaN shall be
3606		converted in the style of an f or F conversion specifier.
3607	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i), unsigned octal
3608		(o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and
3609		i specifiers shall convert to signed decimal in the style "[−]ddd". The x
3610		conversion specifier shall use the numbers and letters "0123456789abcdef" and
3611		the X conversion specifier shall use the numbers and letters
3612		"0123456789ABCDEF". The <i>precision</i> component of the argument shall specify
3613		the minimum number of digits to appear. If the value being converted can be
3614		represented in fewer digits than the specified minimum, it shall be expanded with
3615		leading zeros. The default precision shall be 1. The result of converting a zero
3616		value with a precision of 0 shall be no characters. If both the field width and
3617		precision are omitted, the implementation may precede, follow, or precede and
3618		follow numeric arguments of types d, i, and u with <blank> characters from the
3619		portable character set; arguments of type o (octal) may be preceded with leading
3620		zeros.
3621	f,F	The floating-point number argument shall be written in decimal notation in the
3622		style [−]ddd.ddd, where the number of digits after the radix character (shown here
3623		as a decimal point) shall be equal to the <i>precision</i> specification. The LC_NUMERIC
3624		locale category shall determine the radix character to use in this format. If the
3625		<i>precision</i> is omitted from the argument, six digits shall be written after the radix
3626		character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3627		A floating-point number argument representing an infinity shall be converted in
3628		one of the styles "[−]inf" or "[−]infinity"; which style is implementation-
3629		defined. A floating-point number argument representing a NaN shall be converted
3630		in one of the styles "[−]nan( <i>n-char-sequence</i> )" or "[−]nan"; which style,
3631		and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The F
3632		conversion specifier produces "INF", "INFINITY", or "NaN" instead of "inf",
3633		"infinity", or "nan", respectively.
3634	e,E	The floating-point number argument shall be written in the style [−]d.ddde±dd (the
3635		symbol '±' indicates either a <plus-sign> or <hyphen-minus>), where there is one
3636		digit before the radix character (shown here as a decimal point) and the number of
3637		digits after it is equal to the precision. The LC_NUMERIC locale category shall
3638		determine the radix character to use in this format. When the precision is missing,
3639		six digits shall be written after the radix character; if the precision is 0, no radix
3640		character shall appear. The E conversion specifier shall produce a number with E
3641		instead of e introducing the exponent. The exponent shall always contain at least
3642		two digits. However, if the value to be written requires an exponent greater than
3643		two digits, additional exponent digits shall be written as necessary.
3644		A floating-point number argument representing an infinity or NaN shall be

3645		converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
3646	<code>g,G</code>	The floating-point number argument shall be written in style <code>f</code> or <code>e</code> (or in style <code>F</code> or <code>E</code> in the case of a <code>G</code> conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style <code>e</code> (or <code>E</code> ) shall be used only if the exponent resulting from the conversion is less than $-4$ or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.
3648		
3649		
3650		
3651		
3652		A floating-point number argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
3653		
3654	<code>c</code>	The single-byte character argument shall be written.
3655	<code>s</code>	The argument shall be taken to be a string and bytes from the string shall be written until the end of the string or the number of bytes indicated by the <i>precision</i> specification of the argument is reached. If the precision is omitted from the argument, it shall be taken to be infinite, so all bytes up to the end of the string shall be written.
3656		
3657		
3658		
3659		
3660	<code>%</code>	Write a <code>'%'</code> character; no argument shall be converted. Applications using the <i>printf</i> utility shall ensure that the complete conversion specification is <code>%%</code> .
3661		
3662		In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. The term “field width” should not be confused with the term “precision” used in the description of <code>%s</code> .
3663		
3664		
3665		

### Examples

3666		
3667		To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where <i>weekday</i> and <i>month</i> are strings:
3668		
3669		<code>"%s, Δ%sΔ%d, Δ%d: %.2d\n" &lt;weekday&gt;, &lt;month&gt;, &lt;day&gt;, &lt;hour&gt;, &lt;min&gt;</code>
3670		To show <code>'π'</code> written to 5 decimal places:
3671		<code>"piΔ=Δ%.5f\n", &lt;value of π&gt;</code>
3672		To show an input file format consisting of five <code>&lt;colon&gt;</code> -separated fields:
3673		<code>"%s:%s:%s:%s:%s\n", &lt;arg1&gt;, &lt;arg2&gt;, &lt;arg3&gt;, &lt;arg4&gt;, &lt;arg5&gt;</code>

# Character Set

## 6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of POSIX.1-2024. The first eight entries in Table 6-1 and all characters in Table 6-2 (on page 122) are defined in the ISO/IEC 6429:1992 standard. The rest of the characters in Table 6-1 are defined in the ISO/IEC 10646:2020 standard.

3683

**Table 6-1** Portable Character Set

3684

3685

3686

3687

3688

3689

3690

3691

3692

3693

3694

3695

3696

3697

3698

3699

3700

3701

3702

3703

3704

3705

3706

3707

3708

3709

3710

3711

3712

3713

3714

Symbolic Name(s)	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>, <BEL>		<U0007>	BELL
<backspace>, <BS>		<U0008>	BACKSPACE
<tab>, <HT>		<U0009>	CHARACTER TABULATION
<newline>, <LF>		<U000A>	LINE FEED (LF)
<vertical-tab>, <VT>		<U000B>	LINE TABULATION
<form-feed>, <FF>		<U000C>	FORM FEED (FF)
<carriage-return>, <CR>		<U000D>	CARRIAGE RETURN (CR)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(	<U0028>	LEFT PARENTHESIS
<right-parenthesis>	)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>, <hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>, <period>	.	<U002E>	FULL STOP
<slash>, <solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE
<four>	4	<U0034>	DIGIT FOUR
<five>	5	<U0035>	DIGIT FIVE

	Symbolic Name(s)	Glyph	UCS	Description
3715	<six>	6	<U0036>	DIGIT SIX
3716	<seven>	7	<U0037>	DIGIT SEVEN
3717	<eight>	8	<U0038>	DIGIT EIGHT
3718	<nine>	9	<U0039>	DIGIT NINE
3719	<colon>	:	<U003A>	COLON
3720	<semicolon>	;	<U003B>	SEMICOLON
3721	<less-than-sign>	<	<U003C>	LESS-THAN SIGN
3722	<equals-sign>	=	<U003D>	EQUALS SIGN
3723	<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
3724	<question-mark>	?	<U003F>	QUESTION MARK
3725	<commercial-at>	@	<U0040>	COMMERCIAL AT
3726	<A>	A	<U0041>	LATIN CAPITAL LETTER A
3727	<B>	B	<U0042>	LATIN CAPITAL LETTER B
3728	<C>	C	<U0043>	LATIN CAPITAL LETTER C
3729	<D>	D	<U0044>	LATIN CAPITAL LETTER D
3730	<E>	E	<U0045>	LATIN CAPITAL LETTER E
3731	<F>	F	<U0046>	LATIN CAPITAL LETTER F
3732	<G>	G	<U0047>	LATIN CAPITAL LETTER G
3733	<H>	H	<U0048>	LATIN CAPITAL LETTER H
3734	<I>	I	<U0049>	LATIN CAPITAL LETTER I
3735	<J>	J	<U004A>	LATIN CAPITAL LETTER J
3736	<K>	K	<U004B>	LATIN CAPITAL LETTER K
3737	<L>	L	<U004C>	LATIN CAPITAL LETTER L
3738	<M>	M	<U004D>	LATIN CAPITAL LETTER M
3739	<N>	N	<U004E>	LATIN CAPITAL LETTER N
3740	<O>	O	<U004F>	LATIN CAPITAL LETTER O
3741	<P>	P	<U0050>	LATIN CAPITAL LETTER P
3742	<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
3743	<R>	R	<U0052>	LATIN CAPITAL LETTER R
3744	<S>	S	<U0053>	LATIN CAPITAL LETTER S
3745	<T>	T	<U0054>	LATIN CAPITAL LETTER T
3746	<U>	U	<U0055>	LATIN CAPITAL LETTER U
3747	<V>	V	<U0056>	LATIN CAPITAL LETTER V
3748	<W>	W	<U0057>	LATIN CAPITAL LETTER W
3749	<X>	X	<U0058>	LATIN CAPITAL LETTER X
3750	<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
3751	<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
3752	<left-square-bracket>	[	<U005B>	LEFT SQUARE BRACKET
3753	<backslash>, <reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
3754	<right-square-bracket>	]	<U005D>	RIGHT SQUARE BRACKET
3755	<circumflex-accent>, <circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
3756	<low-line>, <underscore>	_	<U005F>	LOW LINE
3757	<grave-accent>	`	<U0060>	GRAVE ACCENT
3758	<a>	a	<U0061>	LATIN SMALL LETTER A
3759	<b>	b	<U0062>	LATIN SMALL LETTER B
3760	<c>	c	<U0063>	LATIN SMALL LETTER C
3761	<d>	d	<U0064>	LATIN SMALL LETTER D
3762	<e>	e	<U0065>	LATIN SMALL LETTER E
3763	<f>	f	<U0066>	LATIN SMALL LETTER F
3764	<g>	g	<U0067>	LATIN SMALL LETTER G
3765	<h>	h	<U0068>	LATIN SMALL LETTER H

	Symbolic Name(s)	Glyph	UCS	Description
3767				
3768	<i>	i	<U0069>	LATIN SMALL LETTER I
3769	<j>	j	<U006A>	LATIN SMALL LETTER J
3770	<k>	k	<U006B>	LATIN SMALL LETTER K
3771	<l>	l	<U006C>	LATIN SMALL LETTER L
3772	<m>	m	<U006D>	LATIN SMALL LETTER M
3773	<n>	n	<U006E>	LATIN SMALL LETTER N
3774	<o>	o	<U006F>	LATIN SMALL LETTER O
3775	<p>	p	<U0070>	LATIN SMALL LETTER P
3776	<q>	q	<U0071>	LATIN SMALL LETTER Q
3777	<r>	r	<U0072>	LATIN SMALL LETTER R
3778	<s>	s	<U0073>	LATIN SMALL LETTER S
3779	<t>	t	<U0074>	LATIN SMALL LETTER T
3780	<u>	u	<U0075>	LATIN SMALL LETTER U
3781	<v>	v	<U0076>	LATIN SMALL LETTER V
3782	<w>	w	<U0077>	LATIN SMALL LETTER W
3783	<x>	x	<U0078>	LATIN SMALL LETTER X
3784	<y>	y	<U0079>	LATIN SMALL LETTER Y
3785	<z>	z	<U007A>	LATIN SMALL LETTER Z
3786	<left-brace>, <left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
3787	<vertical-line>		<U007C>	VERTICAL LINE
3788	<right-brace>, <right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
3789	<tilde>	~	<U007E>	TILDE

3790 POSIX.1-2024 uses character names other than the above, but only in an informative way; for  
 3791 example, in examples to illustrate the use of characters beyond the portable character set with  
 3792 the facilities of POSIX.1-2024.

3793 [Table 6-1](#) (on page 117) defines the characters in the portable character set and the corresponding  
 3794 symbolic character names used to identify each character in a character set description file.  
 3795 Characters defined in [Table 6-2](#) (on page 122) may also be used in character set description files.

3796 POSIX.1-2024 places only the following requirements on the encoded values of the characters in  
 3797 the portable character set:

- 3798 • If the encoded values associated with each member of the portable character set are not  
 3799 invariant across all locales supported by the implementation, if an application uses any  
 3800 pair of locales where the character encodings differ, or accesses data from an application  
 3801 using a locale which has different encodings from the locales used by the application, the  
 3802 results are unspecified.
- 3803 • The encoded values associated with the digits 0 to 9 shall be such that the value of each  
 3804 character after 0 shall be one greater than the value of the previous character.
- 3805 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.
- 3806 • The encoded values associated with <period>, <slash>, <newline>, and <carriage-return>  
 3807 shall be invariant across all locales supported by the implementation.
- 3808 • The encoded values associated with the members of the portable character set are each  
 3809 represented in a single byte. Moreover, if the value is stored in an object of C-language  
 3810 type **char**, it is guaranteed to be positive (except the NUL, which is always zero).

3811 Conforming implementations shall support certain character and character set attributes, as  
 3812 defined in [Section 7.2](#) (on page 128).

## 3813 6.2 Character Encoding

3814 The POSIX locale shall contain 256 single-byte characters including the characters in [Table 6-1](#)  
3815 (on page 117) and [Table 6-2](#) (on page 122), which have the properties listed in [Section 7.3.1](#) (on  
3816 page 131). It is unspecified whether characters not listed in those two tables are classified as  
3817 **punct** or **cntrl**, or neither. Other locales shall contain the characters in [Table 6-1](#) (on page 117)  
3818 and may contain any or all of the control characters identified in [Table 6-2](#) (on page 122); the  
3819 presence, meaning, and representation of any additional characters are locale-specific.

3820 In locales other than the POSIX locale, a character may have a state-dependent encoding. There  
3821 are two types of these encodings:

- 3822 • A single-shift encoding (where each character not in the initial shift state is preceded by a  
3823 shift code) can be defined if each shift-code and character sequence is considered a multi-  
3824 byte character. This is done using the concatenated-constant format in a character set  
3825 description file, as described in [Section 6.4](#) (on page 121). If the implementation supports a  
3826 character encoding of this type, all of the standard utilities in the Shell and Utilities volume  
3827 of POSIX.1-2024 shall support it. Use of a single-shift encoding with any of the functions in  
3828 the System Interfaces volume of POSIX.1-2024 that do not specifically mention the effects  
3829 of state-dependent encoding is implementation-defined.
- 3830 • A locking-shift encoding (where the state of the character is determined by a shift code  
3831 that may affect more than the single character following it) cannot be defined with the  
3832 current character set description file format. Use of a locking-shift encoding with any of  
3833 the standard utilities in the Shell and Utilities volume of POSIX.1-2024 or with any of the  
3834 functions in the System Interfaces volume of POSIX.1-2024 that do not specifically mention  
3835 the effects of state-dependent encoding is implementation-defined.

3836 While in the initial shift state, all characters in the portable character set shall retain their usual  
3837 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the  
3838 sequence shall be a function of the current shift state. A byte with all bits zero shall be  
3839 interpreted as the null character independent of shift state. Such a byte shall not occur as part of  
3840 any other character. Likewise, the byte values used to encode <period>, <slash>, <newline>, and  
3841 <carriage-return> shall not occur as part of any other character in any locale.

3842 The maximum allowable number of bytes in a character in the current locale shall be indicated  
3843 by {MB\_CUR\_MAX}, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a  
3844 character set description file; see [Section 6.4](#) (on page 121). The implementation's maximum  
3845 number of bytes in a character shall be defined by the C-language macro {MB\_LEN\_MAX}.

## 3846 6.3 C Language Wide-Character Codes

3847 In the shell, the standard utilities are written so that the encodings of characters are described by  
3848 the locale's `LC_CTYPE` definition (see [Section 7.3.1](#), on page 131) and there is no differentiation  
3849 between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C  
3850 language, a differentiation is made. To ease the handling of variable length characters, the C  
3851 language has introduced the concept of wide-character codes.

3852 All wide-character codes in a given process consist of an equal number of bits. This is in contrast  
3853 to characters, which can consist of a variable number of bytes. The byte or byte sequence that  
3854 represents a character can also be represented as a wide-character code. Wide-character codes  
3855 thus provide a uniform size for manipulating text data. A wide-character code having all bits  
3856 zero is the null wide-character code (see [Section 3.233](#), on page 65), and terminates wide-  
3857 character strings (see [Section 3.417](#), on page 93). The wide-character value for each member of  
3858 the portable character set shall equal its value when used as the lone character in an integer



3859 character constant. Wide-character codes for other characters are locale and implementation-  
3860 defined. State shift bytes shall not have a wide-character code representation. POSIX.1-2024  
3861 provides no means of defining a wide-character codeset.

3862 Arguments to the functions declared in the `<wchar.h>` header can point to arrays containing  
3863 `wchar_t` values that do not correspond to valid wide character codes according to the `LC_CTYPE`  
3864 category of the locale being used. Such values shall be processed according to the specified  
3865 semantics for the function in the System Interfaces volume of POSIX.1-2024, except that it is  
3866 unspecified whether an encoding error occurs if such a value appears in the format string of a  
3867 function that has a format string as a parameter and the specified semantics do not require that  
3868 value to be processed as if by `wcrtomb()`.

## 3869 6.4 Character Set Description File

3870 Implementations shall provide a character set description file for at least one coded character set  
3871 supported by the implementation. These files are referred to elsewhere in POSIX.1-2024 as  
3872 *charmap* files. It is implementation-defined whether or not users or applications can provide  
3873 additional character set description files.

3874 POSIX.1-2024 does not require that multiple character sets or codesets be supported. Although  
3875 multiple charmap files are supported, it is the responsibility of the implementation to provide  
3876 the file or files; if only one is provided, only that one is accessible using the *localedef* utility's `-f`  
3877 option.

3878 Each character set description file, except those that use the ISO/IEC 10646:2020 standard  
3879 position values as the encoding values, shall define characteristics for the coded character set  
3880 and the encoding for the characters specified in [Table 6-1](#) (on page 117), and may define  
3881 encoding for additional characters supported by the implementation. Other information about  
3882 the coded character set may also be in the file. Coded character set character values shall be  
3883 defined using symbolic character names followed by character encoding values.

3884 Each symbolic name specified in [Table 6-1](#) (on page 117) shall be included in the file. Each  
3885 character in [Table 6-1](#) (on page 117) (each row in the table) shall be mapped to a unique coding  
3886 value. For each character in [Table 6-2](#) (on page 122) that exists in the character set described by  
3887 the file, the character's symbolic name(s) from [Table 6-2](#) (on page 122) and the character's single-  
3888 byte encoding value shall be included in the file.

3889

Table 6-2 Non-Portable Control Characters

	Symbolic Name(s)	UCS	Description
3890	<SOH>	<U0001>	START OF HEADING
3891	<STX>	<U0002>	START OF TEXT
3892	<ETX>	<U0003>	END OF TEXT
3893	<EOT>	<U0004>	END OF TRANSMISSION
3894	<ENQ>	<U0005>	ENQUIRY
3895	<ACK>	<U0006>	ACKNOWLEDGE
3896	<SO>	<U000E>	SHIFT OUT
3897	<SI>	<U000F>	SHIFT IN
3898	<DLE>	<U0010>	DATA LINK ESCAPE
3899	<DC1>	<U0011>	DEVICE CONTROL ONE
3900	<DC2>	<U0012>	DEVICE CONTROL TWO
3901	<DC3>	<U0013>	DEVICE CONTROL THREE
3902	<DC4>	<U0014>	DEVICE CONTROL FOUR
3903	<NAK>	<U0015>	NEGATIVE ACKNOWLEDGE
3904	<SYN>	<U0016>	SYNCHRONOUS IDLE
3905	<ETB>	<U0017>	END OF TRANSMISSION BLOCK
3906	<CAN>	<U0018>	CANCEL
3907	<EM>	<U0019>	END OF MEDIUM
3908	<SUB>	<U001A>	SUBSTITUTE
3909	<ESC>	<U001B>	ESCAPE
3910	<IS4>, <FS>	<U001C>	INFORMATION SEPARATOR FOUR
3911	<IS3>, <GS>	<U001D>	INFORMATION SEPARATOR THREE
3912	<IS2>, <RS>	<U001E>	INFORMATION SEPARATOR TWO
3913	<IS1>, <US>	<U001F>	INFORMATION SEPARATOR ONE
3914	<DEL>	<U007F>	DELETE
3915			

3916 The following declarations can precede the character definitions. Each shall consist of the  
 3917 symbol shown in the following list, starting in column 1, including the surrounding brackets,  
 3918 followed by one or more <blank> characters, followed by the value to be assigned to the symbol.

3919 **<code\_set\_name>** The name of the coded character set for which the character set  
 3920 description file is defined. The characters of the name shall be taken from  
 3921 the set of characters with visible glyphs defined in Table 6-1 (on page 117).

3922 **<mb\_cur\_max>** The maximum number of bytes in a multi-byte character. This shall  
 3923 default to 1.

3924 **<mb\_cur\_min>** An unsigned positive integer value that defines the minimum number of  
 3925 XSI bytes in a character for the encoded character set. On XSI-conformant  
 3926 systems, <mb\_cur\_min> shall always be 1.

3927 **<escape\_char>** The character used to indicate that the characters following shall be  
 3928 interpreted in a special way, as defined later in this section. This shall  
 3929 default to <backslash> ('\\'), which is the character used in all the  
 3930 following text and examples, unless otherwise noted.

3931 **<comment\_char>** The character that, when placed in column 1 of a charmap line, is used to  
 3932 indicate that the line shall be ignored. The default character shall be the  
 3933 <number-sign> ('#').

3934 The character set mapping definitions shall be all the lines immediately following an identifier  
 3935 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line

3936 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a  
 3937 **<comment\_char>** in the first column shall be ignored. Each non-comment line of the character  
 3938 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)  
 3939 shall be in either of two forms:

3940 "%s %s %s\n", <symbolic-name>, <encoding>, <comments>

3941 or:

3942 "%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,  
 3943 <encoding>, <comments>

3944 In the first format, the line in the character set mapping definition shall define a single symbolic  
 3945 name and a corresponding encoding. A symbolic name is one or more characters from the set  
 3946 shown with visible glyphs in [Table 6-1](#) (on page 117), enclosed between angle brackets. A  
 3947 character following an escape character is interpreted as itself; for example, the sequence  
 3948 "<\\>" represents the symbolic name "\" enclosed between angle brackets.

3949 In the second format, the line in the character set mapping definition shall define a range of one  
 3950 or more symbolic names. In this form, the symbolic names shall consist of zero or more non-  
 3951 numeric characters from the set shown with visible glyphs in [Table 6-1](#) (on page 117), followed  
 3952 by an integer formed by one or more decimal digits. Both integers shall contain the same  
 3953 number of digits. The characters preceding the integer shall be identical in the two symbolic  
 3954 names, and the integer formed by the digits in the second symbolic name shall be equal to or  
 3955 greater than the integer formed by the digits in the first name. This shall be interpreted as a  
 3956 series of symbolic names formed from the common part and each of the integers between the  
 3957 first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the  
 3958 symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3959 A character set mapping definition line shall exist for all symbolic names specified in [Table 6-1](#)  
 3960 (on page 117), and shall define the coded character value that corresponds to the character  
 3961 indicated in the table, or the coded character value that corresponds to the control character  
 3962 symbolic name. If the control characters commonly associated with the symbolic names in [Table](#)  
 3963 [6-2](#) (on page 122) are supported by the implementation, the symbolic name and the  
 3964 corresponding encoding value shall be included in the file. Additional unique symbolic names  
 3965 may be included. A coded character value can be represented by more than one symbolic name.

3966 The encoding part is expressed as one (for single-byte character values) or more concatenated  
 3967 decimal, octal, or hexadecimal constants in the following formats:

3968 "%cd%u", <escape\_char>, <decimal byte value>

3969 "%cx%x", <escape\_char>, <hexadecimal byte value>

3970 "%c%o", <escape\_char>, <octal byte value>

3971 Decimal constants shall be represented by two or three decimal digits, preceded by the escape  
 3972 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".  
 3973 Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape  
 3974 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal  
 3975 constants shall be represented by two or three octal digits, preceded by the escape character; for  
 3976 example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an  
 3977 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the  
 3978 same type, and interpreted in sequence from first to last with the first byte of the multi-  
 3979 byte character specified by the first byte in the sequence. The manner in which these constants  
 3980 are represented in the character stored in the system is implementation-defined. (This notation  
 3981 was chosen for reasons of portability. There is no requirement that the internal representation in  
 3982 the computer memory be in this same order.) Omitting bytes from a multi-byte character  
 3983 definition produces undefined results.

3984 In lines defining ranges of symbolic names, the encoded value shall be the value for the first  
 3985 symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic  
 3986 names defined by the range shall have encoding values in increasing order. Bytes shall be  
 3987 treated as unsigned octets, and carry shall be propagated between the bytes as necessary to  
 3988 represent the range. However, because this causes a null byte in the second or subsequent bytes  
 3989 of a character, such a declaration should not be specified. For example, the line:

```
3990 <j0101>...<j0104> \d129\d254
```

3991 is interpreted as:

```
3992 <j0101>          \d129\d254
3993 <j0102>          \d129\d255
3994 <j0103>          \d130\d00
3995 <j0104>          \d130\d01
```

3996 The expanded declaration of the symbol <j0103> in the above example is an invalid  
 3997 specification, because it contains a null byte in the second byte of a character.

3998 The comment is optional.

3999 POSIX.1-2024 provides no means of defining a wide-character codeset.

4000 The following declarations can follow the character set mapping definitions (after the "END  
 4001 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in  
 4002 column 1, followed by the value(s) to be associated to the keyword, as defined below.

4003 **WIDTH** A non-negative integer value defining the column width (see [Section 3.75](#), on page  
 4004 42) for the printable characters in the coded character set specified in [Table 6-1](#) (on  
 4005 page 117) and [Table 6-2](#) (on page 122). Coded character set character values shall  
 4006 be defined using symbolic character names followed by column width values.  
 4007 Defining a character with more than one **WIDTH** produces undefined results. The  
 4008 **END WIDTH** keyword shall be used to terminate the **WIDTH** definitions.  
 4009 Specifying the width of a non-printable character in a **WIDTH** declaration  
 4010 produces undefined results.

4011 **WIDTH\_DEFAULT**

4012 A non-negative integer value defining the default column width for any printable  
 4013 character not listed by one of the **WIDTH** keywords. If no **WIDTH\_DEFAULT**  
 4014 keyword is included in the charmap, the default character width shall be 1.

### 4015 Example

4016 After the "END CHARMAP" statement, a syntax for a width definition would be:

```
4017 WIDTH
4018 <A> 1
4019 <B> 1
4020 <C>...<Z> 1
4021 ...
4022 <fool>...<foon> 2
4023 ...
4024 END WIDTH
```

4025 In this example, the numerical code point values represented by the symbols <A> and <B> are  
 4026 assigned a width of 1. The code point values <C> to <Z> inclusive (<C>, <D>, <E>, and so on)  
 4027 are also assigned a width of 1. Using <A>...<Z> would have required fewer lines, but the  
 4028 alternative was shown to demonstrate flexibility. The keyword **WIDTH\_DEFAULT** could have

4029           been added as appropriate.

#### 4030 **6.4.1 State-Dependent Character Encodings**

4031           This section addresses the use of state-dependent character encodings (that is, those in which the  
4032           encoding of a character is dependent on one or more shift codes that may precede it).

4033           A single-shift encoding (where each character not in the initial shift state is preceded by a shift  
4034           code) can be defined in the charmap format if each shift-code/character sequence is considered  
4035           a multi-byte character, defined using the concatenated-constant format described in [Section 6.4](#)  
4036           (on page 121). If the implementation supports a character encoding of this type, all of the  
4037           standard utilities shall support it. A locking-shift encoding (where the state of the character is  
4038           determined by a shift code that may affect more than the single character following it) could be  
4039           defined with an extension to the charmap format described in [Section 6.4](#) (on page 121).

4040           If the implementation supports a character encoding of this type, any of the standard utilities  
4041           that describe character (*versus* byte) or text-file manipulation shall have the following  
4042           characteristics:

- 4043           1. The utility shall process the statefully encoded data as a concatenation of state-  
4044           independent characters. The presence of redundant locking shifts shall not affect the  
4045           comparison of two statefully encoded strings.
- 4046           2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall  
4047           produce output that contains locking shifts at the beginning or end of the resulting data,  
4048           if appropriate, to retain correct state information.



# Locale

## 4051 7.1 General

4052 A locale is the definition of the subset of a user's environment that depends on language and  
 4053 cultural conventions. It is made up from one or more categories. Each category is identified by  
 4054 its name and controls specific aspects of the behavior of components of the system. Category  
 4055 names correspond to the following environment variable names:

4056 *LC\_CTYPE* Character classification and case conversion.

4057 *LC\_COLLATE* Collation order.

4058 *LC\_MONETARY* Monetary formatting.

4059 *LC\_NUMERIC* Numeric, non-monetary formatting.

4060 *LC\_TIME* Date and time formats.

4061 *LC\_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

4062 The standard utilities in the Shell and Utilities volume of POSIX.1-2024 shall base their behavior  
 4063 on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility.  
 4064 The behavior of some of the C-language functions defined in the System Interfaces volume of  
 4065 POSIX.1-2024 shall also be modified based on a locale selection. The locale to be used by these  
 4066 functions can be selected in the following ways:

- 4067 1. For functions such as *isalnum\_l()* that take a locale object as an argument, a locale object  
 4068 can be obtained from *newlocale()* or *duplocale()* and passed to the function.
- 4069 2. For functions that do not take a locale object as an argument, the current locale for the  
 4070 thread can be set by calling *uselocale()* or the global locale for the process can be set by  
 4071 calling *setlocale()*. Such functions shall use the current locale of the calling thread if one  
 4072 has been set for that thread; otherwise, they shall use the global locale.
- 4073 3. Some functions, such as *catopen()* and those related to text domains, may reference  
 4074 various environment variables and a locale category of a specific locale to access files they  
 4075 need to use.

4076 Locales other than those supplied by the implementation can be created via the *localedef* utility,  
 4077 provided that the *\_POSIX2\_LOCALEDEF* symbol is defined on the system. Even if *localedef* is  
 4078 not provided, all implementations conforming to the System Interfaces volume of POSIX.1-2024  
 4079 shall provide one or more locales that behave as described in this chapter. The input to the  
 4080 utility is described in Section 7.3 (on page 128). The value that is used to specify a locale when  
 4081 using environment variables shall be the string specified as the *name* operand to the *localedef*  
 4082 utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for  
 4083 the POSIX locale (see Section 7.2, on page 128). When the value of a locale environment variable  
 4084 begins with a <slash> (' / '), it shall be interpreted as the pathname of the locale definition; the  
 4085 type of file (regular, directory, and so on) used to store the locale definition is implementation-  
 4086 defined. If the value does not begin with a <slash>, the mechanism used to locate the locale is  
 4087 implementation-defined.

4088 If incompatible character sets are used by the locale categories, the results achieved by an  
 4089 application utilizing these categories are undefined. Two locale categories have incompatible  
 4090 character sets if one of the categories is *LC\_CTYPE* and the locale data associated with the other  
 4091 category includes at least one character that either is not in the character set used by *LC\_CTYPE*  
 4092 or has a different encoding than the same character in the character set used by *LC\_CTYPE*.

4093 Likewise, unless specified otherwise, if different codesets are used by a particular category of the  
 4094 selected locale and by the data being processed by an interface whose behavior is dependent on  
 4095 that category of the selected locale, the results are undefined.

4096 Applications can select the desired locale by calling the *newlocale()* or *setlocale()* function with  
 4097 the appropriate value. If the function is invoked with an empty string, such as:

```
4098 newlocale(LC_ALL_MASK, "", (locale_t)0);
```

4099 or:

```
4100 setlocale(LC_ALL, "");
```

4101 the value of the corresponding environment variable is used. If the environment variable is  
 4102 unset or is set to the empty string, the implementation shall set the appropriate environment as  
 4103 defined in [Chapter 8](#) (on page 167).

## 4104 7.2 POSIX Locale

4105 Conforming systems shall provide a POSIX locale, also known as the C locale. In POSIX.1 the  
 4106 requirements for the POSIX locale are more extensive than the requirements for the C locale as  
 4107 specified in the ISO C standard. However, in a conforming POSIX implementation, the POSIX  
 4108 locale and the C locale are identical. The behavior of standard utilities and functions in the  
 4109 POSIX locale shall be as if the locale was defined via the *localedef* utility with input data from the  
 4110 POSIX locale tables in [Section 7.3](#).

4111 For C-language programs, the POSIX locale shall be the default locale when the *setlocale()*  
 4112 function is not called.

4113 The POSIX locale can be specified by assigning to the appropriate environment variables the  
 4114 values "C" or "POSIX".

4115 All implementations shall define a locale as the default locale, to be invoked when no  
 4116 environment variables are set, or set to the empty string. This default locale can be the POSIX  
 4117 locale or any other implementation-defined locale. Some implementations may provide facilities  
 4118 for local installation administrators to set the default locale, customizing it for each location.  
 4119 POSIX.1-2024 does not require such a facility.

## 4120 7.3 Locale Definition

4121 The capability to specify additional locales to those provided by an implementation is optional,  
 4122 denoted by the *\_POSIX2\_LOCALEDEF* symbol. If the option is not supported, only  
 4123 implementation-supplied locales are available. Such locales shall be documented using the  
 4124 format specified in this section.

4125 Locales can be described with the file format presented in this section. The file format is that  
 4126 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the  
 4127 "locale definition file", but no locales shall be affected by this file unless it is processed by  
 4128 *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility  
 4129 shall apply to *localedef* or to any other similar utility used to install locale information using the



4130 locale definition file format described here.

4131 The locale definition file shall contain one or more locale category source definitions, and shall  
4132 not contain more than one definition for the same locale category. If the file contains source  
4133 definitions for more than one category, implementation-defined categories, if present, shall  
4134 appear after the categories defined by [Section 7.1](#) (on page 127). A category source definition  
4135 contains either the definition of a category or a **copy** directive. For a description of the **copy**  
4136 directive, see *localedef*. In the event that some of the information for a locale category, as  
4137 specified in this volume of POSIX.1-2024, is missing from the locale source definition, the  
4138 behavior of that category, if it is referenced, is unspecified.

4139 A category source definition shall consist of a category header, a category body, and a category  
4140 trailer. A category header shall consist of the character string naming of the category, beginning  
4141 with the characters *LC\_*. The category trailer shall consist of the string "END", followed by one  
4142 or more <blank> characters and the string used in the corresponding category header.

4143 The category body shall consist of one or more lines of text. Each line shall contain an identifier,  
4144 optionally followed by one or more operands. Identifiers shall be either keywords, identifying a  
4145 particular locale element, or collating elements. In addition to the keywords defined in this  
4146 volume of POSIX.1-2024, the source can contain implementation-defined keywords. Each  
4147 keyword within a locale shall have a unique name (that is, two categories cannot have a  
4148 commonly-named keyword); no keyword shall start with the characters *LC\_*. Identifiers shall be  
4149 separated from the operands by one or more <blank> characters.

4150 Operands shall be characters, collating elements, or strings of characters. Strings shall be  
4151 enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the <escape  
4152 character>, described below. When a keyword is followed by more than one operand, the  
4153 operands shall be separated by <semicolon> characters; <blank> characters shall be allowed  
4154 both before and after a <semicolon>.

4155 The first category header in the file can be preceded by a line modifying the comment character.  
4156 It shall have the following format, starting in column 1:

```
4157 "comment_char %c\n", <comment character>
```

4158 The comment character shall default to the <number-sign> ('#'). Blank lines and lines  
4159 containing the <comment character> in the first position shall be ignored.

4160 The first category header in the file can be preceded by a line modifying the escape character to  
4161 be used in the file. It shall have the following format, starting in column 1:

```
4162 "escape_char %c\n", <escape character>
```

4163 The escape character shall default to <backslash>, which is the character used in all examples  
4164 shown in this volume of POSIX.1-2024.

4165 A line can be continued by placing an escape character as the last character on the line; this  
4166 continuation character shall be discarded from the input. Although the implementation need not  
4167 accept any one portion of a continued line with a length exceeding {LINE\_MAX} bytes, it shall  
4168 place no limits on the accumulated length of the continued line. Comment lines shall not be  
4169 continued on a subsequent line using an escaped <newline>.

4170 Individual characters, characters in strings, and collating elements shall be represented using  
4171 symbolic names, as defined below. In addition, characters can be represented using the  
4172 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic  
4173 notation is used, the resultant locale definitions are in many cases not portable between systems.  
4174 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when  
4175 used to represent itself it shall be preceded by the escape character. The following rules apply to  
4176 character representation:

4177 1. A character can be represented via a symbolic name, enclosed within angle brackets '<' and '>'. The symbolic name, including the angle brackets, shall exactly match a symbolic name defined in the charmap file specified via the *localedef* -f option, and it shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the charmap file shall constitute an error, unless the category is *LC\_CTYPE* or *LC\_COLLATE*, in which case it shall constitute a warning condition (see *localedef* for a description of actions resulting from errors and warnings). The specification of a symbolic name in a **collating-element** or **collating-symbol** section that duplicates a symbolic name in the charmap file (if present) shall be an error. Use of the escape character or a right angle bracket within a symbolic name is invalid unless the character is preceded by the escape character.

4188 For example:

```
4189 <c>;<c-cedilla> "<M><a><y>"
```

4190 2. A character in the portable character set can be represented by the character itself, in which case the value of the character is implementation-defined. (Implementations may allow other characters to be represented as themselves, but such locale definitions are not portable.) Within a string, the double-quote character, the escape character, and the right angle bracket character shall be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters:

```
4196 , ; < > escape_char
```

4197 shall be escaped to be interpreted as the character itself.

4198 For example:

```
4199 c "May"
```

4200 3. A character can be represented as an octal constant. An octal constant shall be specified as the escape character followed by two or three octal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

4204 For example:

```
4205 \143;\347;\143\150 "\115\141\171"
```

4206 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall be specified as the escape character followed by an 'x' followed by two hexadecimal digits. Each constant shall represent a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

4211 For example:

```
4212 \x63;\xe7;\x63\x68 "\x4d\x61\x79"
```

4213 5. A character can be represented as a decimal constant. A decimal constant shall be specified as the escape character followed by a 'd' followed by two or three decimal digits. Each constant represents a byte value. Multi-byte values can be represented by concatenated constants specified in byte order with the last constant specifying the least significant byte of the character.

4218 For example:

```
4219 \d99;\d231;\d99\d104 "\d77\d97\d121"
```

4220 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the

4221 escape character. Only characters existing in the character set for which the locale definition is  
 4222 created shall be specified, whether using symbolic names, the characters themselves, or octal,  
 4223 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the  
 4224 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not  
 4225 present in the charmap file can be specified and shall be ignored, as specified under item 1  
 4226 above.

### 4227 7.3.1 LC\_CTYPE

4228 The *LC\_CTYPE* category shall define character classification, case conversion, and other  
 4229 character attributes. In addition, a series of characters can be represented by three adjacent  
 4230 <period> characters representing an ellipsis symbol (" . . . "). The ellipsis specification shall be  
 4231 interpreted as meaning that all values between the values preceding and following it represent  
 4232 valid characters. The ellipsis specification shall be valid only within a single encoded character  
 4233 set; that is, within a group of characters of the same size. An ellipsis shall be interpreted as  
 4234 including in the list all characters with an encoded value higher than the encoded value of the  
 4235 character preceding the ellipsis and lower than the encoded value of the character following the  
 4236 ellipsis.

4237 For example:

```
4238 \x30; . . . ; \x39;
```

4239 includes in the character class all characters with encoded values between the endpoints.

4240 The following keywords shall be recognized. In the descriptions, the term “automatically  
 4241 included” means that it shall not be an error either to include or omit any of the referenced  
 4242 characters; the implementation provides them if missing (even if the entire keyword is missing)  
 4243 and accepts them silently if present. When the implementation automatically includes a missing  
 4244 character, it shall have an encoded value dependent on the charmap file in effect (see the  
 4245 description of the *localedef* *-f* option); otherwise, it shall have a value derived from an  
 4246 implementation-defined character mapping.

4247 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included  
 4248 characters. It is not possible to define a locale without these automatically included characters  
 4249 unless some implementation extension is used to prevent their inclusion. Such a definition  
 4250 would not be a proper superset of the C or POSIX locale and, thus, it might not be possible for  
 4251 conforming applications to work properly.

4252 **copy** Specify the name of an existing locale which shall be used as the definition of  
 4253 this category. If this keyword is specified, no other keyword shall be specified.

4254 **upper** Define characters to be classified as uppercase letters.

4255 In the POSIX locale, only:

```
4256 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

4257 shall be included:

4258 In a locale definition file, no character specified for the keywords **ctrl**, **digit**,  
 4259 **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as  
 4260 defined in [Section 6.4](#) (on page 121) (the portable character set), are  
 4261 automatically included in this class.

4262 **lower** Define characters to be classified as lowercase letters.

4263 In the POSIX locale, only:

4264		a b c d e f g h i j k l m n o p q r s t u v w x y z
4265		shall be included.
4266		In a locale definition file, no character specified for the keywords <b>cntrl</b> , <b>digit</b> ,
4267		<b>punct</b> , or <b>space</b> shall be specified. The lowercase letters <a> to <z> of the
4268		portable character set are automatically included in this class.
4269	<b>alpha</b>	Define characters to be classified as letters.
4270		In the POSIX locale, only characters in the classes <b>upper</b> and <b>lower</b> shall be
4271		included.
4272		In a locale definition file, no character specified for the keywords <b>cntrl</b> , <b>digit</b> ,
4273		<b>punct</b> , or <b>space</b> shall be specified. Characters classified as either <b>upper</b> or
4274		<b>lower</b> are automatically included in this class.
4275	<b>digit</b>	Define the characters to be classified as numeric digits.
4276		In all locales, only:
4277		0 1 2 3 4 5 6 7 8 9
4278		shall be included.
4279		In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4280		<four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in
4281		contiguous ascending sequence by numerical value. The digits <zero> to
4282		<nine> of the portable character set are automatically included in this class.
4283	<b>alnum</b>	Define characters to be classified as letters and numeric digits. Only the
4284		characters specified for the <b>alpha</b> and <b>digit</b> keywords shall be specified.
4285		Characters specified for the keywords <b>alpha</b> and <b>digit</b> are automatically
4286		included in this class.
4287	<b>space</b>	Define characters to be classified as white-space characters.
4288		In the POSIX locale, exactly <space>, <form-feed>, <newline>, <carriage-
4289		return>, <tab>, and <vertical-tab> shall be included.
4290		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4291		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>graph</b> , or <b>xdigit</b> shall be specified. The <space>, <form-
4292		feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
4293		character set, and any characters included in the class <b>blank</b> are automatically
4294		included in this class.
4295	<b>cntrl</b>	Define characters to be classified as control characters.
4296		In the POSIX locale, no characters in classes <b>alpha</b> or <b>print</b> shall be included.
4297		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4298		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>punct</b> , <b>graph</b> , <b>print</b> , or <b>xdigit</b> shall be specified.
4299	<b>punct</b>	Define characters to be classified as punctuation characters.
4300		In the POSIX locale, neither the <space> nor any characters in classes <b>alpha</b> ,
4301		<b>digit</b> , or <b>cntrl</b> shall be included.
4302		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4303		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>cntrl</b> , <b>xdigit</b> , or as the <space> shall be specified.

4304	<b>graph</b>	Define characters to be classified as printable characters, not including the
4305		<space>.
4306		In the POSIX locale, all characters in classes <b>alpha</b> , <b>digit</b> , and <b>punct</b> shall be
4307		included; no characters in class <b>cntrl</b> shall be included.
4308		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> ,
4309		<b>alpha</b> , <b>digit</b> , <b>xdigit</b> , and <b>punct</b> are automatically included in this class. No
4310		character specified for the keyword <b>cntrl</b> shall be specified.
4311	<b>print</b>	Define characters to be classified as printable characters, including the
4312		<space>.
4313		In the POSIX locale, all characters in class <b>graph</b> shall be included; no
4314		characters in class <b>cntrl</b> shall be included.
4315		In a locale definition file, characters specified for the keywords <b>upper</b> , <b>lower</b> ,
4316		<b>alpha</b> , <b>digit</b> , <b>xdigit</b> , <b>punct</b> , <b>graph</b> , and the <space> are automatically included
4317		in this class. No character specified for the keyword <b>cntrl</b> shall be specified.
4318	<b>xdigit</b>	Define the characters to be classified as hexadecimal digits.
4319		In all locales, only:
4320		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
4321		shall be included.
4322		In a locale definition file, only the characters defined for the class <b>digit</b> shall be
4323		specified, in contiguous ascending sequence by numerical value, followed by
4324		two sets, in either order, of six characters representing the hexadecimal digits
4325		corresponding to the decimal numbers 10 to 15 inclusive, with each set in
4326		ascending order: <A>, <B>, <C>, <D>, <E>, <F> and <a>, <b>, <c>, <d>, <e>
4327		, <f>. The digits <zero> to <nine>, the uppercase letters <A> to <F>, and the
4328		lowercase letters <a> to <f> of the portable character set are automatically
4329		included in this class.
4330	<b>blank</b>	Define characters to be classified as <blank> characters.
4331		In the POSIX locale, only the <space> and <tab> shall be included.
4332		In a locale definition file, no character specified for the keywords <b>upper</b> ,
4333		<b>lower</b> , <b>alpha</b> , <b>digit</b> , <b>graph</b> , or <b>xdigit</b> shall be specified, and none of the
4334		characters <form-feed>, <newline>, <carriage-return>, and <vertical-tab> of
4335		the portable character set shall be specified. The <space> and <tab> are
4336		automatically included in this class.
4337	<b>charclass</b>	Define one or more locale-specific character class names as strings separated
4338		by <semicolon> characters. Each named character class can then be defined
4339		subsequently in the <i>LC_CTYPE</i> definition. A character class name shall consist
4340		of at least one and at most {CHARCLASS_NAME_MAX} bytes of
4341		alphanumeric characters from the portable filename character set. The first
4342		character of a character class name shall not be a digit. The name shall not
4343		match any of the <i>LC_CTYPE</i> keywords defined in this volume of
4344		POSIX.1-2024. Future versions of this standard will not specify any <i>LC_CTYPE</i>
4345	keywords containing uppercase letters.	
4346	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific
4347		character class. In the POSIX locale, locale-specific named character classes
4348		need not exist.

4349 If a class name is defined by a **charclass** keyword, but no characters are  
 4350 subsequently assigned to it, this is not an error; it represents a class without  
 4351 any characters belonging to it.

4352 The *charclass-name* can be used as the *property* argument to the *wctype()*  
 4353 function, in regular expression and shell pattern-matching bracket  
 4354 expressions, and by the *tr* command.

4355 **toupper** Define the mapping of lowercase letters to uppercase letters.

4356 In the POSIX locale, the 26 lowercase characters:

4357 a b c d e f g h i j k l m n o p q r s t u v w x y z

4358 shall be mapped to the corresponding 26 uppercase characters:

4359 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

4360 In a locale definition file, the operand shall consist of character pairs,  
 4361 separated by <semicolon> characters. The characters in each character pair  
 4362 shall be separated by a <comma> and the pair enclosed by parentheses. The  
 4363 first character in each pair is the lowercase letter, the second the corresponding  
 4364 uppercase letter. Only characters specified for the keywords **lower** and **upper**  
 4365 shall be specified. The lowercase letters <a> to <z>, and their corresponding  
 4366 uppercase letters <A> to <Z>, of the portable character set are automatically  
 4367 included in this mapping, but only when the **toupper** keyword is omitted  
 4368 from the locale definition.

4369 **tolower** Define the mapping of uppercase letters to lowercase letters.

4370 In the POSIX locale, the 26 uppercase characters:

4371 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

4372 shall be mapped to the corresponding 26 lowercase characters:

4373 a b c d e f g h i j k l m n o p q r s t u v w x y z

4374 In a locale definition file, the operand shall consist of character pairs,  
 4375 separated by <semicolon> characters. The characters in each character pair  
 4376 shall be separated by a <comma> and the pair enclosed by parentheses. The  
 4377 first character in each pair is the uppercase letter, the second the  
 4378 corresponding lowercase letter. Only characters specified for the keywords  
 4379 **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from  
 4380 the locale definition, the mapping is the reverse mapping of the one specified  
 4381 for **toupper**.

4382 The following table shows the character class combinations allowed:

4383

Table 7-1 Valid Character Class Combinations

4384

4385

4386

4387

4388

4389

4390

4391

4392

4393

4394

4395

4396

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
upper	—	—	A	x	x	x	x	A	A	—	x
lower	—	—	A	x	x	x	x	A	A	—	x
alpha	—	—	—	x	x	x	x	A	A	—	x
digit	x	x	x	—	x	x	x	A	A	A	x
space	x	x	x	x	—	—	*	*	*	x	—
cntrl	x	x	x	x	—	—	x	x	x	x	—
punct	x	x	x	x	—	x	—	A	A	x	—
graph	—	—	—	—	—	x	—	—	A	—	—
print	—	—	—	—	—	x	—	—	—	—	—
xdigit	—	—	—	—	x	x	x	A	A	—	x
blank	x	x	x	x	A	—	*	*	*	x	—

4397

Notes:

4398

4399

4400

4401

4402

4403

4404

4405

- Explanation of codes:  
 A Automatically included; see text.  
 — Permitted.  
 x Mutually-exclusive.  
 \* See note 2.
- The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters can be classified as any of **punct**, **graph**, or **print**.

7.3.1.1 LC\_CTYPE Category in the POSIX Locale

The minimum character classifications for the POSIX locale follow; the code listing depicts the *localedef* input, and the table represents the same information, sorted by character. Implementations may add additional characters to the **cntrl** and **punct** classifications but shall not make any other additions.

```

LC_CTYPE
# The following is the minimum POSIX locale LC_CTYPE.
# "alpha" is by definition "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by definition "alnum", "punct", and the <space>
# "graph" is by definition "alnum" and "punct"
#
upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
         <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
#
lower    <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
         <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
#
digit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
         <seven>;<eight>;<nine>
#
space    <tab>;<newline>;<vertical-tab>;<form-feed>;\

```

```

4428         <carriage-return>;<space>
4429     #
4430     cntrl    <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4431             <form-feed>;<carriage-return>;\
4432             <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4433             <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4434             <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4435             <IS1>;<DEL>
4436     #
4437     punct   <exclamation-mark>;<quotation-mark>;<number-sign>;\
4438             <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4439             <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4440             <plus-sign>;<comma>;<hyphen-minus>;<period>;<slash>;\
4441             <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4442             <greater-than-sign>;<question-mark>;<commercial-at>;\
4443             <left-square-bracket>;<backslash>;<right-square-bracket>;\
4444             <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4445             <vertical-line>;<right-curly-bracket>;<tilde>
4446     #
4447     xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4448             <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4449     #
4450     blank   <space>;<tab>
4451     #
4452     toupper (<a>, <A>); (<b>, <B>); (<c>, <C>); (<d>, <D>); (<e>, <E>); \
4453             (<f>, <F>); (<g>, <G>); (<h>, <H>); (<i>, <I>); (<j>, <J>); \
4454             (<k>, <K>); (<l>, <L>); (<m>, <M>); (<n>, <N>); (<o>, <O>); \
4455             (<p>, <P>); (<q>, <Q>); (<r>, <R>); (<s>, <S>); (<t>, <T>); \
4456             (<u>, <U>); (<v>, <V>); (<w>, <W>); (<x>, <X>); (<y>, <Y>); (<z>, <Z>)
4457     #
4458     tolower (<A>, <a>); (<B>, <b>); (<C>, <c>); (<D>, <d>); (<E>, <e>); \
4459             (<F>, <f>); (<G>, <g>); (<H>, <h>); (<I>, <i>); (<J>, <j>); \
4460             (<K>, <k>); (<L>, <l>); (<M>, <m>); (<N>, <n>); (<O>, <o>); \
4461             (<P>, <p>); (<Q>, <q>); (<R>, <r>); (<S>, <s>); (<T>, <t>); \
4462             (<U>, <u>); (<V>, <v>); (<W>, <w>); (<X>, <x>); (<Y>, <y>); (<Z>, <z>)
4463     END LC_CTYPE

```



	Symbolic Name	Other Case	Character Classes
4464	<NUL>		cntrl
4465	<SOH>		cntrl
4466	<STX>		cntrl
4467	<ETX>		cntrl
4468	<EOT>		cntrl
4469	<ENQ>		cntrl
4470	<ACK>		cntrl
4471	<alert>		cntrl
4472	<backspace>		cntrl
4473	<tab>		cntrl, space, blank
4474	<newline>		cntrl, space
4475	<vertical-tab>		cntrl, space
4476	<form-feed>		cntrl, space
4477	<carriage-return>		cntrl, space
4478	<SO>		cntrl
4479	<SI>		cntrl
4480	<DLE>		cntrl
4481	<DC1>		cntrl
4482	<DC2>		cntrl
4483	<DC3>		cntrl
4484	<DC4>		cntrl
4485	<NAK>		cntrl
4486	<SYN>		cntrl
4487	<ETB>		cntrl
4488	<CAN>		cntrl
4489	<EM>		cntrl
4490	<SUB>		cntrl
4491	<ESC>		cntrl
4492	<IS4>		cntrl
4493	<IS3>		cntrl
4494	<IS2>		cntrl
4495	<IS1>		cntrl
4496	<space>		space, print, blank
4497	<exclamation-mark>		punct, print, graph
4498	<quotation-mark>		punct, print, graph
4499	<number-sign>		punct, print, graph
4500	<dollar-sign>		punct, print, graph
4501	<percent-sign>		punct, print, graph
4502	<ampersand>		punct, print, graph
4503	<apostrophe>		punct, print, graph
4504	<left-parenthesis>		punct, print, graph
4505	<right-parenthesis>		punct, print, graph
4506	<asterisk>		punct, print, graph
4507	<plus-sign>		punct, print, graph
4508	<comma>		punct, print, graph
4509	<hyphen-minus>		punct, print, graph
4510	<period>		punct, print, graph
4511	<slash>		punct, print, graph
4512	<zero>		digit, xdigit, print, graph
4513	<one>		digit, xdigit, print, graph
4514	<two>		digit, xdigit, print, graph
4515			

	Symbolic Name	Other Case	Character Classes
4516	<three>		digit, xdigit, print, graph
4517	<four>		digit, xdigit, print, graph
4518	<five>		digit, xdigit, print, graph
4519	<six>		digit, xdigit, print, graph
4520	<seven>		digit, xdigit, print, graph
4521	<eight>		digit, xdigit, print, graph
4522	<nine>		digit, xdigit, print, graph
4523	<colon>		punct, print, graph
4524	<semicolon>		punct, print, graph
4525	<less-than-sign>		punct, print, graph
4526	<equals-sign>		punct, print, graph
4527	<greater-than-sign>		punct, print, graph
4528	<question-mark>		punct, print, graph
4529	<commercial-at>		punct, print, graph
4530	<A>	<a>	upper, xdigit, alpha, print, graph
4531	<B>	<b>	upper, xdigit, alpha, print, graph
4532	<C>	<c>	upper, xdigit, alpha, print, graph
4533	<D>	<d>	upper, xdigit, alpha, print, graph
4534	<E>	<e>	upper, xdigit, alpha, print, graph
4535	<F>	<f>	upper, xdigit, alpha, print, graph
4536	<G>	<g>	upper, alpha, print, graph
4537	<H>	<h>	upper, alpha, print, graph
4538	<I>	<i>	upper, alpha, print, graph
4539	<J>	<j>	upper, alpha, print, graph
4540	<K>	<k>	upper, alpha, print, graph
4541	<L>	<l>	upper, alpha, print, graph
4542	<M>	<m>	upper, alpha, print, graph
4543	<N>	<n>	upper, alpha, print, graph
4544	<O>	<o>	upper, alpha, print, graph
4545	<P>	<p>	upper, alpha, print, graph
4546	<Q>	<q>	upper, alpha, print, graph
4547	<R>	<r>	upper, alpha, print, graph
4548	<S>	<s>	upper, alpha, print, graph
4549	<T>	<t>	upper, alpha, print, graph
4550	<U>	<u>	upper, alpha, print, graph
4551	<V>	<v>	upper, alpha, print, graph
4552	<W>	<w>	upper, alpha, print, graph
4553	<X>	<x>	upper, alpha, print, graph
4554	<Y>	<y>	upper, alpha, print, graph
4555	<Z>	<z>	upper, alpha, print, graph
4556	<left-square-bracket>		punct, print, graph
4557	<backslash>		punct, print, graph
4558	<right-square-bracket>		punct, print, graph
4559	<circumflex>		punct, print, graph
4560	<underscore>		punct, print, graph
4561	<grave-accent>		punct, print, graph
4562	<a>	<A>	lower, xdigit, alpha, print, graph
4563	<b>	<B>	lower, xdigit, alpha, print, graph
4564	<c>	<C>	lower, xdigit, alpha, print, graph
4565	<d>	<D>	lower, xdigit, alpha, print, graph
4566	<e>	<E>	lower, xdigit, alpha, print, graph

	Symbolic Name	Other Case	Character Classes
4568	<f>	<F>	lower, xdigit, alpha, print, graph
4569	<g>	<G>	lower, alpha, print, graph
4570	<h>	<H>	lower, alpha, print, graph
4571	<i>	<I>	lower, alpha, print, graph
4572	<j>	<J>	lower, alpha, print, graph
4573	<k>	<K>	lower, alpha, print, graph
4574	<l>	<L>	lower, alpha, print, graph
4575	<m>	<M>	lower, alpha, print, graph
4576	<n>	<N>	lower, alpha, print, graph
4577	<o>	<O>	lower, alpha, print, graph
4578	<p>	<P>	lower, alpha, print, graph
4579	<q>	<Q>	lower, alpha, print, graph
4580	<r>	<R>	lower, alpha, print, graph
4581	<s>	<S>	lower, alpha, print, graph
4582	<t>	<T>	lower, alpha, print, graph
4583	<u>	<U>	lower, alpha, print, graph
4584	<v>	<V>	lower, alpha, print, graph
4585	<w>	<W>	lower, alpha, print, graph
4586	<x>	<X>	lower, alpha, print, graph
4587	<y>	<Y>	lower, alpha, print, graph
4588	<z>	<Z>	lower, alpha, print, graph
4589	<left-curly-bracket>		punct, print, graph
4590	<vertical-line>		punct, print, graph
4591	<right-curly-bracket>		punct, print, graph
4592	<tilde>		punct, print, graph
4593	<DEL>		cntrl
4594			

4595 **7.3.2 LC\_COLLATE**

4596 The *LC\_COLLATE* category provides a collation sequence definition for numerous utilities in the  
 4597 Shell and Utilities volume of POSIX.1-2024 (*ls*, *sort*, and so on), regular expression matching (see  
 4598 [Chapter 9](#), on page 179), and the *strcoll()*, *strxfrm()*, *wcscoll()*, and *wcsxfrm()* functions in the  
 4599 System Interfaces volume of POSIX.1-2024.

4600 A collation sequence definition shall define the relative order between collating elements  
 4601 (characters and multi-character collating elements) in the locale. This order is expressed in terms  
 4602 of collation values; that is, by assigning each element one or more collation values (also known  
 4603 as collation weights). This does not imply that implementations shall assign such values, but  
 4604 that ordering of strings using the resultant collation definition in the locale behaves as if such  
 4605 assignment is done and used in the collation process. At least the following capabilities are  
 4606 provided:

- 4607 1. **Multi-character collating elements.** Specification of multi-character collating elements  
 4608 (that is, sequences of two or more characters to be collated as an entity).
- 4609 2. **User-defined ordering of collating elements.** Each collating element shall be assigned a  
 4610 collation value defining its order in the character (or basic) collation sequence. This  
 4611 ordering is used by regular expressions and pattern matching and, unless collation  
 4612 weights are explicitly specified, also as the collation weight to be used in sorting.
- 4613 3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or  
 4614 more (up to the limit [COLL\_WEIGHTS\_MAX]), as defined in [<limits.h>](#) collating  
 4615 weights for use in sorting. The first weight is hereafter referred to as the primary weight.

- 4616 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4617 5. **Equivalence class definition.** Two or more collating elements have the same collation
- 4618 value (primary weight).
- 4619 6. **Ordering by weights.** When two strings are compared to determine their relative order,
- 4620 the two strings are first broken up into a series of collating elements; the elements in each
- 4621 successive pair of elements are then compared according to the relative primary weights
- 4622 for the elements. If equal, and more than one weight has been assigned, then the pairs of
- 4623 collating elements are re-compared according to the relative subsequent weights, until
- 4624 either a pair of collating elements compare unequal or the weights are exhausted.

4625 All implementation-provided locales (either preinstalled or provided as locale definitions which

4626 can be installed later) shall define a collation sequence that has a total ordering of all characters

4627 unless the locale name has an '@' modifier indicating that it has a special collation sequence (for

4628 example, @iCase could indicate that each upper and lowercase character pair collates equally).

4629 **Note:** Users installing their own locales should ensure that they define a collation sequence with a

4630 total ordering of all characters unless an '@' modifier in the locale name (such as @iCase)

4631 indicates that it has a special collation sequence. As <NUL> is reserved as the string terminator

4632 for most usages of LC\_COLLATE, it is the responsibility of the locale writer to ensure <NUL>

4633 has the lowest primary weight in a collation ordering for the interfaces to behave in the way

4634 users typically expect. Unusual behavior may result if it has any other collation order

4635 weighting, or is subject to IGNORE.

4636 The following keywords shall be recognized in a collation sequence definition. They are

4637 described in detail in the following sections.

4638	<b>copy</b>	Specify the name of an existing locale which shall be used as the
4639		definition of this category. If this keyword is specified, no other keyword
4640		shall be specified.
4641	<b>collating-element</b>	Define a collating-element symbol representing a multi-character
4642		collating element. This keyword is optional.
4643	<b>collating-symbol</b>	Define a collating symbol for use in collation order statements. This
4644		keyword is optional.
4645	<b>order_start</b>	Define collation rules. This statement shall be followed by one or more
4646		collation order statements, assigning character collation values and
4647		collation weights to collating elements.
4648	<b>order_end</b>	Specify the end of the collation-order statements.

#### 4649 7.3.2.1 The collating-element Keyword

4650 In addition to the collating elements in the character set, the **collating-element** keyword can be

4651 used to define multi-character collating elements. The syntax is as follows:

```
4652 "collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

4653 The <collating-symbol> operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A <collating-element> defined via this keyword is only recognized with the LC\_COLLATE category.

4658 For example:

```
4659 collating-element <ch> from "<c><h>"
```

4660 collating-element <e-acute> from "<acute><e>"  
 4661 collating-element <ll> from "ll"

### 4662 7.3.2.2 *The collating-symbol Keyword*

4663 This keyword shall be used to define symbols for use in collation sequence statements; that is,  
 4664 between the **order\_start** and the **order\_end** keywords. The syntax is as follows:

```
4665 "collating-symbol %s\n", <collating-symbol>
```

4666 The <collating-symbol> shall be a symbolic name, enclosed between angle brackets ('<' and  
 4667 '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any  
 4668 other symbolic name defined in this collation definition. A <collating-symbol> defined via this  
 4669 keyword is only recognized within the *LC\_COLLATE* category.

4670 For example:

```
4671 collating-symbol <UPPER_CASE>  

  4672 collating-symbol <HIGH>
```

4673 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative  
 4674 position in the character order sequence. While such a symbolic name does not represent any  
 4675 collating element, it can be used as a weight.

### 4676 7.3.2.3 *The order\_start Keyword*

4677 The **order\_start** keyword shall precede collation order entries and also define the number of  
 4678 weights for this collation sequence definition and other collation rules. The syntax is as follows:

```
4679 "order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...
```

4680 The operands to the **order\_start** keyword are optional. If present, the operands define rules to be  
 4681 applied when strings are compared. The number of operands define how many weights each  
 4682 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the  
 4683 first operand defines rules to be applied when comparing strings using the first (primary)  
 4684 weight; the second when comparing strings using the second weight, and so on. Operands shall  
 4685 be separated by <semicolon> characters (';'). Each operand shall consist of one or more  
 4686 collation directives, separated by <comma> characters (','). If the number of operands exceeds  
 4687 the {*COLL\_WEIGHTS\_MAX*} limit, the utility shall issue a warning message. The following  
 4688 directives shall be supported:

4689 **forward** Specifies that comparison operations for the weight level shall proceed from start  
 4690 of string towards the end of string.

4691 **backward** Specifies that comparison operations for the weight level shall proceed from end of  
 4692 string towards the beginning of string.

4693 **position** Specifies that comparison operations for the weight level shall consider the relative  
 4694 position of elements in the strings not subject to **IGNORE**. The string containing  
 4695 an element not subject to **IGNORE** after the fewest collating elements subject to  
 4696 **IGNORE** from the start of the compare shall collate first. If both strings contain a  
 4697 character not subject to **IGNORE** in the same relative position, the collating values  
 4698 assigned to the elements shall determine the ordering. In case of equality,  
 4699 subsequent characters not subject to **IGNORE** shall be considered in the same  
 4700 manner.

4701 The directives **forward** and **backward** are mutually-exclusive.

4702 If no operands are specified, a single **forward** operand shall be assumed.

4703 For example:

```
4704 order_start    forward;backward
```

#### 4705 7.3.2.4 Collation Order

4706 The **order\_start** keyword shall be followed by collating identifier entries. The syntax for the  
4707 collating element entries is as follows:

```
4708 "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...
```

4709 Each *collating-identifier* shall consist of either a character (in any of the forms defined in [Section](#)  
4710 [7.3](#), on page 128), a *collating-element*, a *collating-symbol*, an ellipsis, or the special symbol  
4711 **UNDEFINED**. The order in which collating elements are specified determines the character  
4712 order sequence, such that each collating element shall compare less than the elements following  
4713 it.

4714 A *collating-element* shall be used to specify multi-character collating elements, and indicates  
4715 that the character sequence specified via the *collating-element* is to be collated as a unit and in  
4716 the relative order specified by its place.

4717 A *collating-symbol* can be used to define a position in the relative order for use in weights. No  
4718 weights shall be specified with a *collating-symbol*.

4719 The ellipsis symbol specifies that a sequence of characters shall collate according to their  
4720 encoded character values. It shall be interpreted as indicating that all characters with a coded  
4721 character set value higher than the value of the character in the preceding line, and lower than  
4722 the coded character set value for the character in the following line, in the current coded  
4723 character set, shall be placed in the character collation order between the previous and the  
4724 following character in ascending order according to their coded character set values. An initial  
4725 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing  
4726 ellipsis as if the following line specified the highest coded character set value in the current  
4727 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do  
4728 not specify characters in the current coded character set. The use of the ellipsis symbol ties the  
4729 definition to a specific coded character set and may preclude the definition from being portable  
4730 between implementations.

4731 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not  
4732 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character  
4733 collation order at the point indicated by the symbol, and in ascending order according to their  
4734 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded  
4735 character set contains characters not specified in this section, the utility shall issue a warning  
4736 message and place such characters at the end of the character collation order.

4737 The optional operands for each collation-element shall be used to define the primary, secondary,  
4738 or subsequent weights for the collating element. The first operand specifies the relative primary  
4739 weight, the second the relative secondary weight, and so on. Two or more collation-elements can  
4740 be assigned the same weight; they belong to the same "equivalence class" if they have the same  
4741 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**  
4742 are removed, unless the **position** collation directive is specified for the corresponding level with  
4743 the **order\_start** keyword. Then each successive pair of elements shall be compared according to  
4744 the relative weights for the elements. If the two strings compare equal, the process shall be  
4745 repeated for the next weight level, up to the limit {COLL\_WEIGHTS\_MAX}.

4746 Weights shall be assigned such that the collation sequence has a total ordering of all characters

4747 unless an '@' modifier in the locale name indicates that it has a special collation sequence.

4748 Weights shall be expressed as characters (in any of the forms specified in [Section 7.3](#), on page  
4749 128), `<collating-symbol>s`, `<collating-element>s`, an ellipsis, or the special symbol **IGNORE**. A  
4750 single character, a `<collating-symbol>`, or a `<collating-element>` shall represent the relative position  
4751 in the character collating sequence of the character or symbol, rather than the character or  
4752 characters themselves. Thus, rather than assigning absolute values to weights, a particular  
4753 weight is expressed using the relative order value assigned to a collating element based on its  
4754 order in the character collation sequence.

4755 One-to-many mapping is indicated by specifying two or more concatenated characters or  
4756 symbolic names. For example, if the `<eszet>` is given the string "`<s><s>`" as a weight,  
4757 comparisons are performed as if all occurrences of the `<eszet>` are replaced by "`<s><s>`"  
4758 (assuming that "`<s>`" has the collating weight "`<s>`"). If it is necessary to define `<eszet>` and  
4759 "`<s><s>`" as an equivalence class, then a collating element needs to be defined for the string  
4760 "`ss`".

4761 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the  
4762 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special  
4763 symbol shall by default be assigned the same primary weight (that is, they belong to the same  
4764 equivalence class) if the collation order has more than one weight level. If the collation order has  
4765 only one weight level, these characters shall be assigned unique primary weights, equal to the  
4766 relative order of their character in the character collation sequence.

4767 An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence  
4768 shall have unique weights, equal to the relative order of their character in the character collation  
4769 sequence. The use of the ellipsis as a weight shall be treated as an error if the collating element is  
4770 neither an ellipsis nor the special symbol **UNDEFINED**.

4771 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using  
4772 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that  
4773 is, as if the string did not contain the collating element. In regular expressions and pattern  
4774 matching, all characters that are subject to **IGNORE** in their primary weight form an  
4775 equivalence class.

4776 An empty operand shall be interpreted as the collating element itself.

4777 For example, the order statement:

4778 `<a> <a>; <a>`

4779 is equal to:

4780 `<a>`

4781 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be  
4782 interpreted as the value of each character defined by the ellipsis.

4783 The collation order as defined in this section affects the interpretation of bracket expressions in  
4784 regular expressions (see [Section 9.3.5](#), on page 182).

```

4785     For example:
4786     order_start  forward;backward
4787     <NUL>       <NUL>;<NUL>
4788     <LOW>
4789     <space>     <LOW>;<space>
4790     ...        <LOW>;...
4791     <a>         <a>;<a>
4792     <a-acute>   <a>;<a-acute>
4793     <a-grave>  <a>;<a-grave>
4794     <A>        <a>;<A>
4795     <A-acute>  <a>;<A-acute>
4796     <A-grave>  <a>;<A-grave>
4797     <ch>       <ch>;<ch>
4798     <Ch>       <ch>;<Ch>
4799     <s>        <s>;<s>
4800     <eszet>    "<s><s>"; "<eszet><eszet>"
4801     UNDEFINED  IGNORE;...
4802     order_end

```

4803 This example is interpreted as follows:

- 4804 1. All characters between <space> and 'a' shall have the same primary equivalence class  
4805 and individual secondary weights based on their ordinal encoded values.
- 4806 2. All characters based on the uppercase or lowercase character 'a' belong to the same  
4807 primary equivalence class.
- 4808 3. The multi-character collating element <ch> is represented by the collating symbol <ch>  
4809 and belongs to the same primary equivalence class as the multi-character collating  
4810 element <Ch>.
- 4811 4. The **UNDEFINED** means that all characters not specified in this definition (explicitly or  
4812 via the ellipsis) shall be ignored when comparing primary weights, and have individual  
4813 secondary weights based on their ordinal encoded values.

#### 4814 7.3.2.5 The *order\_end* Keyword

4815 The collating order entries shall be terminated with an **order\_end** keyword.

#### 4816 7.3.2.6 *LC\_COLLATE* Category in the POSIX Locale

4817 The minimum collation sequence definition of the POSIX locale follows; the code listing depicts  
4818 the *localedef* input. All characters not explicitly listed here shall be inserted in the character  
4819 collation order after the listed characters and shall be assigned unique primary weights. If the  
4820 listed characters have ASCII encoding, the other characters shall be in ascending order according  
4821 to their coded character set values; otherwise, the order of the other characters is unspecified.  
4822 The collation sequence shall not include any multi-character collating elements.

```

4823 LC_COLLATE
4824 # This is the minimum input for the POSIX locale definition for the
4825 # LC_COLLATE category. Characters in this list are in the same order
4826 # as in the ASCII codeset.
4827 order_start forward
4828 <NUL>
4829 <SOH>

```



4830	<STX>
4831	<ETX>
4832	<EOT>
4833	<ENQ>
4834	<ACK>
4835	<alert>
4836	<backspace>
4837	<tab>
4838	<newline>
4839	<vertical-tab>
4840	<form-feed>
4841	<carriage-return>
4842	<SO>
4843	<SI>
4844	<DLE>
4845	<DC1>
4846	<DC2>
4847	<DC3>
4848	<DC4>
4849	<NAK>
4850	<SYN>
4851	<ETB>
4852	<CAN>
4853	<EM>
4854	<SUB>
4855	<ESC>
4856	<IS4>
4857	<IS3>
4858	<IS2>
4859	<IS1>
4860	<space>
4861	<exclamation-mark>
4862	<quotation-mark>
4863	<number-sign>
4864	<dollar-sign>
4865	<percent-sign>
4866	<ampersand>
4867	<apostrophe>
4868	<left-parenthesis>
4869	<right-parenthesis>
4870	<asterisk>
4871	<plus-sign>
4872	<comma>
4873	<hyphen-minus>
4874	<period>
4875	<slash>
4876	<zero>
4877	<one>
4878	<two>
4879	<three>
4880	<four>
4881	<five>
4882	<six>

4883	<seven>
4884	<eight>
4885	<nine>
4886	<colon>
4887	<semicolon>
4888	<less-than-sign>
4889	<equals-sign>
4890	<greater-than-sign>
4891	<question-mark>
4892	<commercial-at>
4893	<A>
4894	<B>
4895	<C>
4896	<D>
4897	<E>
4898	<F>
4899	<G>
4900	<H>
4901	<I>
4902	<J>
4903	<K>
4904	<L>
4905	<M>
4906	<N>
4907	<O>
4908	<P>
4909	<Q>
4910	<R>
4911	<S>
4912	<T>
4913	<U>
4914	<V>
4915	<W>
4916	<X>
4917	<Y>
4918	<Z>
4919	<left-square-bracket>
4920	<backslash>
4921	<right-square-bracket>
4922	<circumflex>
4923	<underscore>
4924	<grave-accent>
4925	<a>
4926	<b>
4927	<c>
4928	<d>
4929	<e>
4930	<f>
4931	<g>
4932	<h>
4933	<i>
4934	<j>
4935	<k>

```

4936      <l>
4937      <m>
4938      <n>
4939      <o>
4940      <p>
4941      <q>
4942      <r>
4943      <s>
4944      <t>
4945      <u>
4946      <v>
4947      <w>
4948      <x>
4949      <y>
4950      <z>
4951      <left-curly-bracket>
4952      <vertical-line>
4953      <right-curly-bracket>
4954      <tilde>
4955      <DEL>
4956      order_end
4957      #
4958      END LC_COLLATE

```

### 4959 7.3.3 LC\_MONETARY

4960 The *LC\_MONETARY* category shall define the rules and symbols that are used to format  
 4961 monetary numeric information.

4962 This information is available through the *localeconv()* function and is used by the *strfmon()*  
 4963 function.

4964 Some of the information is also available in an alternative form via the *nl\_langinfo()* function  
 4965 (see CRNCYSTR in [<langinfo.h>](#)).

4966 The following items are defined in this category of the locale. The item names are the keywords  
 4967 recognized by the *localedef* utility when defining a locale. They are also similar to the member  
 4968 names of the **lconv** structure defined in [<locale.h>](#); see [<locale.h>](#) for the exact symbols in the  
 4969 header. The *localeconv()* function returns {CHAR\_MAX} for unavailable integer items and the  
 4970 empty string (" ") for unavailable or size zero string items.

4971 In a locale definition file, the operands are strings, formatted as indicated by the grammar in  
 4972 [Section 7.4](#) (on page 160). For some keywords, the strings can contain only integers. Keywords  
 4973 that are not provided, or integer keywords set to -1, can be used to indicate that the value is not  
 4974 available in the locale. String values set to the empty string (" ") can be used to indicate that the  
 4975 value is available and is an empty string, or that the value is not available. The following  
 4976 keywords shall be recognized:

4977 **copy** Specify the name of an existing locale which shall be used as the  
 4978 definition of this category. If this keyword is specified, no other keyword  
 4979 shall be specified.

4980 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4981	<b>int_curr_symbol</b>	The international currency symbol. The operand shall be a four-character string, with the first three characters containing the alphabetic international currency symbol. The international currency symbol should be chosen in accordance with those specified in the ISO 4217 standard. The fourth character shall be the character used to separate the international currency symbol from the monetary quantity.
4982		
4983		
4984		
4985		
4986		
4987	<b>currency_symbol</b>	The string that shall be used as the local currency symbol.
4988	<b>mon_decimal_point</b>	The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in monetary formatted quantities.
4989		
4990	<b>mon_thousands_sep</b>	The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities.
4991		
4992		
4993	<b>mon_grouping</b>	Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by <semicolon> characters. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not -1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is -1, then no further grouping shall be performed.
4994		
4995		
4996		
4997		
4998		
4999		
5000		
5001	<b>positive_sign</b>	A string that shall be used to indicate a non-negative-valued formatted monetary quantity.
5002		
5003	<b>negative_sign</b>	A string that shall be used to indicate a negative-valued formatted monetary quantity.
5004		
5005	<b>int_frac_digits</b>	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using <b>int_curr_symbol</b> .
5006		
5007		
5008	<b>frac_digits</b>	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using <b>currency_symbol</b> .
5009		
5010		
5011	<b>p_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
5012		
5013		
5014	<b>p_sep_by_space</b>	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the value for a non-negative formatted monetary quantity.
5015		
5016		The values of <b>p_sep_by_space</b> , <b>n_sep_by_space</b> , <b>int_p_sep_by_space</b> , and <b>int_n_sep_by_space</b> are interpreted according to the following:
5017		
5018		0 No <space> separates the currency symbol and value.
5019		1 If the currency symbol and sign string are adjacent, a <space> separates them from the value; otherwise, a <space> separates the currency symbol from the value.
5020		
5021		
5022		2 If the currency symbol and sign string are adjacent, a <space> separates them; otherwise, a <space> separates the sign string from the value.
5023		
5024		

5025	<b>n_cs_precedes</b>	An integer set to 1 if the <b>currency_symbol</b> precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
5026		
5027		
5028	<b>n_sep_by_space</b>	Set to a value indicating the separation of the <b>currency_symbol</b> , the sign string, and the value for a negative formatted monetary quantity.
5029		
5030	<b>p_sign_posn</b>	An integer set to a value indicating the positioning of the <b>positive_sign</b> for a monetary quantity with a non-negative value. The following integer values shall be recognized for <b>int_n_sign_posn</b> , <b>int_p_sign_posn</b> , <b>n_sign_posn</b> , and <b>p_sign_posn</b> :
5031		
5032		
5033		
5034		0 Parentheses enclose the quantity and the <b>currency_symbol</b> .
5035		1 The sign string precedes the quantity and the <b>currency_symbol</b> .
5036		2 The sign string succeeds the quantity and the <b>currency_symbol</b> .
5037		3 The sign string precedes the <b>currency_symbol</b> .
5038		4 The sign string succeeds the <b>currency_symbol</b> .
5039	<b>n_sign_posn</b>	An integer set to a value indicating the positioning of the <b>negative_sign</b> for a negative formatted monetary quantity.
5040		
5041	<b>int_p_cs_precedes</b>	An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
5042		
5043		
5044	<b>int_n_cs_precedes</b>	An integer set to 1 if the <b>int_curr_symbol</b> precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
5045		
5046		
5047	<b>int_p_sep_by_space</b>	Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign string, and the value for a non-negative internationally formatted monetary quantity.
5048		
5049		
5050	<b>int_n_sep_by_space</b>	Set to a value indicating the separation of the <b>int_curr_symbol</b> , the sign string, and the value for a negative internationally formatted monetary quantity.
5051		
5052		
5053	<b>int_p_sign_posn</b>	An integer set to a value indicating the positioning of the <b>positive_sign</b> for a positive monetary quantity formatted with the international format.
5054		
5055	<b>int_n_sign_posn</b>	An integer set to a value indicating the positioning of the <b>negative_sign</b> for a negative monetary quantity formatted with the international format.
5056		
5057		Certain combinations of the <b>*_sign_posn</b> , <b>positive_sign</b> , and <b>negative_sign</b> values are invalid and shall not be accepted by <i>localedef</i> :
5058		
5059		• If <b>p_sign_posn</b> and <b>n_sign_posn</b> are both greater than 0, and <b>positive_sign</b> and <b>negative_sign</b> have the same value or are both either empty strings or omitted, this combination is invalid because it requires signs to be used but does not provide the means to distinguish negative from positive values using signs.
5060		
5061		
5062		
5063		• Likewise, if <b>int_p_sign_posn</b> and <b>int_n_sign_posn</b> are both greater than 0, and <b>positive_sign</b> and <b>negative_sign</b> have the same value or are both either empty strings or omitted, this combination is invalid.
5064		
5065		
5066		• If <b>p_sign_posn</b> and <b>n_sign_posn</b> are both 0, this combination is invalid because it requires parentheses to be used but does not provide the means to distinguish negative from positive values using parentheses.
5067		
5068		

- Likewise, if `int_p_sign_posn` and `int_n_sign_posn` are both 0, this combination is invalid.

### 5070 7.3.3.1 LC\_MONETARY Category in the POSIX Locale

5071 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the  
 5072 `localedef` input, the table representing the same information with the addition of `localeconv()` and  
 5073 `nl_langinfo()` formats. All values shall be unavailable in the POSIX locale.

```

5074 LC_MONETARY
5075 # This is the POSIX locale definition for
5076 # the LC_MONETARY category.
5077 #
5078 int_curr_symbol      ""
5079 currency_symbol     ""
5080 mon_decimal_point   ""
5081 mon_thousands_sep  ""
5082 mon_grouping        -1
5083 positive_sign       ""
5084 negative_sign       ""
5085 int_frac_digits     -1
5086 frac_digits         -1
5087 p_cs_precedes       -1
5088 p_sep_by_space      -1
5089 n_cs_precedes       -1
5090 n_sep_by_space      -1
5091 p_sign_posn         -1
5092 n_sign_posn         -1
5093 int_p_cs_precedes   -1
5094 int_p_sep_by_space  -1
5095 int_n_cs_precedes   -1
5096 int_n_sep_by_space  -1
5097 int_p_sign_posn     -1
5098 int_n_sign_posn     -1
5099 #
5100 END LC_MONETARY

```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
int_curr_symbol	—	N/A	" "	" "
currency_symbol	CRNCYSTR	N/A	" "	" "
mon_decimal_point	—	N/A	" "	" "
mon_thousands_sep	—	N/A	" "	" "
mon_grouping	—	N/A	" "	-1
positive_sign	—	N/A	" "	" "
negative_sign	—	N/A	" "	" "
int_frac_digits	—	N/A	{CHAR_MAX}	-1
frac_digits	—	N/A	{CHAR_MAX}	-1
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1
p_sep_by_space	—	N/A	{CHAR_MAX}	-1
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1
n_sep_by_space	—	N/A	{CHAR_MAX}	-1
p_sign_posn	—	N/A	{CHAR_MAX}	-1
n_sign_posn	—	N/A	{CHAR_MAX}	-1
int_p_cs_precedes	—	N/A	{CHAR_MAX}	-1
int_p_sep_by_space	—	N/A	{CHAR_MAX}	-1
int_n_cs_precedes	—	N/A	{CHAR_MAX}	-1
int_n_sep_by_space	—	N/A	{CHAR_MAX}	-1
int_p_sign_posn	—	N/A	{CHAR_MAX}	-1
int_n_sign_posn	—	N/A	{CHAR_MAX}	-1

The entry N/A indicates that the value is not available in the POSIX locale.

### 7.3.4 LC\_NUMERIC

The *LC\_NUMERIC* category shall define the rules and symbols that are used to format non-monetary numeric information. This information is available through the *localeconv()* function.

Some of the information is also available in an alternative form via the *nl\_langinfo()* function.

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the *lconv* structure defined in *<locale.h>*; see *<locale.h>* for the exact symbols in the header. The *localeconv()* function returns {CHAR\_MAX} for unavailable integer items and the empty string (" ") for unavailable or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in Section 7.4 (on page 160). For some keywords, the strings can only contain integers. Keywords that are not provided, or integer keywords set to -1, can be used to indicate that the value is not available in the locale. String values set to the empty string (" ") can be used to indicate that the value is available and is an empty string, or that the value is not available. The following keywords shall be recognized:

- copy** Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.  
**Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.
- decimal\_point** The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where standards limit the **decimal\_point** to a single byte,

5147 the result of specifying a multi-byte operand shall be unspecified.

5148 **thousands\_sep** The operand is a string containing the symbol that shall be used as a separator  
 5149 for groups of digits to the left of the decimal delimiter in numeric, non-  
 5150 monetary formatted monetary quantities. In contexts where standards limit  
 5151 the **thousands\_sep** to a single byte, the result of specifying a multi-byte  
 5152 operand shall be unspecified.

5153 **grouping** Define the size of each group of digits in formatted non-monetary quantities.  
 5154 The operand is a sequence of integers separated by <semicolon> characters.  
 5155 Each integer specifies the number of digits in each group, with the initial  
 5156 integer defining the size of the group immediately preceding the decimal  
 5157 delimiter, and the following integers defining the preceding groups. If the last  
 5158 integer is not -1, then the size of the previous group (if any) shall be  
 5159 repeatedly used for the remainder of the digits. If the last integer is -1, then no  
 5160 further grouping shall be performed.

5161 7.3.4.1 *LC\_NUMERIC Category in the POSIX Locale*

5162 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing  
 5163 depicting the *localedef* input, the table representing the same information with the addition of  
 5164 *localeconv()* values, and *nl\_langinfo()* constants.

```
5165 LC_NUMERIC
5166 # This is the POSIX locale definition for
5167 # the LC_NUMERIC category.
5168 #
5169 decimal_point    "<period>"
5170 thousands_sep    ""
5171 grouping         -1
5172 #
5173 END LC_NUMERIC
```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
<b>decimal_point</b>	RADIXCHAR	."	."	.
<b>thousands_sep</b>	THOUSEP	N/A	""	""
<b>grouping</b>	—	N/A	""	-1

5179 The entry N/A indicates that the value is not available in the POSIX locale.

5180 7.3.5 **LC\_TIME**

5181 The *LC\_TIME* category shall define the interpretation of the conversion specifications supported  
 5182 by the *date* utility and shall affect the behavior of the *strptime()*, *wcsftime()*, *strptime()*, and  
 5183 *nl\_langinfo()* functions. Since the interfaces for C-language access and locale definition differ  
 5184 significantly, they are described separately.



## 5185 7.3.5.1 LC\_TIME Locale Definition

5186 In a locale definition, the following mandatory keywords shall be recognized:

5187 **copy** Specify the name of an existing locale which shall be used as the definition of  
5188 this category. If this keyword is specified, no other keyword shall be specified.

5189 **abday** Define the abbreviated weekday names, corresponding to the %a conversion  
5190 specification (conversion specification in the *strptime()*, *wcsftime()*, and  
5191 *strptime()* functions). The operand shall consist of seven  
5192 <semicolon>-separated strings, each surrounded by double-quotes. The first  
5193 string shall be the abbreviated name of the day corresponding to Sunday, the  
5194 second the abbreviated name of the day corresponding to Monday, and so on.

5195 **day** Define the full weekday names, corresponding to the %A conversion  
5196 specification. The operand shall consist of seven <semicolon>-separated  
5197 strings, each surrounded by double-quotes. The first string is the full name of  
5198 the day corresponding to Sunday, the second the full name of the day  
5199 corresponding to Monday, and so on.

5200 **abmon** Define the abbreviated month names, corresponding to the %b conversion  
5201 specification. The operand shall consist of twelve <semicolon>-separated  
5202 strings, each surrounded by double-quotes. The first string shall be the  
5203 abbreviated name of the first month of the year (January), the second the  
5204 abbreviated name of the second month, and so on. For languages having both  
5205 a genitive (when used with a day number) and a nominative (no day number)  
5206 case, this operand shall be used to denote the genitive case.

5207 **ab\_alt\_mon** Define the abbreviated month names, corresponding to the %Ob conversion  
5208 specification. The operand shall consist of twelve <semicolon>-separated  
5209 strings, each surrounded by double-quotes. The first string shall be the  
5210 abbreviated name of the first month of the year (January), the second the  
5211 abbreviated name of the second month, and so on. For languages having both  
5212 a genitive (when used with a day number) and a nominative (no day number)  
5213 case, this operand shall be used to denote the nominative case.

5214 **mon** Define the full month names, corresponding to the %B conversion  
5215 specification. The operand shall consist of twelve <semicolon>-separated  
5216 strings, each surrounded by double-quotes. The first string shall be the full  
5217 name of the first month of the year (January), the second the full name of the  
5218 second month, and so on. For languages having both a genitive (when used  
5219 with a day number) and a nominative (no day number) case, this operand  
5220 shall be used to denote the genitive case.

5221 **alt\_mon** Define the full month names, corresponding to the %OB conversion  
5222 specification. The operand shall consist of twelve <semicolon>-separated  
5223 strings, each surrounded by double-quotes. The first string shall be the full  
5224 name of the first month of the year (January), the second the full name of the  
5225 second month, and so on. For languages having both a genitive (when used  
5226 with a day number) and a nominative (no day number) case, this operand  
5227 shall be used to denote the nominative case.

5228 **d\_t\_fmt** Define the appropriate date and time representation, corresponding to the %c  
5229 conversion specification. The operand shall consist of a string containing any  
5230 combination of characters and conversion specifications. In addition, the  
5231 string can contain escape sequences defined in the table in [Table 5-1](#) (on page  
5232 113) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').

5233	<b>d_fmt</b>	Define the appropriate date representation, corresponding to the %x conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 113).
5234		
5235		
5236		
5237	<b>t_fmt</b>	Define the appropriate time representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 113).
5238		
5239		
5240		
5241	<b>am_pm</b>	Define the appropriate representation of the <i>ante-meridiem</i> and <i>post-meridiem</i> strings, corresponding to the %p conversion specification. The operand shall consist of two strings, separated by a <semicolon>, each surrounded by double-quotes; the first string shall represent the <i>ante-meridiem</i> designation, the last string the <i>post-meridiem</i> designation. If and only if the 12-hour format is not supported in the locale, both strings shall be empty.
5242		
5243		
5244		
5245		
5246		
5247	<b>t_fmt_ampm</b>	Define the appropriate time representation in the 12-hour clock format with <b>am_pm</b> , corresponding to the %r conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If and only if the 12-hour format is not supported in the locale, the string shall be empty.
5248		
5249		
5250		
5251		
5252	<b>era</b>	Define how years are counted and displayed for each era in a locale. The operand shall consist of <semicolon>-separated strings. Each string shall be an era description segment with the format:  <i>direction:offset:start_date:end_date:era_name:era_format</i>  according to the definitions below. There can be as many era description segments as are necessary to describe the different eras.  <b>Note:</b> The start of an era might not be the earliest point in the era—it may be the latest. For example, the Christian era BC starts on the day before January 1, AD 1, and increases with earlier time.  <i>direction</i> Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .  <i>offset</i> The number of the year closest to the <i>start_date</i> in the era, corresponding to the %EY conversion specification.  <i>start_date</i> A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.  <i>end_date</i> The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.  <i>era_name</i> A string representing the name of the era, corresponding to the %EC conversion specification.  <i>era_format</i> A string for formatting the year in the era, corresponding to the %EY conversion specification.
5253		
5254		
5255		
5256		
5257		
5258		
5259		
5260		
5261		
5262		
5263		
5264		
5265		
5266		
5267		
5268		
5269		
5270		
5271		
5272		
5273		
5274		
5275		
5276		
5277		
5278		

5279	<b>era_d_fmt</b>	Define the format of the date in alternative era notation, corresponding to the %Ex conversion specification.
5280		
5281	<b>era_t_fmt</b>	Define the locale's appropriate alternative time format, corresponding to the %EX conversion specification.
5282		
5283	<b>era_d_t_fmt</b>	Define the locale's appropriate alternative date and time format, corresponding to the %Ec conversion specification.
5284		
5285	<b>alt_digits</b>	Define alternative symbols for digits, corresponding to the %O modified conversion specification. The operand shall consist of <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the alternative symbol corresponding with zero, the second string the symbol corresponding with one, and so on. Up to 100 alternative symbol strings can be specified. The %O modifier shall indicate that the string corresponding to the value specified via the conversion specification shall be used instead of the value.
5286		
5287		
5288		
5289		
5290		
5291		
5292		
5293	7.3.5.2	<i>LC_TIME C-Language Access</i>
5294		The following constants used to identify items of <i>langinfo</i> data can be used as arguments to the <i>nl_langinfo()</i> function to access information in the <i>LC_TIME</i> category. These constants are defined in the <b>&lt;langinfo.h&gt;</b> header.
5295		
5296		
5297	<b>ABDAY_x</b>	The abbreviated weekday names (for example, Sun), where <i>x</i> is a number from 1 to 7.
5298		
5299	<b>DAY_x</b>	The full weekday names (for example, Sunday), where <i>x</i> is a number from 1 to 7.
5300		
5301	<b>ABMON_x</b>	The abbreviated month names (for example, Jan), where <i>x</i> is a number from 1 to 12.
5302		
5303	<b>ABALTMON_x</b>	The alternative abbreviated month names (for example, Jan), where <i>x</i> is a number from 1 to 12.
5304		
5305	<b>MON_x</b>	The full month names (for example, January), where <i>x</i> is a number from 1 to 12.
5306		
5307	<b>ALTMON_x</b>	The alternative full month names (for example, January), where <i>x</i> is a number from 1 to 12.
5308		
5309	<b>D_T_FMT</b>	The appropriate date and time representation.
5310	<b>D_FMT</b>	The appropriate date representation.
5311	<b>T_FMT</b>	The appropriate time representation.
5312	<b>AM_STR</b>	The appropriate ante-meridiem affix; if <b>AM_STR</b> and <b>PM_STR</b> are both empty strings, the 12-hour format is not supported in the locale.
5313		
5314	<b>PM_STR</b>	The appropriate post-meridiem affix; if <b>AM_STR</b> and <b>PM_STR</b> are both empty strings, the 12-hour format is not supported in the locale.
5315		
5316	<b>T_FMT_AMPM</b>	The appropriate time representation in the 12-hour clock format; if the 12-hour format is not supported in the locale, this shall be either an empty string or a string specifying a 24-hour clock format.
5317		
5318		
5319	<b>ERA</b>	The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment shall have the format:
5320		
5321		

5322		<i>direction:offset:start_date:end_date:era_name:era_format</i>
5323		according to the definitions below. There can be as many era description
5324		segments as are necessary to describe the different eras. Era description
5325		segments are separated by <semicolon> characters.
5326	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate
5327		that years closer to the <i>start_date</i> have lower numbers than those
5328		closer to the <i>end_date</i> . The '-' character shall indicate that years
5329		closer to the <i>start_date</i> have higher numbers than those closer to
5330		the <i>end_date</i> .
5331	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era.
5332	<i>start_date</i>	A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the
5333		year, month, and day numbers respectively of the start of the era.
5334		Years prior to AD 1 shall be represented as negative numbers.
5335	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
5336		or one of the two special values "-*" or "+*". The value "-*"
5337		shall indicate that the ending date is the beginning of time. The
5338		value "+*" shall indicate that the ending date is the end of time.
5339	<i>era_name</i>	The era, corresponding to the %EC conversion specification.
5340	<i>era_format</i>	The format of the year in the era, corresponding to the %EY
5341		conversion specification.
5342	ERA_D_FMT	The era date format.
5343	ERA_T_FMT	The locale's appropriate alternative time format, corresponding to the %EX
5344		conversion specification.
5345	ERA_D_T_FMT	The locale's appropriate alternative date and time format, corresponding to
5346		the %Ec conversion specification.
5347	ALT_DIGITS	The alternative symbols for digits, corresponding to the %O conversion
5348		specification modifier. The value consists of <semicolon>-separated symbols.
5349		The first is the alternative symbol corresponding to zero, the second is the
5350		symbol corresponding to one, and so on. Up to 100 alternative symbols may
5351		be specified.

5352 7.3.5.3 LC\_TIME Category in the POSIX Locale

5353 The LC\_TIME category definition of the POSIX locale follows; the code listing depicts the

5354 *localedef* input; the table represents the same information with the addition of *localedef* keywords,

5355 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*

5356 functions, and *nl\_langinfo()* constants.

```

5357 LC_TIME
5358 # This is the POSIX locale definition for
5359 # the LC_TIME category.
5360 #
5361 # Abbreviated weekday names (%a)
5362 abday      "<S><u><n>"; "<M><o><n>"; "<T><u><e>"; "<W><e><d>"; \
5363           "<T><h><u>"; "<F><r><i>"; "<S><a><t>"
5364 #
5365 # Full weekday names (%A)
5366 day       "<S><u><n><d><a><y>"; "<M><o><n><d><a><y>"; \

```

```

5367         "<T><u><e><s><d><a><y>"; "<W><e><d><n><e><s><d><a><y>"; \
5368         "<T><h><u><r><s><d><a><y>"; "<F><r><i><d><a><y>"; \
5369         "<S><a><t><u><r><d><a><y>"
5370     #
5371     # Abbreviated month names (%b)
5372     abmon      "<J><a><n>"; "<F><e><b>"; "<M><a><r>"; \
5373              "<A><p><r>"; "<M><a><y>"; "<J><u><n>"; \
5374              "<J><u><l>"; "<A><u><g>"; "<S><e><p>"; \
5375              "<O><c><t>"; "<N><o><v>"; "<D><e><c>"
5376     #
5377     # Full month names (%B)
5378     mon        "<J><a><n><u><a><r><y>"; "<F><e><b><r><u><a><r><y>"; \
5379              "<M><a><r><c><h>"; "<A><p><r><i><l>"; \
5380              "<M><a><y>"; "<J><u><n><e>"; \
5381              "<J><u><l><y>"; "<A><u><g><u><s><t>"; \
5382              "<S><e><p><t><e><m><b><e><r>"; "<O><c><t><o><b><e><r>"; \
5383              "<N><o><v><e><m><b><e><r>"; "<D><e><c><e><m><b><e><r>"
5384     #
5385     # Equivalent of AM/PM (%p)          "AM"; "PM"
5386     am_pm      "<A><M>"; "<P><M>"
5387     #
5388     # Appropriate date and time representation (%c)
5389     #      "%a %b %e %H:%M:%S %Y"
5390     d_t_fmt     "<percent-sign><a><space><percent-sign><b>\
5391     <space><percent-sign><e><space><percent-sign><H>\
5392     <colon><percent-sign><M><colon><percent-sign><S>\
5393     <space><percent-sign><Y>"
5394     #
5395     # Appropriate date representation (%x)  "%m/%d/%y"
5396     d_fmt       "<percent-sign><m><slash><percent-sign><d>\
5397     <slash><percent-sign><y>"
5398     #
5399     # Appropriate time representation (%X)  "%H:%M:%S"
5400     t_fmt       "<percent-sign><H><colon><percent-sign><M>\
5401     <colon><percent-sign><S>"
5402     #
5403     # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5404     t_fmt_ampm  "<percent-sign><I><colon><percent-sign><M><colon>\
5405     <percent-sign><S><space><percent-sign><p>"
5406     #
5407     END LC_TIME

```

5408	localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
5410	d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
5411	d_fmt	D_FMT	%x	"%m/%d/%y"
5412	t_fmt	T_FMT	%X	"%H:%M:%S"
5413	am_pm	AM_STR	%p	"AM"
5414	am_pm	PM_STR	%p	"PM"
5415	t_fmt_ampm	T_FMT_AMP	%r	"%I:%M:%S %p"
5416	day	DAY_1	%A	"Sunday"
5417	day	DAY_2	%A	"Monday"

	localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
5418				
5419				
5420	day	DAY_3	%A	"Tuesday"
5421	day	DAY_4	%A	"Wednesday"
5422	day	DAY_5	%A	"Thursday"
5423	day	DAY_6	%A	"Friday"
5424	day	DAY_7	%A	"Saturday"
5425	abday	ABDAY_1	%a	"Sun"
5426	abday	ABDAY_2	%a	"Mon"
5427	abday	ABDAY_3	%a	"Tue"
5428	abday	ABDAY_4	%a	"Wed"
5429	abday	ABDAY_5	%a	"Thu"
5430	abday	ABDAY_6	%a	"Fri"
5431	abday	ABDAY_7	%a	"Sat"
5432	mon	MON_1	%B	"January"
5433	mon	MON_2	%B	"February"
5434	mon	MON_3	%B	"March"
5435	mon	MON_4	%B	"April"
5436	mon	MON_5	%B	"May"
5437	mon	MON_6	%B	"June"
5438	mon	MON_7	%B	"July"
5439	mon	MON_8	%B	"August"
5440	mon	MON_9	%B	"September"
5441	mon	MON_10	%B	"October"
5442	mon	MON_11	%B	"November"
5443	mon	MON_12	%B	"December"
5444	alt_mon	ALTMON_1	%OB	N/A
5445	alt_mon	ALTMON_2	%OB	N/A
5446	alt_mon	ALTMON_3	%OB	N/A
5447	alt_mon	ALTMON_4	%OB	N/A
5448	alt_mon	ALTMON_5	%OB	N/A
5449	alt_mon	ALTMON_6	%OB	N/A
5450	alt_mon	ALTMON_7	%OB	N/A
5451	alt_mon	ALTMON_8	%OB	N/A
5452	alt_mon	ALTMON_9	%OB	N/A
5453	alt_mon	ALTMON_10	%OB	N/A
5454	alt_mon	ALTMON_11	%OB	N/A
5455	alt_mon	ALTMON_12	%OB	N/A
5456	abmon	ABMON_1	%b	"Jan"
5457	abmon	ABMON_2	%b	"Feb"
5458	abmon	ABMON_3	%b	"Mar"
5459	abmon	ABMON_4	%b	"Apr"
5460	abmon	ABMON_5	%b	"May"
5461	abmon	ABMON_6	%b	"Jun"
5462	abmon	ABMON_7	%b	"Jul"
5463	abmon	ABMON_8	%b	"Aug"
5464	abmon	ABMON_9	%b	"Sep"
5465	abmon	ABMON_10	%b	"Oct"
5466	abmon	ABMON_11	%b	"Nov"
5467	abmon	ABMON_12	%b	"Dec"
5468	ab_alt_mon	ABALTMON_1	%Ob	N/A
5469	ab_alt_mon	ABALTMON_2	%Ob	N/A

5470 5471	localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
5472	<b>ab_alt_mon</b>	ABALTMON_3	%Ob	N/A
5473	<b>ab_alt_mon</b>	ABALTMON_4	%Ob	N/A
5474	<b>ab_alt_mon</b>	ABALTMON_5	%Ob	N/A
5475	<b>ab_alt_mon</b>	ABALTMON_6	%Ob	N/A
5476	<b>ab_alt_mon</b>	ABALTMON_7	%Ob	N/A
5477	<b>ab_alt_mon</b>	ABALTMON_8	%Ob	N/A
5478	<b>ab_alt_mon</b>	ABALTMON_9	%Ob	N/A
5479	<b>ab_alt_mon</b>	ABALTMON_10	%Ob	N/A
5480	<b>ab_alt_mon</b>	ABALTMON_11	%Ob	N/A
5481	<b>ab_alt_mon</b>	ABALTMON_12	%Ob	N/A
5482	<b>era</b>	ERA	%EC, %Ey, %EY	N/A
5483	<b>era_d_fmt</b>	ERA_D_FMT	%Ex	N/A
5484	<b>era_t_fmt</b>	ERA_T_FMT	%EX	N/A
5485	<b>era_d_t_fmt</b>	ERA_D_T_FMT	%Ec	N/A
5486	<b>alt_digits</b>	ALT_DIGITS	%O	N/A

5487 The entry N/A indicates the value is not available in the POSIX locale.

5488 **7.3.6 LC\_MESSAGES**

5489 The *LC\_MESSAGES* category shall define the format and values used by various utilities for  
 5490 affirmative and negative responses. This information is available through the *nl\_langinfo()*  
 5491 function.

5492 The message catalog used by the standard utilities and selected by the *catopen()* function shall be  
 5493 determined by the setting of *NLSPATH*; see [Chapter 8](#) (on page 167). The *LC\_MESSAGES*  
 5494 category can be specified as part of an *NLSPATH* substitution field.

5495 The following keywords shall be recognized as part of the locale definition file.

5496 **copy** Specify the name of an existing locale which shall be used as the definition of this  
 5497 category. If this keyword is specified, no other keyword shall be specified.

5498 **Note:** This is a *localedef* keyword, unavailable through *nl\_langinfo()*.

5499 **yesexpr** The operand consists of an extended regular expression (see [Section 9.4](#), on page  
 5500 187) that describes acceptable affirmative responses to a question expecting an  
 5501 affirmative or negative response.

5502 **noexpr** The operand consists of an extended regular expression that describes acceptable  
 5503 negative responses to a question expecting an affirmative or negative response.

5504 **7.3.6.1 LC\_MESSAGES Category in the POSIX Locale**

5505 The format and values for affirmative and negative responses of the POSIX locale follow; the  
 5506 code listing depicting the *localedef* input, the table representing the same information with the  
 5507 addition of *nl\_langinfo()* constants.

```

5508 LC_MESSAGES
5509 # This is the POSIX locale definition for
5510 # the LC_MESSAGES category.
5511 #
5512 yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5513 #
    
```

```

5514 noexpr "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5515 #
5516 END LC_MESSAGES
    
```

	localedef Keyword	langinfo Constant	POSIX Locale Value
5517			
5518	yesexpr	YESEXPR	"^[yY]"
5519	noexpr	NOEXPR	"^[nN]"

## 5520 7.4 Locale Definition Grammar

5521 The grammar and lexical conventions in this section shall together describe the syntax for the  
 5522 locale definition source. The general conventions for this style of grammar are described in XCU  
 5523 [Section 1.3](#) (on page 2461). The grammar shall take precedence over the text in this chapter.

### 5524 7.4.1 Locale Lexical Conventions

5525 The lexical conventions for the locale definition grammar are described in this section.

5526 The following tokens shall be processed (in addition to those string constants shown in the  
 5527 grammar):

- 5528 **LOC\_NAME** A string of characters representing the name of a locale.
- 5529 **CHAR** Any single character.
- 5530 **NUMBER** A decimal number, represented by one or more decimal digits.
- 5531 **COLLSYMBOL** A symbolic name, enclosed between angle brackets. The string  
 5532 cannot duplicate any charmap symbol defined in the current  
 5533 charmap (if any), or a **COLLELEMENT** symbol.
- 5534 **COLLELEMENT** A symbolic name, enclosed between angle brackets, which cannot  
 5535 duplicate either any charmap symbol or a **COLLSYMBOL** symbol.
- 5536 **CHARCLASS** A string of alphanumeric characters from the portable character set,  
 5537 the first of which is not a digit, consisting of at least one and at most  
 5538 {CHARCLASS\_NAME\_MAX} bytes, and optionally surrounded by  
 5539 double-quotes.
- 5540 **CHARSYMBOL** A symbolic name, enclosed between angle brackets, from the current  
 5541 charmap (if any).
- 5542 **OCTAL\_CHAR** One or more octal representations of the encoding of each byte in a  
 5543 single character. The octal representation consists of an escape  
 5544 character (normally a <backslash>) followed by two or more octal  
 5545 digits.
- 5546 **HEX\_CHAR** One or more hexadecimal representations of the encoding of each  
 5547 byte in a single character. The hexadecimal representation consists of  
 5548 an escape character followed by the constant *x* and two or more  
 5549 hexadecimal digits.
- 5550 **DECIMAL\_CHAR** One or more decimal representations of the encoding of each byte in  
 5551 a single character. The decimal representation consists of an escape  
 5552 character followed by a character 'd' and two or more decimal  
 5553 digits.



5554	<b>ELLIPSIS</b>	The string "...".
5555	<b>EXTENDED_REG_EXP</b>	An extended regular expression as defined in the grammar in <a href="#">Section 9.5</a> (on page 191).
5556		
5557	<b>EOL</b>	The line termination character <newline>.

## 5558 7.4.2 Locale Grammar

5559 This section presents the grammar for the locale definition.

```

5560 %token          LOC_NAME
5561 %token          CHAR
5562 %token          NUMBER
5563 %token          COLLSYMBOL COLLELEMENT
5564 %token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5565 %token          ELLIPSIS
5566 %token          EXTENDED_REG_EXP
5567 %token          EOL
5568 %start         locale_definition
5569 %%
5570 locale_definition : global_statements locale_categories
5571                  |                   locale_categories
5572                  ;
5573 global_statements : global_statements symbol_redefine
5574                  | symbol_redefine
5575                  ;
5576 symbol_redefine   : 'escape_char' CHAR EOL
5577                  | 'comment_char' CHAR EOL
5578                  ;
5579 locale_categories : locale_categories locale_category
5580                  | locale_category
5581                  ;
5582 locale_category   : lc_ctype | lc_collate | lc_messages
5583                  | lc_monetary | lc_numeric | lc_time
5584                  ;
5585 /* The following grammar rules are common to all categories */
5586 char_list         : char_list char_symbol
5587                  | char_symbol
5588                  ;
5589 char_symbol       : CHAR | CHARSYMBOL
5590                  | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5591                  ;
5592 elem_list        : elem_list char_symbol
5593                  | elem_list COLLSYMBOL
5594                  | elem_list COLLELEMENT
5595                  | char_symbol
5596                  | COLLSYMBOL

```

```

5597         | COLLELEMENT
5598         ;
5599     symb_list      : symb_list COLLSYMBOL
5600                   | COLLSYMBOL
5601                   ;
5602     locale_name    : LOC_NAME
5603                   | '"" LOC_NAME ""'
5604                   ;
5605     /* The following is the LC_CTYPE category grammar */
5606     lc_ctype       : ctype_hdr ctype_keywords      ctype_tlr
5607                   | ctype_hdr 'copy' locale_name EOL ctype_tlr
5608                   ;
5609     ctype_hdr      : 'LC_CTYPE' EOL
5610                   ;
5611     ctype_keywords : ctype_keywords ctype_keyword
5612                   | ctype_keyword
5613                   ;
5614     ctype_keyword  : charclass_keyword charclass_list EOL
5615                   | charconv_keyword charconv_list EOL
5616                   | 'charclass' charclass_namelist EOL
5617                   ;
5618     charclass_namelist : charclass_namelist ';' CHARCLASS
5619                       | CHARCLASS
5620                       ;
5621     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5622                       | 'punct' | 'xdigit' | 'space' | 'print'
5623                       | 'graph' | 'blank' | 'cntrl' | 'alnum'
5624                       | CHARCLASS
5625                       ;
5626     charclass_list  : charclass_list ';' char_symbol
5627                       | charclass_list ';' ELLIPSIS ';' char_symbol
5628                       | char_symbol
5629                       ;
5630     charconv_keyword : 'toupper'
5631                       | 'tolower'
5632                       ;
5633     charconv_list   : charconv_list ';' charconv_entry
5634                       | charconv_entry
5635                       ;
5636     charconv_entry  : '(' char_symbol ',' char_symbol ')'
5637                       ;
5638     ctype_tlr       : 'END' 'LC_CTYPE' EOL
5639                       ;
5640     /* The following is the LC_COLLATE category grammar */
5641     lc_collate      : collate_hdr collate_keywords      collate_tlr

```

```

5642         | collate_hdr 'copy' locale_name EOL collate_tlr
5643         ;
5644     collate_hdr      : 'LC_COLLATE' EOL
5645         ;
5646     collate_keywords :          order_statements
5647         | opt_statements order_statements
5648         ;
5649     opt_statements   : opt_statements collating_symbols
5650         | opt_statements collating_elements
5651         | collating_symbols
5652         | collating_elements
5653         ;
5654     collating_symbols : 'collating-symbol' COLLSYMBOL EOL
5655         ;
5656     collating_elements : 'collating-element' COLLELEMENT
5657         | 'from' "" elem_list "" EOL
5658         ;
5659     order_statements : order_start collation_order order_end
5660         ;
5661     order_start      : 'order_start' EOL
5662         | 'order_start' order_opts EOL
5663         ;
5664     order_opts       : order_opts ';' order_opt
5665         | order_opt
5666         ;
5667     order_opt        : order_opt ',' opt_word
5668         | opt_word
5669         ;
5670     opt_word         : 'forward' | 'backward' | 'position'
5671         ;
5672     collation_order   : collation_order collation_entry
5673         | collation_entry
5674         ;
5675     collation_entry   : COLLSYMBOL EOL
5676         | collation_element weight_list EOL
5677         | collation_element          EOL
5678         ;
5679     collation_element : char_symbol
5680         | COLLELEMENT
5681         | ELLIPSIS
5682         | 'UNDEFINED'
5683         ;
5684     weight_list      : weight_list ';' weight_symbol
5685         | weight_list ';'
5686         | weight_symbol
5687         ;

```

```

5688     weight_symbol      : /* empty */
5689                          | char_symbol
5690                          | COLLSYMBOL
5691                          | ''' elem_list '''
5692                          | ''' symb_list '''
5693                          | ELLIPSIS
5694                          | 'IGNORE'
5695                          ;
5696     order_end            : 'order_end' EOL
5697                          ;
5698     collate_tlr          : 'END' 'LC_COLLATE' EOL
5699                          ;
5700     /* The following is the LC_MESSAGES category grammar */
5701     lc_messages          : messages_hdr messages_keywords      messages_tlr
5702                          | messages_hdr 'copy' locale_name EOL messages_tlr
5703                          ;
5704     messages_hdr         : 'LC_MESSAGES' EOL
5705                          ;
5706     messages_keywords    : messages_keywords messages_keyword
5707                          | messages_keyword
5708                          ;
5709     messages_keyword     : 'yesexpr' ''' EXTENDED_REG_EXP ''' EOL
5710                          | 'noexpr'  ''' EXTENDED_REG_EXP ''' EOL
5711                          ;
5712     messages_tlr         : 'END' 'LC_MESSAGES' EOL
5713                          ;
5714     /* The following is the LC_MONETARY category grammar */
5715     lc_monetary           : monetary_hdr monetary_keywords      monetary_tlr
5716                          | monetary_hdr 'copy' locale_name EOL monetary_tlr
5717                          ;
5718     monetary_hdr         : 'LC_MONETARY' EOL
5719                          ;
5720     monetary_keywords    : monetary_keywords monetary_keyword
5721                          | monetary_keyword
5722                          ;
5723     monetary_keyword     : mon_keyword_string mon_string EOL
5724                          | mon_keyword_char NUMBER EOL
5725                          | mon_keyword_char '-1' EOL
5726                          | mon_keyword_grouping mon_group_list EOL
5727                          ;
5728     mon_keyword_string   : 'int_curr_symbol' | 'currency_symbol'
5729                          | 'mon_decimal_point' | 'mon_thousands_sep'
5730                          | 'positive_sign' | 'negative_sign'
5731                          ;
5732     mon_string           : ''' char_list '''

```

```

5733         | '""'
5734         ;
5735     mon_keyword_char : 'int_frac_digits' | 'frac_digits'
5736                     | 'p_cs_precedes' | 'p_sep_by_space'
5737                     | 'n_cs_precedes' | 'n_sep_by_space'
5738                     | 'p_sign_posn' | 'n_sign_posn'
5739                     | 'int_p_cs_precedes' | 'int_p_sep_by_space'
5740                     | 'int_n_cs_precedes' | 'int_n_sep_by_space'
5741                     | 'int_p_sign_posn' | 'int_n_sign_posn'
5742         ;
5743     mon_keyword_grouping : 'mon_grouping'
5744         ;
5745     mon_group_list : NUMBER
5746                   | mon_group_list ';' NUMBER
5747         ;
5748     monetary_tlr : 'END' 'LC_MONETARY' EOL
5749         ;
5750     /* The following is the LC_NUMERIC category grammar */
5751     lc_numeric : numeric_hdr numeric_keywords numeric_tlr
5752               | numeric_hdr 'copy' locale_name EOL numeric_tlr
5753         ;
5754     numeric_hdr : 'LC_NUMERIC' EOL
5755         ;
5756     numeric_keywords : numeric_keywords numeric_keyword
5757                     | numeric_keyword
5758         ;
5759     numeric_keyword : num_keyword_string num_string EOL
5760                    | num_keyword_grouping num_group_list EOL
5761        ;
5762     num_keyword_string : 'decimal_point'
5763                       | 'thousands_sep'
5764        ;
5765     num_string : '"' char_list '"'
5766               | '""'
5767        ;
5768     num_keyword_grouping : 'grouping'
5769        ;
5770     num_group_list : NUMBER
5771                   | num_group_list ';' NUMBER
5772        ;
5773     numeric_tlr : 'END' 'LC_NUMERIC' EOL
5774        ;
5775     /* The following is the LC_TIME category grammar */
5776     lc_time : time_hdr time_keywords time_tlr
5777            | time_hdr 'copy' locale_name EOL time_tlr

```

```
5778             ;
5779     time_hdr       : 'LC_TIME' EOL
5780             ;
5781     time_keywords   : time_keywords time_keyword
5782                     | time_keyword
5783             ;
5784     time_keyword     : time_keyword_name time_list EOL
5785                     | time_keyword_fmt time_string EOL
5786                     | time_keyword_opt time_list EOL
5787             ;
5788     time_keyword_name : 'abday' | 'day' | 'abmon' | 'mon'
5789             ;
5790     time_keyword_fmt  : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5791                     | 'am_pm' | 't_fmt_ampm'
5792             ;
5793     time_keyword_opt  : 'era' | 'era_d_fmt' | 'era_t_fmt'
5794                     | 'era_d_t_fmt' | 'alt_digits'
5795                     | 'ab_alt_mon' | 'alt_mon'
5796             ;
5797     time_list        : time_list ';' time_string
5798                     | time_string
5799             ;
5800     time_string       : '"' char_list '"'
5801             ;
5802     time_tlr         : 'END' 'LC_TIME' EOL
5803             ;
```

# Environment Variables

## 8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of POSIX.1-2024 for information on environment variable usage.

The value of an environment variable is an arbitrary sequence of bytes, except for the null byte. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name=value*; *names* shall not contain any bytes that have the encoded value of the character '='. For values to be portable across systems conforming to POSIX.1-2024, the value shall be composed of bytes that have the encoded value of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in an environment of a process has the same *name*, the consequences are undefined.

Environment variable names used by the utilities in the Shell and Utilities volume of POSIX.1-2024 consist solely of uppercase letters, digits, and the <underscore> ('\_') from the characters defined in Table 6-1 (on page 117) and do not begin with a digit. Other characters, and byte sequences that do not form valid characters, may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

**Note:** Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG\_MAX} bytes.

Other *name=value* pairs may be placed in the environment by, for example, calling any of the *setenv()*, *unsetenv()*, or *putenv()* functions, assigning a new value to the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of POSIX.1-2024.

If the application modifies the pointers to which *environ* points, the behavior of all interfaces described in the System Interfaces volume of POSIX.1-2024 is undefined.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

5845	ARFLAGS	IFS	MAILPATH	PS1
5846	CC	LANG	MAILRC	PS2
5847	CDPATH	LC_ALL	MAKEFLAGS	PS3
5848	CFLAGS	LC_COLLATE	MAKESHELL	PS4
5849	CHARSET	LC_CTYPE	MANPATH	PWD
5850	COLUMNS	LC_MESSAGES	MBOX	RANDOM
5851	DATEMSK	LC_MONETARY	MORE	SECONDS
5852	DEAD	LC_NUMERIC	MSGVERB	SHELL
5853	EDITOR	LC_TIME	NLSPATH	TERM
5854	ENV	LDFLAGS	NPROC	TERMCAP
5855	EXINIT	LEX	OLDPWD	TERMINFO
5856	FC	LFLAGS	OPTARG	TMPDIR
5857	FCEDIT	LINENO	OPTERR	TZ
5858	FFLAGS	LINES	OPTIND	USER
5859	GET	LISTER	PAGER	VISUAL
5860	GFLAGS	LOGNAME	PATH	YACC
5861	HISTFILE	LPDEST	PPID	YFLAGS
5862	HISTORY	MAIL	PRINTER	
5863	HISTSIZE	MAILCHECK	PROCLANG	
5864	HOME	MAILER	PROJECTDIR	

5865 Additionally, a subset of the above variables are manipulated by shell built-in utilities outside of  
 5866 shell assignments. If an attempt is made to mark any of the following variables as *readonly*, then  
 5867 either the *readonly* utility shall reject the attempt, or *readonly* shall succeed but the shell can still  
 5868 modify the variables outside of assignment context, or *readonly* shall succeed but use of a shell  
 5869 built-in that would otherwise modify such a variable shall fail.

5870 *LINENO*  
 5871 *OLDPWD*  
 5872 *OPTARG*  
 5873 *OPTIND*  
 5874 *PWD*

5875 Implementations may provide an implementation-defined set of additional variables which are  
 5876 manipulated by implementation-specific built-in utilities not defined in this standard. The  
 5877 *readonly* utility shall not reject marking these additional variables as *readonly*, but when marked  
 5878 *readonly*, those extension utilities shall either continue to modify the variables, or shall fail  
 5879 because the variable is *readonly*. None of the variables defined by this standard shall be in this  
 5880 implementation-defined set.

5881 If the variables in the following two sections are present in the environment during the  
 5882 execution of an application or utility, they shall be given the meaning described below. Some are  
 5883 placed into the environment by the implementation at the time the user logs in; all can be added  
 5884 or changed by the user or any ancestor of the current process. The implementation adds or  
 5885 changes environment variables named in POSIX.1-2024 only as specified in POSIX.1-2024. If  
 5886 they are defined in the application's environment, the utilities in the Shell and Utilities volume  
 5887 of POSIX.1-2024 and the functions in the System Interfaces volume of POSIX.1-2024 assume they  
 5888 have the specified meaning. Conforming applications shall not set these environment variables  
 5889 to have meanings other than as described. See *getenv()* (on page 1120) and XCU Section 2.13 (on  
 5890 page 2522) for methods of accessing these variables.

5891 Implementations may ignore some environment variables at the point of use for security  
 5892 reasons, for example in programs whose real and effective user IDs or real and effective group  
 5893 IDs were not equal at program startup. The behavior shall be as if the implementation obtains  
 5894 the values for these environment variables using *secure\_getenv()* instead of *getenv()* (see



5895 `getenv()`; they shall not be removed from the environment of affected processes and shall be  
 5896 inherited as required by this standard.

## 5897 8.2 Internationalization Variables

5898 This section describes environment variables that are relevant to the operation of  
 5899 internationalized interfaces described in POSIX.1-2024.

5900 Users may use the following environment variables to announce specific localization  
 5901 requirements to applications. Applications can retrieve this information using the `setlocale()`  
 5902 function to initialize the correct behavior of the internationalized interfaces. The descriptions of  
 5903 the internationalization environment variables describe the resulting behavior only when the  
 5904 application locale is initialized in this way. The use of the internationalization variables by  
 5905 utilities described in the Shell and Utilities volume of POSIX.1-2024 is described in the  
 5906 ENVIRONMENT VARIABLES section for those utilities in addition to the global effects  
 5907 described in this section.

5908 **LANG** This variable shall determine the locale category for native language, local  
 5909 customs, and coded character set in the absence of the `LC_ALL` and other `LC_*`  
 5910 (`LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`,  
 5911 `LC_TIME`) environment variables. This can be used by applications to determine  
 5912 the language to use for error messages and instructions, collating sequences, date  
 5913 formats, and so on.

### 5914 LANGUAGE

5915 The `LANGUAGE` environment variable shall be examined to determine the  
 5916 messages object to be used for the `gettext` family of functions or the `gettext` and  
 5917 XSI `ngettext` utilities if `NLSPATH` is not set or the evaluation of `NLSPATH` did not lead  
 5918 to a suitable messages object being found. The value of `LANGUAGE` shall be a list  
 5919 of locale names separated by a <colon> (':') character. If `LANGUAGE` is set to a  
 5920 non-empty string, each locale name shall be tried in the specified order and if a  
 5921 messages object is found, it shall be used for translation. If a locale name has the  
 5922 format `language[_territory][.codeset][@modifier]`, additional searches of locale names  
 5923 without `.codeset` (if present), without `_territory` (if present), and without `@modifier` (if  
 5924 present) may be performed; if `.codeset` is not present, additional searches of locale  
 5925 names with an added `.codeset` may be performed. If locale names contain a <slash>  
 5926 ('/') character, or consist entirely of a dot (".") or dot-dot (". .") character  
 5927 sequence, or are empty the behavior is implementation defined and they may be  
 5928 ignored for security reasons.

5929 The locale names in `LANGUAGE` shall override the locale name associated with  
 5930 the "active category" of the current locale or, in the case of functions with an `_l`  
 5931 suffix, the provided locale object, and the language-specific part of the default  
 5932 search path for messages objects, unless the locale name that would be overridden  
 5933 is C or POSIX. For the `dcgettext()`, `dcgettext_l()`, `dcngettext()`, and `dcngettext_l()`  
 5934 functions, the active category is specified by the `category` argument; for all other  
 5935 `gettext` family functions and for the `gettext` and `ngettext` utilities, the active category  
 5936 is `LC_MESSAGES`.

5937 For example, if:

- 5938 • The `LC_MESSAGES` environment variable is "de\_DE" (and `LC_ALL` is unset)
- 5939 and `setlocale(LC_ALL, "")` has been used to set the current locale

- 5940 • The *LANGUAGE* environment variable is "fr\_FR:it"
  - 5941 • Messages objects are by default searched for in **/gettextlib**
- 5942 then the following pathnames are tried in this order by *gettext* family functions that  
 5943 have neither a category argument nor an *\_l* suffix until a valid messages object is  
 5944 found:
- 5945 • **/gettextlib/fr\_FR/LC\_MESSAGES/textdomain.mo**
  - 5946 • (Optionally) **/gettextlib/fr/LC\_MESSAGES/textdomain.mo**
  - 5947 • (Optionally) the above two pathnames with added *.codeset* elements
  - 5948 • **/gettextlib/it/LC\_MESSAGES/textdomain.mo**
  - 5949 • (Optionally) the above pathname with added *.codeset* elements
  - 5950 • **/gettextlib/de\_DE/LC\_MESSAGES/textdomain.mo**
- 5951 *LC\_ALL* This variable shall determine the values for all locale categories. The value of the  
 5952 *LC\_ALL* environment variable has precedence over any of the other environment  
 5953 variables starting with *LC\_* (*LC\_COLLATE*, *LC\_CTYPE*, *LC\_MESSAGES*,  
 5954 *LC\_MONETARY*, *LC\_NUMERIC*, *LC\_TIME*) and the *LANG* environment variable.
- 5955 *LC\_COLLATE*  
 5956 This variable shall determine the locale category for character collation. It  
 5957 determines collation information for regular expressions and sorting, including  
 5958 equivalence classes and multi-character collating elements, in various utilities and  
 5959 the *strcoll()* and *strxfrm()* functions. Additional semantics of this variable, if any,  
 5960 are implementation-defined.
- 5961 *LC\_CTYPE* This variable shall determine the locale category for character handling functions,  
 5962 such as *tolower()*, *toupper()*, and *isalpha()*. This environment variable determines  
 5963 the interpretation of sequences of bytes of text data as characters (for example,  
 5964 single as opposed to multi-byte characters), the classification of characters (for  
 5965 example, alpha, digit, graph), and the behavior of character classes. Additional  
 5966 semantics of this variable, if any, are implementation-defined.
- 5967 *LC\_MESSAGES*  
 5968 This variable shall determine the locale category for processing affirmative and  
 5969 negative responses and the language and cultural conventions in which messages  
 5970 should be written. It also affects the behavior of the *catopen()* function in  
 5971 determining the message catalog. Additional semantics of this variable, if any, are  
 5972 implementation-defined. The language and cultural conventions of diagnostic and  
 5973 informative messages whose format is unspecified by POSIX.1-2024 should be  
 5974 affected by the setting of *LC\_MESSAGES*.
- 5975 *LC\_MONETARY*  
 5976 This variable shall determine the locale category for monetary-related numeric  
 5977 formatting information. Additional semantics of this variable, if any, are  
 5978 implementation-defined.
- 5979 *LC\_NUMERIC*  
 5980 This variable shall determine the locale category for numeric formatting (for  
 5981 example, thousands separator and radix character) information in various utilities  
 5982 as well as the formatted I/O operations in *printf()* and *scanf()* and the string  
 5983 conversion functions in *strtod()*. Additional semantics of this variable, if any, are  
 5984 implementation-defined.

5985		<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in <i>strftime()</i> . Additional semantics of this variable, if any, are implementation-defined.
5986			
5987			
5988	XSI	<i>NLSPATH</i>	This variable shall contain a sequence of templates to be used by <i>catopen()</i> when attempting to locate message catalogs, and by the <i>gettext</i> family of functions when locating messages objects. Each template consists of an optional prefix, one or more conversion specifications, and an optional suffix.
5989			
5990			
5991			
5992			The conversion specification descriptions below refer to a "currently active text domain". The currently active text domain is, in decreasing order of precedence:
5993			
5994			• The <i>domain</i> parameter of the <i>gettext</i> family of functions or the <i>gettext</i> and <i>ngettext</i> utilities
5995			
5996			• The text domain bound by the last call to <i>textdomain()</i> when using a <i>gettext</i> family function, or the <i>TEXTDOMAIN</i> environment variable when using the <i>gettext</i> and <i>ngettext</i> utilities
5997			
5998			
5999			• The default text domain
6000			Conversion specifications consist of a '%' symbol, followed by a single-letter keyword. The following conversion specifications are currently defined:
6001			
6002		%N	The value of the <i>name</i> parameter passed to <i>catopen()</i> or the currently active text domain of the <i>gettext</i> family of functions and the <i>gettext</i> and <i>ngettext</i> utilities (see above).
6003			
6004			
6005		%L	The locale name given by the value of the active category (see <i>LANGUAGE</i> above) in either the current locale or, in the case of functions with an <i>_l</i> suffix, the provided locale object.
6006			
6007			
6008		%l	The <i>language</i> element of the locale name that would result from a %L conversion.
6009			
6010		%t	The <i>territory</i> element of the locale name that would result from a %L conversion.
6011			
6012		%c	The <i>codeset</i> element of the locale name that would result from a %L conversion.
6013		%%	A single '%' character.
6014			An empty string shall be substituted if the specified value is not currently defined. The separators <underscore> ('_') and <period> ('.') shall not be included in the %t and %c conversion specifications.
6015			
6016			
6017			Templates defined in <i>NLSPATH</i> are separated by <colon> characters (':'). A leading, trailing, or two adjacent <colon> characters ("::") shall be equivalent to specifying %N.
6018			
6019			
6020			Since <colon> is a separator in this context, directory names that might be used in <i>NLSPATH</i> should not include a <colon> character.
6021			
6022			Example 1, for an application that uses <i>catopen()</i> but does not use the <i>gettext</i> family of functions:
6023			
6024			<code>NLSPATH="/system/nlslib/%N.cat"</code>
6025			indicates that <i>catopen()</i> should look for all message catalogs in the directory <code>/system/nlslib</code> , where the catalog name should be constructed from the <i>name</i> argument (replacing %N) passed to <i>catopen()</i> , with the suffix <code>.cat</code> .
6026			
6027			

6028 Example 2, for an application that uses the *gettext* family of functions but does not  
6029 use *catopen()*:

```
6030 NLSPATH="/usr/lib/locale/fr/LC_MESSAGES/%N.mo"
```

6031 indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities)  
6032 should look for all messages objects in the directory  
6033 */usr/lib/locale/fr/LC\_MESSAGES*, where the messages object's name should be  
6034 constructed from the currently active text domain (replacing %N), with the suffix  
6035 **.mo**.

6036 Example 3, for an application that uses *catopen()* but does not use the *gettext* family  
6037 of functions:

```
6038 NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

6039 indicates that *catopen()* should look for the requested message catalog in *name*,  
6040 *name.cat*, and */nlslib/localename/name.cat*, where *localename* is the locale name given  
6041 by the value of the *LC\_MESSAGES* category of the current locale.

6042 Example 4, for an application that uses the *gettext* family of functions but does not  
6043 use *catopen()*:

```
6044 NLSPATH="/usr/lib/locale/%L/%N.mo:/usr/lib/locale/fr/%N.mo"
```

6045 indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities)  
6046 should look for all messages objects first in  
6047 */usr/lib/locale/localename/textdomain.mo*, and if not found there, then try in  
6048 */usr/lib/locale/fr/textdomain.mo*, where *localename* is the locale name given by the  
6049 value of the active category in the current locale or provided locale object.

6050 Example 5, for an application that uses *catopen()* and the *gettext* family of  
6051 functions:

```
6052 NLSPATH="/usr/lib/locale/%L/%N.mo:/system/nlslib/%L/%N.cat"
```

6053 indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities)  
6054 should look for all messages objects in */usr/lib/locale/localename/textdomain.mo*,  
6055 where *localename* is the locale name given by the value of the active category in the  
6056 current locale or provided locale object. Also, *catopen()* should look for all message  
6057 catalogs in the directory */system/nlslib/localename/name.cat*, (assuming that  
6058 */usr/lib/locale/localename/name.mo* is not a message catalog). In this scenario,  
6059 *catopen()* ignores all files that are not valid message catalogs while traversing  
6060 *NLSPATH*. Furthermore, the *gettext* family of functions and the *gettext* and *ngettext*  
6061 utilities ignore all files that are not valid messages objects found while traversing  
6062 *NLSPATH*.

6063 Users should not set the *NLSPATH* variable unless they have a specific reason to  
6064 override the default system path. Setting *NLSPATH* to override the default system  
6065 path may produce undefined results in the standard utilities other than *gettext* and  
6066 *ngettext*, and in applications with appropriate privileges.

6067 Specifying a relative pathname in the *NLSPATH* environment variable should be  
6068 avoided without a specific reason, including the use of a leading, trailing, or two  
6069 adjacent <colon> characters, since it may result in messages objects being searched  
6070 for in a directory relative to the current working directory of the calling process; if  
6071 the process calls the *chdir()* function, the directory searched for may also be  
6072 changed.

6073 *TEXTDOMAIN*  
 6074 Specify the text domain name that the *gettext* and *ngettext* utilities use during the  
 6075 search for messages objects. This is identical to the messages object filename  
 6076 without the **.mo** suffix.

6077 *TEXTDOMAINDIR*  
 6078 Specify the pathname to the root directory of the messages object hierarchy the  
 6079 *gettext* and *ngettext* utilities use during the search for messages objects. If present, it  
 6080 XSI shall replace the default root directory pathname. *NLSPATH* has precedence over  
 6081 *TEXTDOMAINDIR*.

6082 The environment variables *LANG*, *LC\_ALL*, *LC\_COLLATE*, *LC\_CTYPE*, *LC\_MESSAGES*,  
 6083 *LC\_MONETARY*, *LC\_NUMERIC*, *LC\_TIME*, and *NLSPATH* provide for the support of  
 6084 internationalized applications. The standard utilities shall make use of these environment  
 6085 variables as described in this section and the individual ENVIRONMENT VARIABLES sections  
 6086 for the utilities. See [Section 7.1](#) (on page 127) for the consequences of setting these variables to  
 6087 locales with different character sets.

6088 The values of locale categories shall be determined by a precedence order; the first condition met  
 6089 below determines the value:

- 6090 1. If the *LC\_ALL* environment variable is defined and is not null, the value of *LC\_ALL* shall  
 6091 be used.
- 6092 2. If the *LC\_\** environment variable (*LC\_COLLATE*, *LC\_CTYPE*, *LC\_MESSAGES*,  
 6093 *LC\_MONETARY*, *LC\_NUMERIC*, *LC\_TIME*) is defined and is not null, the value of the  
 6094 environment variable shall be used to initialize the category that corresponds to the  
 6095 environment variable.
- 6096 3. If the *LANG* environment variable is defined and is not null, the value of the *LANG*  
 6097 environment variable shall be used.
- 6098 4. Otherwise, the implementation-defined default locale shall be used.

6099 If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities  
 6100 behave in accordance with the rules in [Section 7.2](#) (on page 128) for the associated category.

6101 If the locale value begins with a <slash>, it shall be interpreted as the pathname of a file that was  
 6102 created in the output format used by the *localedef* utility; see OUTPUT FILES under *localedef*.  
 6103 Referencing such a pathname shall result in that locale being used for the indicated category.

6104 XSI If the locale value has the form:

```
6105 language[_territory] [.codeset]
```

6106 it refers to an implementation-provided locale, where settings of language, territory, and codeset  
 6107 are implementation-defined.

6108 *LC\_COLLATE*, *LC\_CTYPE*, *LC\_MESSAGES*, *LC\_MONETARY*, *LC\_NUMERIC*, and *LC\_TIME* are  
 6109 defined to accept an additional field *@modifier*, which allows the user to select a specific instance  
 6110 of localization data within a single category (for example, for selecting the dictionary as opposed  
 6111 to the character ordering of data). The syntax for these environment variables is thus defined as:

```
6112 [language[_territory] [.codeset] [@modifier]]
```

6113 For example, if a user wanted to interact with the system in French, but required to sort German  
 6114 text files, *LANG* and *LC\_COLLATE* could be defined as:

```
6115 LANG=Fr_FR
```

```
6116 LC_COLLATE=De_DE
```

6117 This could be extended to select dictionary collation (say) by use of the `@modifier` field; for  
 6118 example:  
 6119 `LC_COLLATE=De_DE@dict`

6120 An implementation may support other formats.

6121 If the locale value is not recognized by the implementation, the behavior is unspecified.

6122 These environment variables are used by the `newlocale()` and `setlocale()` functions, and by the  
 6123 standard utilities.

6124 Additional criteria for determining a valid locale name are implementation-defined.

### 6125 8.3 Other Environment Variables

6126 **COLUMNS** This variable shall represent a decimal integer >0 used to indicate the user's  
 6127 preferred width in column positions for the terminal screen or window; see  
 6128 [Section 3.75](#) (on page 42). If this variable is unset or null, the number of  
 6129 columns shall be set according to the terminal window size (see XSH  
 6130 `tcgetwinsize()`); if the terminal window size cannot be obtained, the  
 6131 implementation determines the number of columns, appropriate for the  
 6132 terminal or window, in an unspecified manner. When **COLUMNS** is set, the  
 6133 number of columns in the terminal window size and any terminal-width  
 6134 information implied by **TERM** are overridden. Users and conforming  
 6135 applications should not set **COLUMNS** unless they wish to override the  
 6136 system selection and produce output unrelated to the terminal characteristics.

6137 Users should not need to set this variable in the environment unless there is a  
 6138 specific reason to override the implementation's default behavior, such as to  
 6139 display data in an area arbitrarily smaller than the terminal or window.

6140 XSI **DATEMSK** Indicates the pathname of the template file used by `getdate()`.

6141 **HOME** The system shall initialize this variable at the time of login to be a pathname of  
 6142 the user's home directory. See [<pwd.h>](#).

6143 **LINES** This variable shall represent a decimal integer >0 used to indicate the user's  
 6144 preferred number of lines on a page or the vertical screen or window size in  
 6145 lines. A line in this case is a vertical measure large enough to hold the tallest  
 6146 character in the character set being displayed. If this variable is unset or null,  
 6147 the number of lines shall be set either to the number of rows in the terminal  
 6148 window size (see XSH `tcgetwinsize()`) or to a smaller number if appropriate for  
 6149 the terminal or window (for example, if the terminal baud rate is low); if the  
 6150 terminal window size cannot be obtained, the implementation determines the  
 6151 number of lines, appropriate for the terminal or window, in an unspecified  
 6152 manner. When **LINES** is set, the number of rows in the terminal window size  
 6153 and any terminal-height information implied by **TERM** are overridden. Users  
 6154 and conforming applications should not set **LINES** unless they wish to  
 6155 override the system selection and produce output unrelated to the terminal  
 6156 characteristics.

6157 Users should not need to set this variable in the environment unless there is a  
 6158 specific reason to override the implementation's default behavior, such as to  
 6159 display data in an area arbitrarily smaller than the terminal or window.

6160		<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's login name. See <a href="#">&lt;pwd.h&gt;</a> . For a value of <i>LOGNAME</i> to be portable across implementations of POSIX.1-2024, the value should be composed of characters from the portable filename character set.
6161			
6162			
6163			
6164	XSI	<i>MSGVERB</i>	Describes which message components shall be used in writing messages by <i>fntmsg()</i> .
6165			
6166		<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain functions and utilities apply in searching for an executable file. The prefixes shall be separated by a <colon> (' : '). If the pathname being sought contains no <slash> (' / ') characters, and hence is a filename, the list shall be searched from beginning to end, applying the filename to each prefix and attempting to resolve the resulting pathname (see <a href="#">Section 4.16</a> , on page 105), until an executable file with appropriate execution permissions is found. When a non-zero-length prefix is applied to this filename, a <slash> shall be inserted between the prefix and the filename if the prefix did not end in <slash>. A zero-length prefix is a legacy feature that indicates the current working directory. It appears as two adjacent <colon> characters (" : : "), as an initial <colon> preceding the rest of the list, or as a trailing <colon> following the rest of the list. A strictly conforming application shall use an actual pathname (such as .) to represent the current working directory in <i>PATH</i> . If the pathname being sought contains any <slash> characters, the search through the path prefixes shall not be performed and the pathname shall be resolved as described in <a href="#">Section 4.16</a> (on page 105). If <i>PATH</i> is unset or is set to null, or if a path prefix in <i>PATH</i> contains a <percent-sign> character (' % '), the path search is implementation-defined.
6167			
6168			
6169			
6170			
6171			
6172			
6173			
6174			
6175			
6176			
6177			
6178			
6179			
6180			
6181			
6182			
6183			
6184			
6185			Since <colon> is a separator in this context, directory names that might be used in <i>PATH</i> should not include a <colon> character. Since <percent-sign> may have an implementation-defined meaning when searching for built-in utilities, directory names in <i>PATH</i> to be used to search for non-built-in utilities should not contain a <percent-sign> character.
6186			
6187			
6188			
6189			
6190		<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. It shall not contain any components that are dot or dot-dot. The value is set by the <i>cd</i> utility, and by the <i>sh</i> utility during initialization.
6191			
6192			
6193		<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command language interpreter. If this interpreter does not conform to the Shell Command Language in XCU <a href="#">Chapter 2</a> (on page 2472), utilities may behave differently from those described in POSIX.1-2024.
6194			
6195			
6196			
6197		<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for programs that need a place to create temporary files.
6198			
6199		<i>TERM</i>	This variable shall represent the terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
6200			
6201			
6202			
6203		<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>localtime()</i> , <i>localtime_r()</i> , <i>strftime()</i> , and <i>mktime()</i> functions, and by various utilities, to override the default timezone. The application shall ensure that the value of <i>TZ</i> is in one of the three formats (spaces inserted for clarity):
6204			
6205			
6206			
6207			

6208                    : *characters*

6209                    or:

6210                    *std offset dst offset, rule*

6211                    or:

6212                    A format specifying a geographical timezone or a special timezone.

6213                    If *TZ* is of the first format (that is, if the first character is a <colon>), the

6214                    characters following the <colon> are handled in an implementation-defined

6215                    manner.

6216                    The expanded form of the second format (without the inserted spaces) is as

6217                    follows:

6218                    *stdoffset [dst [offset] [, start [/time], end [/time]]]*

6219                    Where:

6220                    *std* and *dst*   Indicate no less than three, nor more than {TZNAME\_MAX},

6221                    bytes that are the designation for the standard (*std*) or the

6222                    Daylight Saving (*dst*) timezone. Only *std* is required; if *dst* is

6223                    missing, then Daylight Saving Time does not apply in this locale.

6224                    **Note:**     The usage of the terms “Standard Time” and “Daylight

6225                    Saving Time” is not necessarily related to any legislated

6226                    timezone.

6227                    Each of these fields may occur in either of two formats quoted or

6228                    unquoted:

6229                    — In the quoted form, the first character shall be the <less-

6230                    than-sign> ('<') character and the last character shall be

6231                    the <greater-than-sign> ('>') character. All characters

6232                    between these quoting characters shall be alphanumeric

6233                    characters from the portable character set in the current

6234                    locale, the <plus-sign> ('+') character, or the <hyphen-

6235                    minus> ('-') character. The *std* and *dst* fields in this case

6236                    shall not include the quoting characters and the quoting

6237                    characters do not contribute to the three byte minimum

6238                    length and {TZNAME\_MAX} maximum length.

6239                    — In the unquoted form, all characters in these fields shall be

6240                    alphabetic characters from the portable character set in the

6241                    current locale.

6242                    The interpretation of *std* and, if present, *dst* is unspecified if the

6243                    field is less than three bytes or more than {TZNAME\_MAX}

6244                    bytes, or if it contains characters other than those specified.

6245                    *offset*       Indicates the value added to the local time to arrive at

6246                    Coordinated Universal Time. The *offset* has the form:

6247                    *hh[:mm[:ss]]*

6248                    The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)

6249                    shall be required and may be a single digit. The *offset* following

6250                    *std* shall be required. If no *offset* follows *dst*, Daylight Saving

6251                    Time is assumed to be one hour ahead of standard time. One or

6252                    more digits may be used; the value is always interpreted as a



6253		decimal number. The hour shall be between zero and 24, and the
6254		minutes (and seconds)—if present—between zero and 59. The
6255		result of using values outside of this range is unspecified. If
6256		preceded by a '-', the timezone shall be east of the Prime
6257		Meridian; otherwise, it shall be west (which may be indicated by
6258		an optional preceding '+').
6259	<i>rule</i>	Indicates when to change from standard time to Daylight Saving
6260		Time, and when to change back. The <i>rule</i> has the form:
6261		<i>date</i> [/ <i>time</i> ], <i>date</i> [/ <i>time</i> ]
6262		where the first <i>date</i> describes when the change from standard
6263		time to Daylight Saving Time occurs and the second <i>date</i>
6264		describes when it ends; if the second <i>date</i> is specified as earlier in
6265		the year than the first, then the year begins and ends in Daylight
6266		Saving Time. Each <i>time</i> field describes when, in current local
6267		time, the change to the other time is made.
6268		The format of <i>date</i> is one of the following:
6269	<i>Jn</i>	The Julian day <i>n</i> ( $1 \leq n \leq 365$ ). Leap days shall not be
6270		counted. That is, in all years—including leap years—
6271		February 28 is day 59 and March 1 is day 60. It is
6272		impossible to refer explicitly to the occasional February
6273		29.
6274	<i>n</i>	The zero-based Julian day ( $0 \leq n \leq 365$ ). Leap days shall
6275		be counted, and it is possible to refer to February 29.
6276	<i>Mm.n.d</i>	The <i>d</i> 'th day ( $0 \leq d \leq 6$ ) of week <i>n</i> of month <i>m</i> of the
6277		year ( $1 \leq n \leq 5, 1 \leq m \leq 12$ , where week 5 means "the last
6278		<i>d</i> day in month <i>m</i> " which may occur in either the fourth
6279		or the fifth week). Week 1 is the first week in which the
6280		<i>d</i> 'th day occurs. Day zero is Sunday.
6281		The <i>time</i> has the same format as <i>offset</i> except that the hour can
6282		range from zero to 167. If preceded by a '-', the time shall
6283		count backwards before midnight. For example, "47:30"
6284		stands for 23:30 the next day, and "-3:30" stands for 20:30 the
6285		previous day. The default, if <i>time</i> is not given, shall be 02:00:00.
6286		Daylight Saving Time is in effect all year if it starts January 1 at 00:00 and ends
6287		December 31 at 24:00 plus the difference between Daylight Saving Time and
6288		standard time, leaving no room for standard time in the calendar. For
6289		example, TZ='EST5EDT,0/0,J365/25' represents a time zone that
6290		observes Daylight Saving Time all year, being 4 hours west of UTC with
6291		abbreviation "EDT".
6292		If the <i>dst</i> field is specified and the <i>rule</i> field is not, it is implementation-defined
6293		when the changes to and from Daylight Saving Time occur.
6294		If <i>TZ</i> is of the third format (that is, if the first character is not a <colon> and
6295		the value does not match the syntax for the second format), the value indicates
6296		either a geographical timezone or a special timezone from an implementation-
6297		defined timezone database. Typically these take the form
6298		<i>Area/Location</i>

6299 as in the IANA timezone database. Examples of geographical timezones that  
6300 may be supported include Africa/Cairo, America/New\_York,  
6301 America/Indiana/Indianapolis, Asia/Tokyo, and Europe/London.  
6302 The data for each geographical timezone shall include:

- 6303 • The offset from Coordinated Universal Time of the timezone's standard  
6304 time.
- 6305 • If Daylight Saving Time (DST) is, or has historically been, observed: a  
6306 method to discover the dates and times of transitions to and from DST  
6307 and the offset from Coordinated Universal Time during periods when  
6308 DST was, is, or is predicted to be, in effect.
- 6309 • The timezone names for standard time (*std*) and, if observed, for DST  
6310 (*dst*) to be used by *tzset()*. These shall each contain no more than  
6311 {TZNAME\_MAX} bytes.

6312 If there are any historical variations, or known future variations, of the above  
6313 data for a geographical timezone, these variations shall be included in the  
6314 database, except that historical variations from before the Epoch need not be  
6315 included.

6316 If the database incorporates an external database such as the one maintained  
6317 by IANA, the implementation shall provide an implementation-defined  
6318 method to allow the database to be updated, for example the method specified  
6319 by RFC 6557.

6320

6321

# Regular Expressions

6322 Regular Expressions (REs) provide a mechanism to select specific strings from a set of character  
6323 strings.

6324 Regular expressions are a context-independent syntax that can represent a wide variety of  
6325 character sets and character set orderings, where these character sets are interpreted according  
6326 to the current locale. While many regular expressions can be interpreted differently depending  
6327 on the current locale, many features, such as character class expressions, provide for contextual  
6328 invariance across locales.

6329 The Basic Regular Expression (BRE) notation and construction rules in Section 9.3 (on page 181)  
6330 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the  
6331 Extended Regular Expressions (ERE) described in Section 9.4 (on page 187); any exceptions for  
6332 both cases are noted in the descriptions of the specific utilities using regular expressions. Both  
6333 BREs and EREs are supported by the Regular Expression Matching interface in the System  
6334 Interfaces volume of POSIX.1-2024 under *regcomp()*, *regex()*, and related functions.

## 6335 9.1 Regular Expression Definitions

6336 For the purposes of this section, the following definitions shall apply:

### 6337 entire regular expression

6338 The concatenated set of one or more BREs or EREs that make up the pattern specified for  
6339 string selection.

### 6340 escape sequence

6341 The escape character followed by any single character, which is thereby “escaped”. The  
6342 escape character is a <backslash> that is neither in a bracket expression nor itself escaped.

### 6343 leftmost

6344 The characters closest to the beginning of the string.

### 6345 matched

6346 A sequence of zero or more characters shall be said to be matched by a BRE or ERE when  
6347 the characters in the sequence correspond to a sequence of characters defined by the  
6348 pattern.

6349 Matching shall be based on the bit pattern used for encoding the character, not on the  
6350 graphic representation of the character. This means that if a character set contains two or  
6351 more encodings for a graphic symbol, or if the strings searched contain text encoded in  
6352 more than one codeset, no attempt is made to search for any other representation of the  
6353 encoded symbol. If that is required, the user can specify equivalence classes containing all  
6354 variations of the desired graphic symbol.

6355 The search for a matching sequence starts at the beginning of a string and stops when the  
6356 first sequence matching the expression is found, where “first” is defined to mean “begins  
6357 earliest in the string”. If the pattern permits a variable number of matching characters and  
6358 thus there is more than one such sequence starting at that point, the longest such sequence  
6359 is matched. For example, the BRE "bb\*" matches the second to fourth characters of the

6360 string "abbbc", and the ERE "(wee|week)(knights|night)" matches all ten  
6361 characters of the string "weeknights".

6362 Consistent with the whole match being the longest of the leftmost matches, each subpattern,  
6363 from left to right, shall match the longest possible string. For this purpose, a null string shall  
6364 be considered to be longer than no match at all. For example, matching the BRE  
6365 "\(.\*\)" against "abcdef", the subexpression "(\\1)" is "abcdef", and matching  
6366 the BRE "\(a\*\)" against "bc", the subexpression "(\\1)" is the null string. However,  
6367 matching the ERE "(.\*?)" against "abcdef", the subpattern "(.\*?)" matches the  
6368 empty string, since that is the longest possible match for the ERE ".\*?".

6369 When a multi-character collating element in a bracket expression (see [Section 9.3.5](#), on page  
6370 182) is involved, the longest sequence shall be measured in characters consumed from the  
6371 string to be matched; that is, the collating element counts not as one element, but as the  
6372 number of characters it matches.

### 6373 **BRE (ERE) matching a single character**

6374 A BRE or ERE that shall match either a single character or a single collating element.

6375 Only a BRE or ERE of this type that includes a bracket expression (see [Section 9.3.5](#), on page  
6376 182) can match a collating element.

### 6377 **BRE (ERE) matching multiple characters**

6378 A BRE or ERE that shall match a concatenation of single characters or collating elements.

6379 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE  
6380 (ERE) special characters.

### 6381 **invalid**

6382 This section uses the term "invalid" for certain constructs or conditions. Invalid REs shall  
6383 cause the utility or function using the RE to generate an error condition. When invalid is not  
6384 used, violations of the specified syntax or semantics for REs produce undefined results: this  
6385 may entail an error, enabling an extended syntax for that RE, or using the construct in error  
6386 as literal characters to be matched. For example, the BRE construct "{1,2,3}" does not  
6387 comply with the grammar. A conforming application cannot rely on it producing an error  
6388 nor matching the literal characters "{1,2,3}".

## 6389 **9.2 Regular Expression General Requirements**

6390 The requirements in this section shall apply to both basic and extended regular expressions.

6391 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)  
6392 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter  
6393 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that  
6394 is, zero or more characters followed by a <newline>.

6395 In the functions processing regular expressions described in System Interfaces volume of  
6396 POSIX.1-2024, the <newline> is regarded as an ordinary character and both a <period> and a  
6397 non-matching list can match one. The Shell and Utilities volume of POSIX.1-2024 specifies  
6398 within the individual descriptions of those standard utilities employing regular expressions  
6399 whether they permit matching of <newline> characters; if not stated otherwise, the use of literal  
6400 <newline> characters or any escape sequence equivalent in either patterns or matched text  
6401 produces undefined results. Those utilities (like *grep*) that do not allow <newline> characters to  
6402 match are responsible for eliminating any <newline> from strings before matching against the  
6403 RE. The *regcomp()* function in the System Interfaces volume of POSIX.1-2024, however, can  
6404 provide support for such processing without violating the rules of this section.

6405 The interfaces specified in POSIX.1-2024 do not permit the inclusion of a NUL character in an RE  
 6406 or in the string to be matched. If during the operation of a standard utility a NUL is included in  
 6407 the text designated to be matched, that NUL may designate the end of the text string for the  
 6408 purposes of matching.

6409 Some standard utilities and functions support case-insensitive regular expression matching.  
 6410 When this type of matching is in effect, the matching process shall be modified as described in  
 6411 [Section 4.1](#) (on page 95).

6412 The implementation shall support any regular expression that does not exceed 256 bytes in  
 6413 length.

## 6414 9.3 Basic Regular Expressions

### 6415 9.3.1 BREs Matching a Single Character or Collating Element

6416 When not inside a bracket expression, the following shall match a single character:

- 6417 • a BRE ordinary character
- 6418 • a BRE special character or ']' preceded by an unescaped <backslash>
- 6419 • a <period>

6420 A bracket expression shall match a single character or a single collating element.

### 6421 9.3.2 BRE Ordinary Characters

6422 An ordinary character is a BRE that matches itself: any character in the supported character set,  
 6423 except for the BRE special characters listed in [Section 9.3.3](#) (on page 182).

6424 When not inside a bracket expression, the interpretation of an ordinary character preceded by an  
 6425 unescaped <backslash> is undefined, except for:

- 6426 • The characters ')', '(', '{', and '}'
- 6427 • The digits 1 to 9 inclusive (see [Section 9.3.6](#), on page 185)
- 6428 • The ']' character; "\]" shall match a ']' character
- 6429 • The '?', '+', and '|' characters; it is implementation-defined whether "\?", "\+", and  
 6430 "\|" each match the literal character '?', '+', or '|', respectively, or behave as  
 6431 described for the ERE special characters '?', '+', and '|', respectively (see [Section 9.4.3](#),  
 6432 on page 188).

6433 **Note:** A future version of this standard may require "\?", "\+", and "\|" to behave as  
 6434 described for the ERE special characters '?', '+', and '|', respectively.

### 6435 9.3.3 BRE Special Characters

6436 A BRE special character has special properties in certain contexts. Outside those contexts, or  
6437 when preceded by an unescaped <backslash>, such a character is a BRE that matches the special  
6438 character itself. The BRE special characters and the contexts in which they have their special  
6439 meaning are as follows:

- 6440 . [ \ The <period>, <left-square-bracket>, and <backslash> shall be special except when  
6441 used in a bracket expression (see [Section 9.3.5](#)). An expression containing a '[' that is  
6442 unescaped and is not part of a bracket expression produces undefined results.
- 6443 \* The <asterisk> shall be special except when used:
  - 6444 — In a bracket expression
  - 6445 — As the first character of an entire BRE (after an initial '^', if any)
  - 6446 — Immediately following a "\|" escape sequence (after an initial '^', if any), if the  
6447 implementation does not match the escape sequence "\|" to the literal character  
6448 '|'.
    - 6449 — As the first character of a subexpression (after an initial '^', if any); see [Section](#)  
6450 [9.3.6](#) (on page 185)
- 6451 ^ The <circumflex> shall be special when used as an anchor (see [Section 9.3.8](#), on page  
6452 186). The <circumflex> shall signify a non-matching list expression when it occurs first  
6453 in a list, immediately following a <left-square-bracket> (see [Section 9.3.5](#)).
- 6454 \$ The <dollar-sign> shall be special when used as an anchor.

### 6455 9.3.4 Periods in BREs

6456 When not inside a bracket expression, a <period> ('.') is a BRE that shall match any character  
6457 in the supported character set except NUL.

### 6458 9.3.5 RE Bracket Expression

6459 A bracket expression (an expression enclosed in square brackets, "[ ]") is an RE that shall  
6460 match a specific set of single characters, and may match a specific set of multi-character collating  
6461 elements, based on the non-empty set of list expressions contained in the bracket expression.

6462 The following rules and definitions apply to bracket expressions:

- 6463 1. A bracket expression is either a matching list expression or a non-matching list  
6464 expression. It consists of one or more expressions: ordinary characters, collating elements,  
6465 collating symbols, equivalence classes, character classes, or range expressions. The <right-  
6466 square-bracket> (']') shall lose its special meaning and represent itself in a bracket  
6467 expression if it occurs first in the list (after an initial <circumflex> ('^'), if any).  
6468 Otherwise, it shall terminate the bracket expression, unless it appears in a collating  
6469 symbol (such as "[.].]") or is the ending <right-square-bracket> for a collating symbol,  
6470 equivalence class, or character class. When the bracket expression appears within a BRE,  
6471 the special characters '.', '\*', '[', and '\\ ' (<period>, <asterisk>, <left-square-  
6472 bracket>, and <backslash>, respectively) shall lose their special meaning within the  
6473 bracket expression. When the bracket expression appears within an ERE, the special  
6474 characters '.', '(', '\*', '+', '?', '{', '|', '\$', '[', and '\\ ' (<period>, <left-  
6475 parenthesis>, <asterisk>, <plus-sign>, <question-mark>, <left-brace>, <vertical-line>,  
6476 <dollar-sign>, <left-square-bracket>, and <backslash>, respectively) shall lose their

6477 special meaning within the bracket expression; <circumflex> ('^') shall lose its special  
 6478 meaning as an anchor. When the bracket expression appears within a shell pattern (see  
 6479 XCU [Section 2.14](#), on page 2523), the special characters '?', '\*', and '[' (<question-  
 6480 mark>, <asterisk>, and <left-square-bracket>, respectively) shall lose their special  
 6481 meaning within the bracket expression; whether or not <backslash> ('\') loses its  
 6482 special meaning as a pattern matching character is described in XCU [Section 2.14.1](#) (on  
 6483 page 2523), but in contexts where a shell-quoting <backslash> can be used it shall retain  
 6484 its special meaning (see XCU [Section 2.2](#), on page 2472). For example:

```
6485 $ ls
6486 ! $ - \ a b c
6487 $ echo [a\-c]
6488 - a c
6489 $ echo [\!a]
6490 ! a
6491 $ echo ["!\$a-c"]
6492 ! $ - a c
6493 $ echo [!"\$a-c"]
6494 ! \ b
6495 $ echo [!\]\]
6496 ! $ - a b c
```

6497 The character sequences "[.", "[=", and "[:" (<left-square-bracket> followed by a  
 6498 <period>, <equals-sign>, or <colon>) shall be special inside a bracket expression and are  
 6499 used to delimit collating symbols, equivalence class expressions, and character class  
 6500 expressions. These symbols shall be followed by a valid expression and the matching  
 6501 terminating sequence ".]", "=]", or ":]", as described in the following items.

6502 2. A matching list expression specifies a list that shall match any single character that is  
 6503 matched by one of the expressions represented in the list. The first character in the list  
 6504 cannot be the <circumflex>. An ordinary character in the list shall only match that  
 6505 character; for example, "[abc]" is an RE that only matches one of the characters 'a',  
 6506 'b', or 'c'.

6507 It is unspecified whether a matching list expression matches a multi-character collating  
 6508 element that is matched by one of the expressions.

6509 3. A non-matching list expression begins with a <circumflex> ('^'), and the matching  
 6510 behavior shall be the logical inverse of the corresponding matching list expression (the  
 6511 same bracket expression but without the leading <circumflex>). For example, since the  
 6512 RE "[abc]" only matches 'a', 'b', or 'c', it follows that "[^abc]" is an RE that  
 6513 matches any character except 'a', 'b', or 'c'. It is unspecified whether a non-matching  
 6514 list expression matches a multi-character collating element that is not matched by any of  
 6515 the expressions. The <circumflex> shall have this special meaning only when it occurs  
 6516 first in the list, immediately following the <left-square-bracket>.

6517 4. A collating symbol is a collating element enclosed within bracket-period ("[" and  
 6518 ".]") delimiters. Collating elements are defined as described in [Section 7.3.2.4](#) (on page  
 6519 142). Conforming applications shall represent multi-character collating elements as  
 6520 collating symbols when it is necessary to distinguish them from a list of the individual  
 6521 characters that make up the multi-character collating element. For example, if the string  
 6522 "ch" is a collating element defined using the line:

```
6523 collating-element <ch-digraph> from "<c><h>"
```

6524 in the locale definition, the expression "[[.ch.]]" shall be treated as an RE containing  
 6525 the collating symbol 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'.

6526 Collating symbols are recognized only inside bracket expressions. If the string is not a  
6527 collating element in the current locale, the expression is invalid.

6528 5. An equivalence class expression shall represent the set of collating elements belonging to  
6529 an equivalence class, as described in [Section 7.3.2.4](#) (on page 142). Only primary  
6530 equivalence classes shall be recognized. The class shall be expressed by enclosing any one  
6531 of the collating elements in the equivalence class within bracket-equal ("`[=`" and "`=]`")  
6532 delimiters. For example, if 'a', 'à', and '^' belong to the same equivalence class, then  
6533 "`[ [=a=] b ]`", "`[ [=à=] b ]`", and "`[ [=^=] b ]`" are each equivalent to "`[ aâ^ b ]`". If the  
6534 collating element does not belong to an equivalence class, the equivalence class  
6535 expression shall be treated as a collating symbol.

6536 6. A character class expression shall represent the union of two sets:  
6537 a. The set of single characters that belong to the character class, as defined in the  
6538 `LC_CTYPE` category in the current locale.  
6539 b. An unspecified set of multi-character collating elements.

6540 All character classes specified in the current locale shall be recognized. A character class  
6541 expression is expressed as a character class name enclosed within bracket-`<colon>` ("`[:`"  
6542 and "`:]`") delimiters.

6543 The following character class expressions shall be supported in all locales:

```
6544 [:alnum:]    [:cntrl:]    [:lower:]    [:space:]
6545 [:alpha:]    [:digit:]    [:print:]    [:upper:]
6546 [:blank:]    [:graph:]    [:punct:]    [:xdigit:]
```

6547 In addition, character class expressions of the form:

```
6548 [:name:]
```

6549 are recognized in those locales where the *name* keyword has been given a **charclass**  
6550 definition in the `LC_CTYPE` category.

6551 7. In the POSIX locale, a range expression represents the set of collating elements that fall  
6552 between two elements in the collation sequence, inclusive. In other locales, a range  
6553 expression has unspecified behavior: strictly conforming applications shall not rely on  
6554 whether the range expression is valid, or on the set of collating elements matched. A  
6555 range expression shall be expressed as the starting point and the ending point separated  
6556 by a `<hyphen-minus>` ('-').

6557 In the following, all examples assume the POSIX locale.

6558 The starting range point and the ending range point shall be a collating element or  
6559 collating symbol. An equivalence class expression used as a starting or ending point of a  
6560 range expression produces unspecified results. An equivalence class can be used portably  
6561 within a bracket expression, but only outside the range. If the represented set of collating  
6562 elements is empty, it is unspecified whether the expression matches nothing, or is treated  
6563 as invalid.

6564 The interpretation of range expressions where the ending range point is also the starting  
6565 range point of a subsequent range expression (for example, "`[a-m-o]`") is undefined.

6566 The `<hyphen-minus>` character shall be treated as itself if it occurs first (after an initial  
6567 '`^`', if any) or last in the list, or as an ending range point in a range expression. As  
6568 examples, the expressions "`[-ac]`" and "`[ac-]`" are equivalent and match any of the  
6569 characters 'a', 'c', or '-'; "`^[^-ac]`" and "`^[^ac-]`" are equivalent and match any  
6570 characters except 'a', 'c', or '-'; the expression "`[%--]`" matches any of the



6571 characters between '-' and '@' inclusive; the expression "[--@]" matches any of the  
 6572 characters between '-' and '@' inclusive; and the expression "[a--@]" is either invalid  
 6573 or equivalent to '@', because the letter 'a' follows the symbol '-' in the POSIX locale.  
 6574 To use a <hyphen-minus> as the starting range point, it shall either come first in the  
 6575 bracket expression or be specified as a collating symbol; for example, "[[.-.]-0]",  
 6576 which matches either a <right-square-bracket> or any character or collating element that  
 6577 collates between <hyphen-minus> and 0, inclusive.

6578 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the  
 6579 '^', if any) and the '-' last within the bracket expression.

6580 8. If a bracket expression contains at least three list elements, where the first and last list  
 6581 elements are the same single-character element of <period>, <equals-sign>, or <colon>,  
 6582 then it is unspecified whether the bracket expression will be treated as a collating symbol,  
 6583 equivalence class, or character class, respectively; treated as a matching list expression; or  
 6584 treated as an invalid bracket expression.

### 6585 9.3.6 BREs Matching Multiple Characters

6586 The following rules can be used to construct BREs matching multiple characters from BREs  
 6587 matching a single character:

6588 1. The concatenation of BREs shall match the concatenation of the strings matched by each  
 6589 component of the BRE.

6590 2. A subexpression can be defined within a BRE by enclosing it between the character pairs  
 6591 "\(" and "\)". Such a subexpression shall match whatever it would have matched  
 6592 without the "\(" and "\)", except that anchoring within subexpressions is optional  
 6593 behavior; see [Section 9.3.8](#) (on page 186). Subexpressions can be arbitrarily nested.

6594 3. The back-reference expression '\n' shall match the same (possibly empty) string of  
 6595 characters as was matched by a subexpression enclosed between "\(" and "\)"  
 6596 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the  
 6597 *n*th subexpression (the one that begins with the *n*th "\(" from the beginning of the  
 6598 pattern and ends with the corresponding paired "\)"). The expression is invalid if less  
 6599 than *n* subexpressions precede the '\n'. The string matched by a contained  
 6600 subexpression shall be within the string matched by the containing subexpression. If the  
 6601 containing subexpression does not match, or if there is no match for the contained  
 6602 subexpression within the string matched by the containing subexpression, then back-  
 6603 reference expressions corresponding to the contained subexpression shall not match.  
 6604 When a subexpression matches more than one string, a back-reference expression  
 6605 corresponding to the subexpression shall refer to the last matched string. For example, the  
 6606 expression "\^(.\*\)\1\$" matches strings consisting of two adjacent appearances of the  
 6607 same substring, and the expression "\(a\)\*\1" fails to match 'a', the expression  
 6608 "\(a\ (b\)\*\)\*\2" fails to match 'abab', and the expression "\^(ab\*\)\*\1\$" matches  
 6609 'ababbabb', but fails to match 'ababbab'.

6610 4. When a BRE matching a single character, a subexpression, or a back-reference is followed  
 6611 by the special character <asterisk> ('\*'), together with that <asterisk> it shall match  
 6612 what zero or more consecutive occurrences of the BRE would match. For example,  
 6613 "[ab]\*" and "[ab][ab]" are equivalent when matching the string "ab".

6614 5. When a BRE matching a single character, a subexpression, or a back-reference is followed  
 6615 by an interval expression of the format "\{m\}", "\{m,\}", or "\{m,n\}", together  
 6616 with that interval expression it shall match what repeated consecutive occurrences of the  
 6617 BRE would match. The values of *m* and *n* are decimal integers in the range 0

6618  $\leq m \leq n \leq \{RE\_DUP\_MAX\}$ , where  $m$  specifies the exact or minimum number of occurrences  
 6619 and  $n$  specifies the maximum number of occurrences. The expression " $\{m\}$ " shall  
 6620 match exactly  $m$  occurrences of the preceding BRE, " $\{m, \}$ " shall match at least  $m$   
 6621 occurrences, and " $\{m, n\}$ " shall match any number of occurrences between  $m$  and  $n$ ,  
 6622 inclusive.

6623 For example, in the string "abababcccccd" the BRE " $c\{3\}$ " is matched by  
 6624 characters seven to nine, the BRE " $\(ab\)\{4,\}$ " is not matched at all, and the BRE  
 6625 " $c\{1, 3\}d$ " is matched by characters ten to thirteen.

6626 The behavior of multiple adjacent duplication symbols ('\*' and intervals) produces undefined  
 6627 results.

6628 A subexpression repeated by an <asterisk> ('\*') or an interval expression shall not match a null  
 6629 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or  
 6630 minimum number of occurrences for the interval expression.

### 6631 9.3.7 BRE Precedence

6632 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[ ]
Subexpressions/back-references	\( \) \n
Single-character-BRE duplication	* \{m, n\}
Concatenation	
Anchoring	^ \$

### 6641 9.3.8 BRE Expression Anchoring

6642 A BRE can be limited to matching expressions that begin or end a string; this is called  
 6643 "anchoring". The <circumflex> and <dollar-sign> special characters shall be considered BRE  
 6644 anchors in the following contexts:

6645 1. A <circumflex> ('^') shall be an anchor when used as the first character of an entire BRE  
 6646 and, if the implementation does not match the escape sequence "\|" to the literal  
 6647 character '|', when used immediately following a "\|" escape sequence that is not  
 6648 inside a subexpression. The implementation may also treat a <circumflex> as an anchor  
 6649 when used inside a subexpression; in this case it shall be an anchor only when either of  
 6650 the following is true:

- 6651 • It is the first character of the subexpression.
- 6652 • It immediately follows a "\|" escape sequence and the implementation does not  
 6653 match the escape sequence "\|" to the literal character '|'.

6654 The <circumflex> shall anchor the expression (or optionally subexpression) to the  
 6655 beginning of a string; only sequences starting at the first character of a string shall be  
 6656 matched by the BRE. For example, the BRE " $^ab$ " matches "ab" in the string "abcdef",  
 6657 but fails to match in the string "cdefab". The BRE " $\(^ab\)$ " may match the former  
 6658 string. A portable BRE shall escape a leading <circumflex> in a subexpression to match a  
 6659 literal <circumflex>.

- 6660 2. A <dollar-sign> ( '\$ ' ) shall be an anchor when used as the last character of an entire BRE  
 6661 and, if the implementation does not match the escape sequence "\|" to the literal  
 6662 character '|', when used immediately preceding a "\|" escape sequence that is not  
 6663 inside a subexpression. The implementation may also treat a <dollar-sign> as an anchor  
 6664 when used inside a subexpression; in this case it shall be an anchor only when either of  
 6665 the following is true:
- 6666 • It is the last character of the subexpression.
  - 6667 • It immediately precedes a "\|" escape sequence and the implementation does not  
 6668 match the escape sequence "\|" to the literal character '| '.
- 6669 The <dollar-sign> shall anchor the expression (or optionally subexpression) to the end of  
 6670 the string being matched; the <dollar-sign> can be said to match the end-of-string  
 6671 following the last character. A portable BRE shall escape a trailing <dollar-sign> in a  
 6672 subexpression to match a literal <dollar-sign>.
- 6673 3. A BRE anchored by both '^' and '\$' shall match only an entire string. For example, the  
 6674 BRE "^abcdef\$" matches strings consisting only of "abcdef".

## 6675 9.4 Extended Regular Expressions

6676 The extended regular expression (ERE) notation and construction rules shall apply to utilities  
 6677 defined as using extended regular expressions; any exceptions to the following rules are noted  
 6678 in the descriptions of the specific utilities using EREs.

### 6679 9.4.1 EREs Matching a Single Character or Collating Element

6680 When not inside a bracket expression, the following shall match a single character:

- 6681 • an ERE ordinary character
- 6682 • an ERE special character, ']' , or '}' ' preceded by an unescaped <backslash>
- 6683 • a <period>

6684 A bracket expression shall match a single character or a single collating element. An ERE  
 6685 matching a single character enclosed in parentheses shall match the same as the ERE without  
 6686 parentheses would have matched.

### 6687 9.4.2 ERE Ordinary Characters

6688 An ordinary character is an ERE that matches itself. An ordinary character is any character in the  
 6689 supported character set, except for the ERE special characters listed in [Section 9.4.3](#) (on page  
 6690 188). When not inside a bracket expression, the interpretation of an ordinary character preceded  
 6691 by an unescaped <backslash> is undefined, except for the ']' ' and '}' ' characters; "\]" and  
 6692 "\}" shall match the ']' ' and '}' ' characters, respectively.

### 6693 9.4.3 ERE Special Characters

6694 An ERE special character has special properties in certain contexts. Outside those contexts, or  
6695 when preceded by an unescaped <backslash>, such a character shall be an ERE that matches the  
6696 special character itself. The extended regular expression special characters and the contexts in  
6697 which they shall have their special meaning are as follows:

- 6698 . [ \ ( The <period>, <left-square-bracket>, <backslash>, and <left-parenthesis> shall be  
6699 special except when used in a bracket expression (see [Section 9.3.5](#), on page 182). When  
6700 not inside a bracket expression, an unescaped <left-parenthesis> immediately followed  
6701 by a <right-parenthesis> produces undefined results. A <left-square-bracket> that is  
6702 unescaped and is not part of a bracket expression also produces undefined results.
- 6703 ) The <right-parenthesis> shall be special when matched with a preceding <left-  
6704 parenthesis>, both not inside a bracket expression.
- 6705 \* + ? { The <asterisk>, <plus-sign>, <question-mark>, and <left-brace> shall be special except  
6706 when used in a bracket expression (see [Section 9.3.5](#), on page 182). Any of the  
6707 following uses produce undefined results:
  - 6708 — If these characters appear first in an ERE, or immediately following an unescaped  
6709 <vertical-line>, <circumflex>, <dollar-sign>, or <left-parenthesis>
  - 6710 — If a <left-brace> is not part of a valid interval expression (see [Section 9.4.6](#), on  
6711 page 189)
- 6712 | The <vertical-line> is special except when used in a bracket expression (see [Section](#)  
6713 [9.3.5](#), on page 182). A <vertical-line> appearing first or last in an ERE, or immediately  
6714 following a <vertical-line> or a <left-parenthesis>, or immediately preceding a <right-  
6715 parenthesis>, produces undefined results.
- 6716 ^ The <circumflex> shall be special when used as an anchor (see [Section 9.4.9](#), on page  
6717 190). The <circumflex> shall signify a non-matching list expression when it occurs first  
6718 in a list, immediately following a <left-square-bracket> (see [Section 9.3.5](#), on page 182).
- 6719 \$ The <dollar-sign> shall be special when used as an anchor.

### 6720 9.4.4 Periods in EREs

6721 When not inside a bracket expression, a <period> ( ' . ' ) is an ERE that shall match any character  
6722 in the supported character set except NUL.

### 6723 9.4.5 ERE Bracket Expression

6724 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see [Section](#)  
6725 [9.3.5](#) (on page 182).

6726 **9.4.6 EREs Matching Multiple Characters**

6727 The following rules shall be used to construct EREs matching multiple characters from EREs  
6728 matching a single character:

6729 1. A concatenation of EREs shall match the concatenation of the character sequences  
6730 matched by each component of the ERE. A concatenation of EREs enclosed in parentheses  
6731 shall match whatever the concatenation without the parentheses matches. For example,  
6732 both the ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of  
6733 the string "abcdefabcdef".

6734 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6735 by the special character <plus-sign> ('+'), together with that <plus-sign> it shall match  
6736 what one or more consecutive occurrences of the ERE would match. For example, the  
6737 ERE "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde".  
6738 And, "[ab]+" and "[ab][ab]\*" are equivalent.

6739 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6740 by the special character <asterisk> ('\*'), together with that <asterisk> it shall match  
6741 what zero or more consecutive occurrences of the ERE would match. For example, the  
6742 ERE "b\*c" matches the first character in the string "cabbbbcde", and the ERE "b\*cd"  
6743 matches the third to seventh characters in the string "cabbbbcdebbbbbbbcdbc". And,  
6744 "[ab]\*" and "[ab][ab]" are equivalent when matching the string "ab".

6745 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6746 by the special character <question-mark> ('?'), together with that <question-mark> it  
6747 shall match what zero or one consecutive occurrences of the ERE would match. For  
6748 example, the ERE "b?c" matches the second character in the string "acabbbbcde".

6749 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed  
6750 by an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that  
6751 interval expression it shall match what repeated consecutive occurrences of the ERE  
6752 would match. The values of *m* and *n* are decimal integers in the range  $0 \leq m \leq n \leq \{RE\_DUP\_MAX\}$ , where *m* specifies the exact or minimum number of occurrences  
6753 and *n* specifies the maximum number of occurrences. The expression "{m}" matches  
6754 exactly *m* occurrences of the preceding ERE, "{m,}" matches at least *m* occurrences, and  
6755 "{m,n}" matches any number of occurrences between *m* and *n*, inclusive.  
6756

6757 For example, in the string "abababcccccd" the ERE "c{3}" is matched by characters  
6758 seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

6759 6. Each of the duplication symbols ('+', '\*', '?', and intervals) can be suffixed by the  
6760 repetition modifier '?' (<question-mark>), in which case matching behavior for that  
6761 repetition shall be changed from the leftmost longest possible match to the leftmost  
6762 shortest possible match, including the null match (see [Section A.9](#), on page 3709). For  
6763 example, the ERE ".\*c" matches up to and including the last character ('c') in the  
6764 string "abc abc", whereas the ERE ".\*?c" matches up to and including the first  
6765 character 'c', the third character in the string.

6766 If the REG\_MINIMAL flag, defined in the <regex.h> header, is used when compiling an  
6767 ERE via *regcomp()*, the leftmost shortest possible match shall be the default for all  
6768 duplication symbols, and the repetition modifier '?' can be used to select the leftmost  
6769 longest possible match for the repetition it modifies.

6770 The behavior of multiple adjacent duplication symbols ('+', '\*', '?', and intervals, possibly  
6771 suffixed by the repetition modifier '?') produces undefined results.

6772 An ERE matching a single character repeated by an '\*', '?', or an interval expression shall not

6773 match a null expression unless this is the only match for the repetition or it is necessary to satisfy  
6774 the exact or minimum number of occurrences for the interval expression.

#### 6775 9.4.7 ERE Alternation

6776 Two EREs separated by the special character <vertical-line> ('|') shall match a string that is  
6777 matched by either. For example, the ERE "a(bc|d)" matches the string "abc" and the  
6778 string "ad". Single characters, or expressions matching single characters, separated by the  
6779 <vertical-line> and enclosed in parentheses, shall be treated as an ERE matching a single  
6780 character.

#### 6781 9.4.8 ERE Precedence

6782 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
6783 Collation-related bracket symbols	[==] [::] [..]
6784 Escaped characters	\<special character>
6785 Bracket expression	[]
6786 Grouping	()
6787 Single-character-ERE duplication	* + ? {m,n}
6788 Concatenation	
6789 Anchoring	^ \$
6790 Alternation	
6791	

6792 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"  
6793 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of  
6794 precedence than alternation).

#### 6795 9.4.9 ERE Expression Anchoring

6796 An ERE can be limited to matching expressions that begin or end a string; this is called  
6797 "anchoring". The <circumflex> and <dollar-sign> special characters shall be considered ERE  
6798 anchors when used anywhere except inside a bracket expression. This shall have the following  
6799 effects:

- 6800 1. When not inside a bracket expression, a <circumflex> ('^') shall anchor the expression  
6801 or subexpression it begins to the beginning of a string; such an expression or  
6802 subexpression can match only a sequence starting at the first character of a string. For  
6803 example, the EREs "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to  
6804 match in the string "cdefab", and the ERE "a^b" is valid, but can never match because  
6805 the 'a' prevents the expression "a^b" from matching starting at the first character.
- 6806 2. When not inside a bracket expression, a <dollar-sign> ('\$') shall anchor the expression  
6807 or subexpression it ends to the end of a string; such an expression or subexpression can  
6808 match only a sequence ending at the last character of a string. For example, the EREs  
6809 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string  
6810 "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents  
6811 the expression "e\$f" from matching ending at the last character.

## 6812 9.5 Regular Expression Grammar

6813 Grammars describing the syntax of both basic and extended regular expressions are presented in  
6814 this section. The grammar takes precedence over the text. See XCU [Section 1.3](#) (on page 2461).

### 6815 9.5.1 BRE/ERE Grammar Lexical Conventions

6816 The lexical conventions for regular expressions are as described in this section.

6817 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6818 The following tokens are processed (in addition to those string constants shown in the  
6819 grammar):

6820	<b>COLL_ELEM_SINGLE</b>	Any single-character collating element, unless it is a <b>META_CHAR</b> .
6821	<b>COLL_ELEM_MULTI</b>	Any multi-character collating element.
6822	<b>BACKREF</b>	Applicable only to basic regular expressions. The character string consisting of a <backslash> character followed by a single-digit numeral, '1' to '9'.
6823		
6824		
6825	<b>DUP_COUNT</b>	Represents a numeric constant. It shall be an integer in the range 0 ≤ <b>DUP_COUNT</b> ≤ { <b>RE_DUP_MAX</b> }. This token is only recognized when the context of the grammar requires it. At all other times, digits not preceded by a <backslash> character are treated as <b>ORD_CHAR</b> .
6826		
6827		
6828		
6829	<b>META_CHAR</b>	One of the characters:
6830		^      When found first in a bracket expression
6831		–      When found anywhere but first (after an initial '^', if any)
6832		or last in a bracket expression, or as the ending range point
6833		in a range expression
6834		]      When found anywhere but first (after an initial '^', if any)
6835		in a bracket expression
6836	<b>L_ANCHOR</b>	Applicable only to basic regular expressions. The character '^' when it appears either as the first character of a basic regular expression or, if the implementation does not match the escape sequence "\ " to the literal character ' ', when used immediately following a "\ " escape sequence that is not inside a subexpression, and when not <b>QUOTED_CHAR</b> . The '^' may be recognized as an anchor elsewhere; see <a href="#">Section 9.3.8</a> (on page 186).
6837		
6838		
6839		
6840		
6841		
6842		
6843	<b>ORD_CHAR</b>	A character, other than one of the special characters in <b>SPEC_CHAR</b> .
6844	<b>QUOTED_CHAR</b>	In a BRE, one of the character sequences:
6845		\^    \.    \*    \[    \]    \\$    \\
6846		On implementations where the escape sequences "\?", "\+", and "\ " match the literal characters '?', '+', and ' ', respectively, <b>QUOTED_CHAR</b> shall also include:
6847		\?    \+    \
6848		
6849		
6850		In an ERE, one of the character sequences:
6851		\^    \.    \[    \]    \\$    \(    \)    \

6852		<code>\* \+ \? \{ \} \\</code>
6853	<b>R_ANCHOR</b>	(Applicable only to basic regular expressions.) The character '\$' when it appears either as the last character of a basic regular expression or, if the implementation does not match the escape sequence "\ " to the literal character ' ', when used immediately preceding a "\ " escape sequence that is not inside a subexpression, and when not <b>QUOTED_CHAR</b> . The '\$' may be recognized as an anchor elsewhere; see <a href="#">Section 9.3.8</a> (on page 186).
6854		
6855		
6856		
6857		
6858		
6859		
6860	<b>SPEC_CHAR</b>	For basic regular expressions, one of the following special characters:
6861		. Anywhere except inside bracket expressions
6862		\ Anywhere except inside bracket expressions
6863		[ Anywhere except inside bracket expressions
6864		^ When used as an anchor (see <a href="#">Section 9.3.8</a> , on page 186)
6865		\$ When used as an anchor
6866		* Anywhere except first in an entire RE, anywhere in a bracket expression, directly following "\(", directly following an anchoring '^'
6867		
6868		
6869		For extended regular expressions, shall be one of the following special characters found anywhere except inside bracket expressions:
6870		
6871		^ . [ \$ ( )
6872		* + ? { \
6873		The close-parenthesis shall be considered special in this context only if matched with a preceding open-parenthesis.
6874		

## 6875 9.5.2 RE and Bracket Expression Grammar

6876 This section presents the grammar for basic regular expressions, including the bracket  
6877 expression grammar that is common to both BREs and EREs.

6878	%token	ORD_CHAR QUOTED_CHAR DUP_COUNT
6879	%token	BACKREF L_ANCHOR R_ANCHOR
6880	%token	Back_open_paren Back_close_paren
6881	/*	'\(' '\)' */
6882	%token	Back_open_brace Back_close_brace
6883	/*	'\{' '\}' */
6884	/*	The following shall be tokens on implementations where
6885		\?, \+, and \  are not included in QUOTED_CHAR */
6886	%token	Back_qm Back_plus Back_bar
6887	/*	'\?' '\+' '\ ' */
6888	/*	The following tokens are for the Bracket Expression
6889		grammar common to both REs and EREs. */
6890	%token	COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6891	%token	Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close



```

6892      /*          '['          '='          '['          '.'          ':'          ':'          */
6893      %token      class_name
6894      /* class_name is a keyword to the LC_CTYPE locale category */
6895      /* (representing a character class) in the current locale */
6896      /* and is only recognized between [: and :] */
6897
6898      %start      basic_reg_exp
6899      %%
6900
6901      /* -----
6902      Basic Regular Expression
6903      -----
6904      */
6905      basic_reg_exp      :          BRE_branch
6906                        | basic_reg_exp Back_bar BRE_branch /* if Back_bar
6907                        |                                     is a token */
6908                        ;
6909      BRE_branch         :          BRE_expression
6910                        | BRE_branch BRE_expression
6911                        ;
6912      BRE_expression    :          simple_BRE
6913                        | L_ANCHOR
6914                        |                                     R_ANCHOR
6915                        | L_ANCHOR R_ANCHOR
6916                        | L_ANCHOR simple_BRE
6917                        | simple_BRE R_ANCHOR
6918                        | L_ANCHOR simple_BRE R_ANCHOR
6919                        ;
6920      simple_BRE        :      nondupl_BRE
6921                        | nondupl_BRE BRE_dupl_symbol
6922                        ;
6923      nondupl_BRE       :      one_char_or_coll_elem_BRE
6924                        | Back_open_paren basic_reg_exp Back_close_paren
6925                        | BACKREF
6926                        ;
6927      one_char_or_coll_elem_BRE : ORD_CHAR
6928                        | QUOTED_CHAR
6929                        | '.'
6930                        | bracket_expression
6931                        ;
6932      BRE_dupl_symbol   :      '*'
6933                        | Back_qm /* if Back_qm is a token */
6934                        | Back_plus /* if Back_plus is a token */
6935                        | Back_open_brace DUP_COUNT Back_close_brace
6936                        | Back_open_brace DUP_COUNT ',' Back_close_brace
6937                        | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6938                        ;
6939
6940      /* -----
6941      Bracket Expression
6942      -----
6943      */
6944      bracket_expression : '[' matching_list ']'
6945                        | '[' nonmatching_list ']'

```

```

6943         ;
6944     matching_list : bracket_list
6945         ;
6946     nonmatching_list : '^' bracket_list
6947         ;
6948     bracket_list : follow_list
6949                 | follow_list '-'
6950         ;
6951     follow_list :          expression_term
6952                | follow_list expression_term
6953         ;
6954     expression_term : single_expression
6955                    | range_expression
6956         ;
6957     single_expression : end_range
6958                       | character_class
6959                       | equivalence_class
6960         ;
6961     range_expression : start_range end_range
6962                     | start_range '-'
6963         ;
6964     start_range : end_range '-'
6965         ;
6966     end_range : COLL_ELEM_SINGLE
6967              | collating_symbol
6968         ;
6969     collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
6970                    | Open_dot COLL_ELEM_MULTI Dot_close
6971                    | Open_dot META_CHAR Dot_close
6972         ;
6973     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6974                      | Open_equal COLL_ELEM_MULTI Equal_close
6975         ;
6976     character_class : Open_colon class_name Colon_close
6977         ;

```

6978 Note that although the BRE grammar appears always to permit **L\_ANCHOR** or **R\_ANCHOR**  
6979 inside "**\**(" and "**\**", the lexical conventions (see [Section 9.5.1](#), on page 191) imply that '^'  
6980 and '\$' may be ordinary characters there. This reflects the semantic limits on the application, as  
6981 noted in [Section 9.3.8](#) (on page 186). Since it is an implementation option whether to interpret  
6982 '^' and '\$' as anchors in these locations, conforming applications cannot use unescaped '^'  
6983 and '\$' in positions inside "**\**(" and "**\**" that might be interpreted as anchors.

### 6984 9.5.3 ERE Grammar

6985 This section presents the grammar for extended regular expressions, excluding the bracket  
6986 expression grammar.

6987 **Note:** The bracket expression grammar and the associated **%token** lines are identical between BREs  
6988 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6989 %token  ORD_CHAR QUOTED_CHAR DUP_COUNT
6990 %start  extended_reg_exp
6991 %%

```

```

6992      /* -----
6993         Extended Regular Expression
6994         -----
6995      */
6996      extended_reg_exp      :          ERE_branch
6997                          | extended_reg_exp '|' ERE_branch
6998                          ;
6999      ERE_branch            :          ERE_expression
7000                          | ERE_branch ERE_expression
7001                          ;
7002      ERE_expression       : one_char_or_coll_elem_ERE
7003                          | '^'
7004                          | '$'
7005                          | '(' extended_reg_exp ')'
7006                          | ERE_expression ERE_dupl_symbol
7007                          ;
7008      one_char_or_coll_elem_ERE : ORD_CHAR
7009                          | QUOTED_CHAR
7010                          | '.'
7011                          | bracket_expression
7012                          ;
7013      ERE_dupl_symbol       : '*'
7014                          | '+'
7015                          | '?'
7016                          | '{' DUP_COUNT '}'
7017                          | '{' DUP_COUNT ',' '}'
7018                          | '{' DUP_COUNT ',' DUP_COUNT '}'
7019                          ;

```

7020 The ERE grammar does not permit several constructs that previous sections specify as having  
7021 undefined results. Additionally, there are some constructs which the grammar permits but  
7022 which still give undefined results:

- 7023 • **ORD\_CHAR** preceded by an unescaped <backslash> character
- 7024 • One or more *ERE\_dupl\_symbols* appearing first in an ERE, or immediately following '|',  
7025 '^', '(', or '\$'
- 7026 • '{' not part of a valid *ERE\_dupl\_symbol*
- 7027 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or  
7028 immediately preceding ')'

7029 Implementations are permitted to extend the language to allow these. Strictly Conforming  
7030 applications cannot use such constructs.



# Directory Structure and Devices

## 7033 10.1 Directory Structure and Files

7034 The following directories shall exist on conforming systems and conforming applications shall  
7035 make use of them only as described. Strictly conforming applications shall not assume the  
7036 ability to create files in any of these directories, unless specified below.

7037 **/** The root directory.

7038 **/dev** Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

7039 The following directory shall exist on conforming systems and shall be used as described:

7040 **/tmp** A directory made available for applications that need a place to create temporary  
7041 files. Applications shall be allowed to create files in this directory, but shall not  
7042 assume that such files are preserved between invocations of the application.

7043 The following files shall exist on conforming systems and shall be both readable and writable:

7044 **/dev/null** An empty data source and infinite data sink. Data written to **/dev/null** shall be  
7045 discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

7046 **/dev/tty** In each process, a synonym for the controlling terminal associated with the process  
7047 group of that process, if any. It is useful for programs or shell procedures that wish  
7048 to be sure of writing messages to or reading data from the terminal no matter how  
7049 output has been redirected. It can also be used for applications that demand the  
7050 name of a file for output, when typed output is desired and it is tiresome to find  
7051 out what terminal is currently in use.

7052 The following file shall exist on conforming systems and need not be readable or writable:

7053 **/dev/console** The **/dev/console** file is a generic name given to the system console (see [Section](#)  
7054 [3.376](#), on page 86). It is usually linked to an implementation-defined special file. It  
7055 shall provide an interface to the system console conforming to the requirements of  
7056 [Chapter 11](#) (on page 199).

## 7057 10.2 Output Devices and Terminal Types

7058 The utilities in the Shell and Utilities volume of POSIX.1-2024 historically have been  
7059 implemented on a wide range of terminal types, but a conforming implementation need not  
7060 support all features of all utilities on every conceivable terminal. POSIX.1-2024 states which  
7061 features are optional for certain classes of terminals in the individual utility description sections.  
7062 The implementation shall document in the system documentation which terminal types it  
7063 supports and which of these features and utilities are not supported by each terminal.

7064 When a feature or utility is not supported on a specific terminal type, as allowed by  
7065 POSIX.1-2024, and the implementation considers such a condition to be an error preventing use  
7066 of the feature or utility, the implementation shall indicate such conditions through diagnostic  
7067 messages or exit status values or both (as appropriate to the specific utility description) that

7068 inform the user that the terminal type lacks the appropriate capability.

7069 POSIX.1-2024 uses a notational convention based on historical practice that identifies some of

7070 the control characters defined in [Section 7.3.1](#) (on page 131) in a manner easily remembered by

7071 users on many terminals. The correspondence between this “<control>-char” notation and the

7072 actual control characters is shown in the following table. When POSIX.1-2024 refers to a

7073 character by its <control>-name, it is referring to the actual control character shown in the Value

7074 column of the table, which is not necessarily the exact control key sequence on all terminals.

7075 Some terminals have keyboards that do not allow the direct transmission of all the non-

7076 alphanumeric characters shown. In such cases, the system documentation shall describe which

7077 data sequences transmitted by the terminal are interpreted by the system as representing the

7078 special characters.

7079 **Table 10-1** Control Character Names

Name	Value	Name	Value
<control>-A	<SOH>	<control>-Q	<DC1>
<control>-B	<STX>	<control>-R	<DC2>
<control>-C	<ETX>	<control>-S	<DC3>
<control>-D	<EOT>	<control>-T	<DC4>
<control>-E	<ENQ>	<control>-U	<NAK>
<control>-F	<ACK>	<control>-V	<SYN>
<control>-G	<BEL>	<control>-W	<ETB>
<control>-H	<BS>	<control>-X	<CAN>
<control>-I	<HT>	<control>-Y	<EM>
<control>-J	<LF>	<control>-Z	<SUB>
<control>-K	<VT>	<control>-[	<ESC>
<control>-L	<FF>	<control>-\	<FS>
<control>-M	<CR>	<control>-]	<GS>
<control>-N	<SO>	<control>-^	<RS>
<control>-O	<SI>	<control>-_	<US>
<control>-P	<DLE>	<control>-?	<DEL>

7097 **Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that

7098 the keystrokes represent control-shift-letter sequences.

# General Terminal Interface

7101 This chapter describes a general terminal interface that shall be provided. It shall be supported  
 7102 on any asynchronous communications ports if the implementation provides them. It is  
 7103 implementation-defined whether it supports network connections or synchronous ports, or  
 7104 both.

## 7105 11.1 Interface Characteristics

### 7106 11.1.1 Opening a Terminal Device File

7107 When a terminal device file is opened, it normally causes the thread to wait until a connection is  
 7108 established. In practice, application programs seldom open these files; they are opened by  
 7109 special programs and become an application's standard input, output, and error files.

7110 Cases where applications do open a terminal device are as follows:

- 7111 1. Opening `/dev/tty`, or the pathname returned by `ctermid()`, in order to obtain a file  
 7112 descriptor for the controlling terminal; see [Section 11.1.3](#) (on page 200).
- 7113 2. Opening the subsidiary side of a pseudo-terminal; see XSH [ptsname\(\)](#).
- 7114 3. Opening a modem or similar piece of equipment connected by a serial line. In this case,  
 7115 the terminal parameters (see [Section 11.2](#), on page 205) may be initialized to default  
 7116 settings by the implementation in between the last close of the device by any process and  
 7117 the next open of the device, or they may persist from one use to the next. The terminal  
 7118 parameters can be set to values that ensure the terminal behaves in a conforming manner  
 7119 by means of the `O_TTY_INIT` open flag when opening a terminal device that is not  
 7120 already open in any process, or by executing the `stty` utility with the operand `sane`.

7121 As described in `open()`, opening a terminal device file with the `O_NONBLOCK` flag clear shall  
 7122 cause the thread to block until the terminal device is ready and available. If `CLOCAL` mode is  
 7123 not set, this means blocking until a connection is established. If `CLOCAL` mode is set in the  
 7124 terminal, or the `O_NONBLOCK` flag is specified in the `open()`, the `open()` function shall return a  
 7125 file descriptor without waiting for a connection to be established.

### 7126 11.1.2 Process Groups

7127 A terminal may have a foreground process group associated with it. This foreground process  
 7128 group plays a special role in handling signal-generating input characters, as discussed in [Section](#)  
 7129 [11.1.9](#) (on page 203).

7130 A command interpreter process supporting job control can allocate the terminal to different jobs,  
 7131 or process groups, by placing related processes in a single process group and associating this  
 7132 process group with the terminal. A terminal's foreground process group may be set or examined  
 7133 by a process, assuming the permission requirements are met; see `tcgetpgrp()` and `tcsetpgrp()`.

7134 The terminal interface aids in this allocation by restricting access to the terminal by processes  
7135 that are not in the current process group; see [Section 11.1.4](#).

7136 When there is no longer any process whose process ID or process group ID matches the  
7137 foreground process group ID, the terminal shall have no foreground process group. It is  
7138 unspecified whether the terminal has a foreground process group when there is a process whose  
7139 process ID matches the foreground process group ID, but whose process group ID does not. No  
7140 actions defined in POSIX.1-2024, other than allocation of a controlling terminal or a successful  
7141 call to *tcsetpgrp()*, shall cause a process group to become the foreground process group of the  
7142 terminal.

### 7143 11.1.3 The Controlling Terminal

7144 A terminal may belong to a process as its controlling terminal. Each process of a session that has  
7145 a controlling terminal has the same controlling terminal. A terminal may be the controlling  
7146 terminal for at most one session. The controlling terminal for a session is allocated by the session  
7147 leader in an implementation-defined manner. If a session leader has no controlling terminal, and  
7148 opens a terminal device file that is not already associated with a session without using the  
7149 O\_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the  
7150 controlling terminal of the session leader. If a process which is not a session leader opens a  
7151 terminal file, or the O\_NOCTTY option is used on *open()*, then that terminal shall not become  
7152 the controlling terminal of the calling process. When a controlling terminal becomes associated  
7153 with a session, its foreground process group shall be set to the process group of the session  
7154 leader.

7155 The controlling terminal is inherited by a child process during a *fork()* function call. A process  
7156 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;  
7157 other processes remaining in the old session that had this terminal as their controlling terminal  
7158 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in  
7159 the current session) associated with the controlling terminal, it is unspecified whether all  
7160 processes that had that terminal as their controlling terminal cease to have any controlling  
7161 terminal. Whether and how a session leader can reacquire a controlling terminal after the  
7162 controlling terminal has been relinquished in this fashion is unspecified. A process does not  
7163 relinquish its controlling terminal simply by closing all of its file descriptors associated with the  
7164 controlling terminal if other processes continue to have it open.

7165 When a controlling process terminates, the controlling terminal is dissociated from the current  
7166 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by  
7167 other processes in the earlier session may be denied, with attempts to access the terminal treated  
7168 as if a modem disconnect had been sensed.

### 7169 11.1.4 Terminal Access Control

7170 If a process is in the foreground process group of its controlling terminal, read operations shall  
7171 be allowed, as described in [Section 11.1.5](#) (on page 201). Any attempts by a process in a  
7172 background process group to read from its controlling terminal cause its process group to be  
7173 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is  
7174 ignoring the SIGTTIN signal or the reading thread is blocking the SIGTTIN signal, or if the  
7175 process group of the reading process is orphaned, the *read()* shall return *-1*, with *errno* set to  
7176 [EIO] and no signal shall be sent. The default action of the SIGTTIN signal shall be to stop the  
7177 process to which it is sent. See [<signal.h>](#).

7178 If a process is in the foreground process group of its controlling terminal, write operations shall



7179 be allowed as described in [Section 11.1.8](#) (on page 203). Attempts by a process in a background  
7180 process group to write to its controlling terminal shall cause the process group to be sent a  
7181 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if  
7182 TOSTOP is set and the process is ignoring the SIGTTOU signal or the writing thread is blocking  
7183 the SIGTTOU signal, the process is allowed to write to the terminal and the SIGTTOU signal is  
7184 not sent. If TOSTOP is set, the process group of the writing process is orphaned, the writing  
7185 process is not ignoring the SIGTTOU signal, and the writing thread is not blocking the SIGTTOU  
7186 signal, the *write()* shall return  $-1$ , with *errno* set to [EIO] and no signal shall be sent.

7187 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that  
7188 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set  
7189 (see [Section 11.2.5](#) (on page 210), *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcsetattr()*, *tcsetpgrp()*,  
7190 and *tcsetwinsize()*).

### 7191 11.1.5 Input Processing and Reading Data

7192 A terminal device associated with a terminal device file may operate in full-duplex mode, so  
7193 that data may arrive even while output is occurring. Each terminal device file has an input  
7194 queue associated with it, into which incoming data is stored by the system before being read by  
7195 a process. The system may impose a limit, {MAX\_INPUT}, on the number of bytes that may be  
7196 stored in the input queue. The behavior of the system when this limit is exceeded is  
7197 implementation-defined.

7198 Two general kinds of input processing are available, determined by whether the terminal device  
7199 file is in canonical mode or non-canonical mode. These modes are described in [Section 11.1.6](#) (on  
7200 page 202) and [Section 11.1.7](#) (on page 202). Additionally, input characters are processed  
7201 according to the *c\_iflag* (see [Section 11.2.2](#), on page 206) and *c\_lflag* (see [Section 11.2.5](#), on page  
7202 210) fields. Such processing can include “echoing”, which in general means transmitting input  
7203 characters immediately back to the terminal when they are received from the terminal. This is  
7204 useful for terminals that can operate in full-duplex mode.

7205 The manner in which data is provided to a process reading from a terminal device file is  
7206 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether  
7207 or not the O\_NONBLOCK flag is set by *open()* or *fcntl()*.

7208 If the O\_NONBLOCK flag is clear, then the read request shall be blocked until data is available  
7209 or a signal has been received. If the O\_NONBLOCK flag is set, then the read request shall be  
7210 completed, without blocking, in one of three ways:

- 7211 1. If there is enough data available to satisfy the entire request, the *read()* shall complete  
7212 successfully and shall return the number of bytes read.
- 7213 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete  
7214 successfully, having read as much data as possible, and shall return the number of bytes it  
7215 was able to read.
- 7216 3. If there is no data available, the *read()* shall return  $-1$ , with *errno* set to [EAGAIN].

7217 When data is available depends on whether the input processing mode is canonical or non-  
7218 canonical. [Section 11.1.6](#) (on page 202) and [Section 11.1.7](#) (on page 202) describe each of these  
7219 input processing modes.

### 7220 11.1.6 Canonical Mode Input Processing

7221 In canonical mode input processing, terminal input is processed in units of lines. A line is  
 7222 delimited by a <newline> character (NL), an end-of-file character (EOF), or an end-of-line (EOL)  
 7223 character. See Section 11.1.9 (on page 203) for more information on EOF and EOL. This means  
 7224 that a read request shall not return until an entire line has been typed or a signal has been  
 7225 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall  
 7226 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even  
 7227 one, may be requested in a *read()* without losing information.

7228 If {MAX\_CANON} is defined for this terminal device, it shall be a limit on the number of bytes  
 7229 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If  
 7230 {MAX\_CANON} is not defined, there shall be no such limit; see *pathconf()*.

7231 Erase and kill processing occur when either of two special characters, the ERASE and KILL  
 7232 characters (see Section 11.1.9, on page 203), is received. This processing shall affect data in the  
 7233 input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited  
 7234 data makes up the current line. The ERASE character shall delete the last character in the current  
 7235 line, if there is one. The KILL character shall delete all data in the current line, if there is any.  
 7236 The ERASE and KILL characters shall have no effect if there is no data in the current line. The  
 7237 ERASE and KILL characters themselves shall not be placed in the input queue.

### 7238 11.1.7 Non-Canonical Mode Input Processing

7239 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and  
 7240 kill processing shall not occur. The values of the MIN and TIME members of the *c\_cc* array are  
 7241 used to determine how to process the bytes received. POSIX.1-2024 does not specify whether  
 7242 the setting of O\_NONBLOCK takes precedence over MIN or TIME settings. Therefore, if  
 7243 O\_NONBLOCK is set, *read()* may return immediately, regardless of the setting of MIN or TIME.  
 7244 Also, if no data is available, *read()* may either return 0, or return -1 with *errno* set to [EAGAIN].

7245 MIN represents the minimum number of bytes that should be received when the *read()* function  
 7246 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and  
 7247 short-term data transmissions. If MIN is greater than {MAX\_INPUT}, the response to the request  
 7248 is undefined. The four possible values for MIN and TIME and their interactions are described  
 7249 below.

#### 7250 Case A: MIN>0, TIME>0

7251 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is  
 7252 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction  
 7253 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall  
 7254 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer  
 7255 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN  
 7256 bytes are received, the characters received to that point shall be returned to the user. Note that if  
 7257 TIME expires at least one byte shall be returned because the timer would not have been enabled  
 7258 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and  
 7259 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is  
 7260 in the buffer at the time of the *read()*, the result shall be as if data has been received immediately  
 7261 after the *read()*.

7262 **Case B: MIN>0, TIME=0**

7263 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A  
 7264 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall  
 7265 block until MIN bytes are received), or a signal is received. A program that uses case B to read  
 7266 record-based terminal I/O may block indefinitely in the read operation.

7267 **Case C: MIN=0, TIME>0**

7268 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read  
 7269 timer that shall be activated as soon as the *read()* function is processed. A read shall be satisfied  
 7270 as soon as a single byte is received or the read timer expires. Note that in case C if the timer  
 7271 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be  
 7272 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely  
 7273 waiting for a byte; if no byte is received within TIME\*0.1 seconds after the read is initiated, the  
 7274 *read()* shall return a value of zero, having read no data. If data is in the buffer at the time of the  
 7275 *read()*, the timer shall be started as if data has been received immediately after the *read()*.

7276 **Case D: MIN=0, TIME=0**

7277 The minimum of either the number of bytes requested or the number of bytes currently  
 7278 available shall be returned without waiting for more bytes to be input. If no characters are  
 7279 available, *read()* shall return a value of zero, having read no data.

7280 **11.1.8 Writing Data and Output Processing**

7281 When a process writes one or more bytes to a terminal device file, they are processed according  
 7282 to the *c\_oflag* field (see [Section 11.2.3](#), on page 207). The implementation may provide a  
 7283 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have  
 7284 been scheduled for transmission to the device, but the transmission has not necessarily  
 7285 completed. See *write()* for the effects of O\_NONBLOCK on *write()*.

7286 **11.1.9 Special Characters**

7287 Certain characters have special functions on input or output or both. These functions are  
 7288 summarized as follows:

7289 **INTR** Special character on input, which is recognized if the ISIG flag is set. Generates a  
 7290 SIGINT signal which is sent to all processes in the foreground process group for which  
 7291 the terminal is the controlling terminal. If ISIG is set, the INTR character shall be  
 7292 discarded when processed.

7293 **QUIT** Special character on input, which is recognized if the ISIG flag is set. Generates a  
 7294 SIGQUIT signal which is sent to all processes in the foreground process group for  
 7295 which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be  
 7296 discarded when processed.

7297 **ERASE** Special character on input, which is recognized if the ICANON flag is set. Erases the  
 7298 last character in the current line; see [Section 11.1.6](#) (on page 202). It shall not erase  
 7299 beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is  
 7300 set, the ERASE character shall be discarded when processed.

7301 **KILL** Special character on input, which is recognized if the ICANON flag is set. Deletes the  
 7302 entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL  
 7303 character shall be discarded when processed.

7304	EOF	Special character on input, which is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process without waiting for a <newline>, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be returned from the <i>read()</i> , representing an end-of-file indication. If ICANON is set, the EOF character shall be discarded when processed.
7305		
7306		
7307		
7308		
7309		
7310	NL	Special character on input, which is recognized if the ICANON flag is set. It is the line delimiter <newline>. It cannot be changed.
7311		
7312	EOL	Special character on input, which is recognized if the ICANON flag is set. It is an additional line delimiter, like NL.
7313		
7314	SUSP	If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character shall be discarded when processed.
7315		
7316		
7317	STOP	Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character shall be discarded when processed.
7318		
7319		
7320		
7321	START	Special character on both input and output, which is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character shall be discarded when processed.
7322		
7323		
7324		
7325	CR	Special character on input, which is recognized if the ICANON flag is set; it is the <carriage-return> character. When ICANON and ICRNL are set and IGNCR is not set, this character shall be translated into an NL, and shall have the same effect as an NL character. It cannot be changed.
7326		
7327		
7328		
7329		The NL and CR characters cannot be changed. It is implementation-defined whether the START and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and SUSP shall be changeable to suit individual tastes. Special character functions associated with changeable special control characters can be disabled individually.
7330		
7331		
7332		
7333		If two or more special characters have the same value, the function performed when that character is received is undefined.
7334		
7335		A special character is recognized not only by its value, but also by its context; for example, an implementation may support multi-byte sequences that have a meaning different from the meaning of the bytes when considered individually. Implementations may also support additional single-byte functions. These implementation-defined multi-byte or single-byte functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without interpretation, except as required to recognize the special characters defined in this section.
7336		
7337		
7338		
7339		
7340		
7341	XSI	If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding <backslash> character, in which case no special function shall occur.
7342		

7343 **11.1.10 Modem Disconnect**

7344 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if  
 7345 CLOCAL is not set in the *c\_cflag* field for the terminal (see Section 11.2.4, on page 209), the  
 7346 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling  
 7347 terminal. Unless other arrangements have been made, this shall cause the controlling process to  
 7348 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of  
 7349 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-  
 7350 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*  
 7351 also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent  
 7352 *write()* to the terminal device shall return  $-1$ , with *errno* set to [EIO], until the device is closed.

7353 **11.1.11 Closing a Terminal Device File**

7354 The last process to close a terminal device file shall cause any output to be sent to the device and  
 7355 shall cause any input to be discarded. If HUPCL is set in the control modes and the  
 7356 communications port supports a disconnect function, the terminal device shall perform a  
 7357 disconnect.

7358 **11.2 Parameters that Can be Set**7359 **11.2.1 The termios Structure**

7360 Routines that need to control certain terminal I/O characteristics shall do so by using the  
 7361 **termios** structure as defined in the [<termios.h>](#) header.

7362 Since the **termios** structure may include additional members, and the standard members may  
 7363 include both standard and non-standard modes, the structure should never be initialized  
 7364 directly by the application as this may cause the terminal to behave in a non-conforming  
 7365 manner. When opening a terminal device (other than a pseudo-terminal) that is not already open  
 7366 in any process, it should be opened with the O\_TTY\_INIT flag before initializing the structure  
 7367 using *tcgetattr()* to ensure that any non-standard elements of the **termios** structure are set to  
 7368 values that result in conforming behavior of the terminal interface.

7369 The members of the **termios** structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
<b>tcflag_t</b>		<i>c_iflag</i>	Input modes.
<b>tcflag_t</b>		<i>c_oflag</i>	Output modes.
<b>tcflag_t</b>		<i>c_cflag</i>	Control modes.
<b>tcflag_t</b>		<i>c_lflag</i>	Local modes.
<b>cc_t</b>	NCCS	<i>c_cc[]</i>	Control characters.

7377 The **tcflag\_t** and **cc\_t** types are defined in the [<termios.h>](#) header. They shall be unsigned  
 7378 integer types.

7379 **11.2.2 Input Modes**

7380 Values of the *c\_iflag* field describe the basic terminal input control, and are composed of the  
7381 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
7382 symbols in this table are defined in `<termios.h>`:

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

7396 In the context of asynchronous serial data transmission, a break condition shall be defined as a  
7397 sequence of zero-valued bits that continues for more than the time to send one byte. The entire  
7398 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a  
7399 time equivalent to more than one byte. In contexts other than asynchronous serial data  
7400 transmission, the definition of a break condition is implementation-defined.

7401 If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the  
7402 input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the  
7403 break condition shall flush the input and output queues, and if the terminal is the controlling  
7404 terminal of a foreground process group, the break condition shall generate a single SIGINT  
7405 signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break  
7406 condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

7407 If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

7408 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than  
7409 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is  
7410 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid  
7411 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff  
7412 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be  
7413 given to the application as a single byte 0x00.

7414 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking  
7415 shall be disabled, allowing output parity generation without input parity errors. Note that  
7416 whether input parity checking is enabled or disabled is independent of whether parity detection  
7417 is enabled or disabled (see [Section 11.2.4](#), on page 209). If parity detection is enabled but input  
7418 parity checking is disabled, the hardware to which the terminal is connected shall recognize the  
7419 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

7420 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits  
7421 shall be processed.

7422 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a  
7423 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a  
7424 received CR character shall be translated into an NL character.

7425 If IXANY is set, any input character shall restart output that has been suspended.

7426 If IXON is set, start/stop output control shall be enabled. A received STOP character shall  
7427 suspend output and a received START character shall restart output. When IXON is set, START  
7428 and STOP characters are not read, but merely perform flow control functions. When IXON is not  
7429 set, the START and STOP characters shall be read.

7430 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP  
7431 characters, which are intended to cause the terminal device to stop transmitting data, as needed  
7432 to prevent the input queue from overflowing and causing implementation-defined behavior,  
7433 and shall transmit START characters, which are intended to cause the terminal device to resume  
7434 transmitting data, as soon as the device can continue transmitting data without risk of  
7435 overflowing the input queue. The precise conditions under which STOP and START characters  
7436 are transmitted are implementation-defined.

7437 The initial input control value after *open()* is implementation-defined.

### 7438 11.2.3 Output Modes

7439 The *c\_oflag* field specifies the terminal interface's treatment of output, and is composed of the  
7440 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name  
7441 symbols in the following table are defined in `<termios.h>`:

	Mask Name	Description	
7442	OPOST	Perform output processing.	
7443	XSI	ONLCR	Map NL to CR-NL on output.
7444		OCRNL	Map CR to NL on output.
7445		ONOCR	No CR output at column 0.
7446		ONLRET	NL performs CR function.
7447		OFILL	Use fill characters for delay.
7448		OFDEL	Fill is DEL, else NUL.
7449		NLDLY	Select newline delays:
7450		NL0	Newline character type 0.
7451		NL1	Newline character type 1.
7452		CRDLY	Select carriage-return delays:
7453		CR0	Carriage-return delay type 0.
7454		CR1	Carriage-return delay type 1.
7455		CR2	Carriage-return delay type 2.
7456		CR3	Carriage-return delay type 3.
7457		TABDLY	Select horizontal-tab delays:
7458		TAB0	Horizontal-tab delay type 0.
7459		TAB1	Horizontal-tab delay type 1.
7460		TAB2	Horizontal-tab delay type 2.
7461		TAB3	Expand tabs to spaces.
7462		BSDLY	Select backspace delays:
7463		BS0	Backspace-delay type 0.
7464	BS1	Backspace-delay type 1.	
7465	VTDLY	Select vertical-tab delays:	
7466	VT0	Vertical-tab delay type 0.	
7467	VT1	Vertical-tab delay type 1.	
7468	FFDLY	Select form-feed delays:	
7469	FF0	Form-feed delay type 0.	
7470	FF1	Form-feed delay type 1.	
7471			

7472 If OPOST is set, output data shall be post-processed as described below, so that lines of text are  
7473 modified to appear appropriately on the terminal device; otherwise, characters shall be  
7474 transmitted without change.

7475 XSI If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is  
7476 set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character  
7477 shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is  
7478 assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays  
7479 specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed  
7480 function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the  
7481 CR character is actually transmitted.

7482 The delay bits specify how long transmission stops to allow for mechanical or other movement  
7483 when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If  
7484 OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful  
7485 for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character  
7486 shall be DEL; otherwise, NUL.

7487 If a <form-feed> or <vertical-tab> delay is specified, it shall last for about 2 seconds.

7488 Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be  
7489 used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

7490 Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be



7491 about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall  
7492 transmit two fill characters, and type 2 four fill characters.

7493 Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be  
7494 about 0.10 seconds. Type 3 specifies that <tab> characters shall be expanded into <space>  
7495 characters. If OFILL is set, two fill characters shall be transmitted for any delay.

7496 Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be  
7497 transmitted.

7498 The actual delays depend on line speed and system load.

7499 The initial output control value after *open()* is implementation-defined.

#### 7500 11.2.4 Control Modes

7501 The *c\_cflag* field describes the hardware control of the terminal, and is composed of the bitwise-  
7502 inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in  
7503 this table are defined in <[termios.h](#)>; not all values specified are required to be supported by the  
7504 underlying hardware (if any). If the terminal is a pseudo-terminal, it is unspecified whether non-  
7505 default values are unsupported, or are supported and emulated in software, or are handled by  
7506 *tcsetattr()*, *tcgetattr()*, and the *stty* utility as if they are supported but have no effect on the  
7507 behavior of the terminal interface.

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

7520 In addition, the input and output baud rates are stored in the **termios** structure. The symbols in  
7521 the following table are defined in <[termios.h](#)>. Not all values specified are required to be  
7522 supported by the underlying hardware (if any). For pseudo-terminals, the input and output  
7523 baud rates set in the **termios** structure need not affect the speed of data transmission through the  
7524 terminal interface.

7525 **Note:** The term “baud” is used historically here, but is not technically correct. This is properly “bits  
7526 per second”, which may not be the same as baud. However, the term is used because of the  
7527 historical usage and understanding.

7528  
7529  
7530  
7531  
7532  
7533  
7534  
7535  
7536

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

7537  
7538  
7539  
7540

The following functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*. The effects on the terminal device shall not become effective and not all errors need be detected until the *tcsetattr()* function is successfully called.

7541  
7542  
7543  
7544  
7545

The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read. CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used; otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

7546

If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

7547  
7548  
7549

If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity shall be used.

7550  
7551  
7552

If HUPCL is set, the modem control lines for the port shall be lowered when the last process with the port open closes the port or the process terminates. The modem connection shall be broken.

7553  
7554

If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines shall be monitored.

7555  
7556  
7557

Under normal circumstances, a call to the *open()* function shall wait for the modem connection to complete. However, if the O\_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set, the *open()* function shall return immediately without waiting for the connection.

7558  
7559  
7560  
7561

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate need not be set on the connection between that terminal and the machine to which it is directly connected.

7562

The initial hardware control value after *open()* is implementation-defined.

7563 **11.2.5 Local Modes**7564  
7565  
7566

The *c\_iflag* field of the argument structure is used to control various functions. It is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in [<termios.h>](#).

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

7577 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input  
7578 characters shall not be echoed.

7579 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if  
7580 possible, the last character in the current line from the display. If there is no character to erase, an  
7581 implementation may echo an indication that this was the case, or do nothing.

7582 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the  
7583 line from the display or shall echo the <newline> character after the KILL character.

7584 If ECHONL and ICANON are set, the <newline> character shall be echoed even if ECHO is not  
7585 set.

7586 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit  
7587 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as  
7588 described in [Section 11.1.6](#) (on page 202).

7589 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall  
7590 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired  
7591 between bytes. The time value represents tenths of a second. See [Section 11.1.7](#) (on page 202) for  
7592 more details.

7593 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is  
7594 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.  
7595 If IEXTEN is not set, implementation-defined functions shall not be recognized and the  
7596 corresponding input characters are processed as described for ICANON, ISIG, IXON, and  
7597 IXOFF.

7598 If ISIG is set, each input character shall be checked against the special control characters INTR,  
7599 QUIT, and SUSP. If an input character matches one of these control characters, the function  
7600 associated with that character shall be performed. If ISIG is not set, no checking shall be done.  
7601 Thus these special input functions are possible only if ISIG is set.

7602 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,  
7603 QUIT, and SUSP characters shall not be done.

7604 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to  
7605 write to its controlling terminal if it is not in the foreground process group for that terminal. This  
7606 signal, by default, stops the members of the process group. Otherwise, the output generated by  
7607 that process shall be output to the current output stream. If the writing process is ignoring the  
7608 SIGTTOU signal or the writing thread is blocking the SIGTTOU signal, the process is allowed to  
7609 produce output, and the SIGTTOU signal shall not be sent.

7610 The initial local control value after *open()* is implementation-defined.

7611 **11.2.6 Special Control Characters**

7612 The special control character values shall be defined by the array *c\_cc*. The subscript name and  
 7613 description for each element in both canonical and non-canonical modes are as follows:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

7628 The subscript values are unique, except that the VMIN and VTIME subscripts may have the  
 7629 same values as the VEOF and VEOL subscripts, respectively.

7630 Implementations that do not support changing the START and STOP characters may ignore the  
 7631 character values in the *c\_cc* array indexed by the VSTART and VSTOP subscripts when  
 7632 *tcsetattr()* is called, but shall return the value in use when *tcgetattr()* is called.

7633 The initial values of all control characters are implementation-defined.

7634 If the value of one of the changeable special control characters (see [Section 11.1.9](#), on page 203) is  
 7635 `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the  
 7636 disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special  
 7637 meaning for the VMIN and VTIME entries of the *c\_cc* array.

# Utility Conventions

## 7640 12.1 Utility Argument Syntax

7641 This section describes the argument syntax of the standard utilities and introduces terminology  
7642 used throughout POSIX.1-2024 for describing the arguments processed by the utilities.

7643 Within POSIX.1-2024, a special notation is used for describing the syntax of a utility's  
7644 arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated  
7645 by this example (see XCU Section 2.9.1, on page 2500):

```
7646 utility_name [-a] [-b] [-c option_argument]
7647             [-d|-e] [-f[option_argument]] [operand...]
```

7648 The notation used for the SYNOPSIS sections imposes requirements on the implementors of the  
7649 standard utilities and provides a simple reference for the application developer or system user.

- 7650 1. The utility in the example is named *utility\_name*. It is followed by options, option-  
7651 arguments, and operands. The arguments that consist of <hyphen-minus> characters  
7652 and single letters or digits, such as 'a', are known as "options" (or, historically, "flags").  
7653 Certain options are followed by an "option-argument", as shown with [-c  
7654 *option\_argument*]. The arguments following the last options and option-arguments are  
7655 named "operands".
- 7656 2. Option-arguments are shown separated from their options by <blank> characters, except  
7657 when the option-argument is enclosed in the '[' and ']' notation to indicate that it is  
7658 optional. This reflects the situation in which an optional option-argument (if present) is  
7659 included within the same argument string as the option; for a mandatory option-  
7660 argument, it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page  
7661 215) require that the option be a separate argument from its option-argument and that  
7662 option-arguments not be optional, but there are some exceptions in POSIX.1-2024 to  
7663 provide for continued operation of historical applications:
  - 7664 a. If the SYNOPSIS of a standard utility shows an option with a mandatory option-  
7665 argument (as with [-c *option\_argument*] in the example), a conforming application  
7666 shall use separate arguments for that option and its option-argument. However, a  
7667 conforming implementation shall also permit applications to specify the option  
7668 and option-argument in the same argument string without intervening <blank>  
7669 characters.
  - 7670 b. If the SYNOPSIS shows an optional option-argument (as with  
7671 [-f[*option\_argument*]] in the example), a conforming application shall place any  
7672 option-argument for that option directly adjacent to the option in the same  
7673 argument string, without intervening <blank> characters. If the utility receives an  
7674 argument containing only the option, it shall behave as specified in its description  
7675 for an omitted option-argument; it shall not treat the next argument (if any) as the  
7676 option-argument for that option.

- 7677 3. Options are usually listed in alphabetical order unless this would make the utility  
7678 description more confusing. There are no implied relationships between the options  
7679 based upon the order in which they appear, unless otherwise stated in the OPTIONS  
7680 section, or unless the exception in Guideline 11 of [Section 12.2](#) (on page 215) applies. If an  
7681 option that does not have option-arguments is repeated, the results are undefined, unless  
7682 otherwise stated.
- 7683 4. Frequently, names of parameters that require substitution by actual values are shown  
7684 with embedded <underscore> characters. Alternatively, parameters are shown as follows:  
7685  
`<parameter name>`  
7686  
The angle brackets are used for the symbolic grouping of a phrase representing a single  
7687 parameter and conforming applications shall not include them in data submitted to the  
7688 utility.
- 7689 5. When a utility has only a few permissible options, they are sometimes shown  
7690 individually, as in the example. Utilities with many flags generally show all of the  
7691 individual flags (that do not take option-arguments) grouped, as in:  
7692  
`utility_name [-abcDxyz] [-p arg] [operand]`  
7693  
Utilities with very complex arguments may be shown as follows:  
7694  
`utility_name [options] [operands]`
- 7695 6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a  
7696 numeric value:  
7697  
  - The number is interpreted as a decimal integer.
  - Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric values.
  - When the utility description states that it accepts negative numbers as operands or option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are syntactically recognized as numeric values.
  - When the utility description states that the number is a file size-related value (such as a file size or offset, line number, or block count), numerals in the range 0 to the maximum file size supported by the implementation are syntactically recognized as numeric values (see [XCU Section 1.5](#), on page 2469). Where negative values are permitted, any value in the range -(maximum file size) to the maximum file size is accepted.
  - Ranges greater than those listed here are allowed.
- 7710 This does not mean that all numbers within the allowable range are necessarily  
7711 semantically correct. A standard utility that accepts an option-argument or operand that  
7712 is to be interpreted as a number, and for which a range of values smaller than that shown  
7713 above is permitted by the POSIX.1-2024, describes that smaller range along with the  
7714 description of the option-argument or operand. If an error is generated, the utility's  
7715 diagnostic message shall indicate that the value is out of the supported range, not that it  
7716 is syntactically incorrect.
- 7717 7. Arguments or option-arguments enclosed in the '[' and ']' notation are optional and  
7718 can be omitted. Conforming applications shall not include the '[' and ']' symbols in  
7719 data submitted to the utility.

7720 8. Arguments separated by the '|' (<vertical-line>) bar notation are mutually-exclusive.  
 7721 Conforming applications shall not include the '|' symbol in data submitted to the utility.  
 7722 Alternatively, mutually-exclusive options and operands may be listed with multiple  
 7723 synopsis lines.

7724 For example:

```
7725 utility_name -d [-a] [-c option_argument] [operand...]
```

```
7726 utility_name [-a] [-b] [operand...]
```

7727 When multiple synopsis lines are given for a utility, it is an indication that the utility has  
 7728 mutually-exclusive arguments. These mutually-exclusive arguments alter the  
 7729 functionality of the utility so that only certain other arguments are valid in combination  
 7730 with one of the mutually-exclusive arguments. Only one of the mutually-exclusive  
 7731 arguments is allowed for invocation of the utility. Unless otherwise stated in an  
 7732 accompanying OPTIONS section, the relationships between arguments depicted in the  
 7733 SYNOPSIS sections are mandatory requirements placed on conforming applications. The  
 7734 use of conflicting mutually-exclusive arguments produces undefined results, unless a  
 7735 utility description specifies otherwise. When an option is shown without the '[' and  
 7736 ']' brackets, it means that option is required for that version of the SYNOPSIS. However,  
 7737 it is not required to be the first argument, as shown in the example above, unless  
 7738 otherwise stated.

7739 9. Ellipses ("...") are used to denote that one or more occurrences of an operand are  
 7740 allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero  
 7741 or more options or operands can be specified. The form:

```
7742 utility_name [-g option_argument]... [operand...]
```

7743 indicates that multiple occurrences of the option and its option-argument preceding the  
 7744 ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See  
 7745 also Guideline 11 in [Section 12.2](#).)

7746 The form:

```
7747 utility_name -f option_argument [-f option_argument]... [operand...]
```

7748 indicates that the -f option is required to appear at least once and may appear multiple  
 7749 times.

7750 10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities  
 7751 volume of POSIX.1-2024, the indented lines following the initial line are continuation  
 7752 lines. An actual use of the command would appear on a single logical line.

## 7753 12.2 Utility Syntax Guidelines

7754 The following guidelines are established for the naming of utilities and for the specification of  
 7755 options, option-arguments, and operands. The *getopt()* function in the System Interfaces  
 7756 volume of POSIX.1-2024 assists utilities in handling options and operands that conform to these  
 7757 guidelines.

7758 Operands and option-arguments can contain characters not specified in the portable character  
 7759 set.

7760 The guidelines are intended to provide guidance to the authors of future utilities, such as those  
 7761 written specific to a local system or that are components of a larger application. Some of the  
 7762 standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections  
 7763 describe the deviations.

- 7764       **Guideline 1:**     Utility names should be between two and nine characters, inclusive.
- 7765       **Guideline 2:**     Utility names should include lowercase letters (the **lower** character  
7766                           classification) and digits only from the portable character set.
- 7767       **Guideline 3:**     Each option name should be a single alphanumeric character (the **alnum**  
7768                           character classification) from the portable character set. The **-W** (capital-W)  
7769                           option shall be reserved for vendor options.
- 7770                           Multi-digit options should not be allowed.
- 7771       **Guideline 4:**     All options should be preceded by the '-' delimiter character.
- 7772       **Guideline 5:**     One or more options without option-arguments, followed by at most one  
7773                           option that takes an option-argument, should be accepted when grouped  
7774                           behind one '-' delimiter.
- 7775       **Guideline 6:**     Each option and option-argument should be a separate argument, except as  
7776                           noted in [Section 12.1](#) (on page 213), item (2).
- 7777       **Guideline 7:**     Option-arguments should not be optional.
- 7778       **Guideline 8:**     When multiple option-arguments are specified to follow a single option, they  
7779                           should be presented as a single argument, using <comma> characters within  
7780                           that argument or <blank> characters within that argument to separate them.
- 7781       **Guideline 9:**     All options should precede operands on the command line.
- 7782       **Guideline 10:**    The first -- argument that is not an option-argument should be accepted as a  
7783                           delimiter indicating the end of options. Any following arguments should be  
7784                           treated as operands, even if they begin with the '-' character.
- 7785       **Guideline 11:**    The order of different options relative to one another should not matter, unless  
7786                           the options are documented as mutually-exclusive and such an option is  
7787                           documented to override any incompatible options preceding it. If an option  
7788                           that has option-arguments is repeated, the option and option-argument  
7789                           combinations should be interpreted in the order specified on the command  
7790                           line.
- 7791       **Guideline 12:**    The order of operands may matter and position-related interpretations should  
7792                           be determined on a utility-specific basis.
- 7793       **Guideline 13:**    For utilities that use operands to represent files to be opened for either reading  
7794                           or writing, the '-' operand should be used to mean only standard input (or  
7795                           standard output when it is clear from context that an output file is being  
7796                           specified) or a file named -.
- 7797       **Guideline 14:**    If an argument can be identified according to Guidelines 3 through 10 as an  
7798                           option, or as a group of options without option-arguments behind one '-'  
7799                           delimiter, then it should be treated as such.
- 7800       The utilities in the Shell and Utilities volume of POSIX.1-2024 that claim conformance to these  
7801       guidelines shall conform completely to these guidelines as if these guidelines contained the term  
7802       "shall" instead of "should". On some implementations, the utilities accept usage in violation of  
7803       these guidelines for backwards-compatibility as well as accepting the required form.
- 7804       Where a utility described in the Shell and Utilities volume of POSIX.1-2024 as conforming to  
7805       these guidelines is required to accept, or not to accept, the operand '-' to mean standard input  
7806       or output, this usage is explained in the OPERANDS section. Otherwise, if such a utility uses  
7807       operands to represent files, it is implementation-defined whether the operand '-' stands for  
7808       standard input (or standard output), or for a file named -.



7809           It is recommended that all future utilities and applications use these guidelines to enhance user  
7810 portability. The fact that some historical utilities could not be changed (to avoid breaking  
7811 existing applications) should not deter this future goal.



7812

Chapter 13

7813

## *Namespace and Future Directions*

7814

In order to prevent future versions of this standard from introducing features that could cause older applications to fail, the prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by this standard for any name (e.g. function, variable, utility, etc.). Neither implementations nor applications shall introduce objects in this namespace.

7815

7816

7817



# Headers

7820 This chapter describes the contents of headers.

7821 Headers contain function prototypes, the definition of symbolic constants, common structures,  
7822 preprocessor macros, and defined types. Each function in the System Interfaces volume of  
7823 POSIX.1-2024 specifies the headers that an application shall include in order to use that function.  
7824 In most cases, only one header is required. These headers are present on an application  
7825 development system; they need not be present on the target execution system.

## 7826 **Format of Entries**

7827 The entries in this chapter are based on a common format as follows. The only sections relating  
7828 to conformance are the SYNOPSIS and DESCRIPTION.

### 7829 **NAME**

7830 This section gives the name or names of the entry and briefly states its purpose.

### 7831 **SYNOPSIS**

7832 This section summarizes the use of the entry being described.

### 7833 **DESCRIPTION**

7834 This section describes the functionality of the header.

### 7835 **APPLICATION USAGE**

7836 This section is informative. This section gives warnings and advice to application  
7837 developers about the entry. In the event of conflict between warnings and advice and a  
7838 normative part of this volume of POSIX.1-2024, the normative material is to be taken as  
7839 correct.

### 7840 **RATIONALE**

7841 This section is informative. This section contains historical information concerning the  
7842 contents of this volume of POSIX.1-2024 and why features were included or discarded  
7843 by the standard developers.

### 7844 **FUTURE DIRECTIONS**

7845 This section is informative. This section provides comments which should be used as a  
7846 guide to current thinking; there is not necessarily a commitment to adopt these future  
7847 directions.

### 7848 **SEE ALSO**

7849 This section is informative. This section gives references to related information.

### 7850 **CHANGE HISTORY**

7851 This section is informative. This section shows the derivation of the entry and any  
7852 significant changes that have been made to it.

7853 **NAME**

7854 aio.h — asynchronous input and output

7855 **SYNOPSIS**

7856 #include &lt;aio.h&gt;

7857 **DESCRIPTION**7858 The **<aio.h>** header shall define the **aio\_cb** structure, which shall include at least the following  
7859 members:

7860	int	aio_fildes	File descriptor.
7861	off_t	aio_offset	File offset.
7862	volatile void *	aio_buf	Location of buffer.
7863	size_t	aio_nbytes	Length of transfer.
7864	int	aio_reqprio	Request priority offset.
7865	struct sigevent	aio_sigevent	Signal number and value.
7866	int	aio_lio_opcode	Operation to be performed.

7867 The **<aio.h>** header shall define the **off\_t**, **pthread\_attr\_t**, **size\_t**, and **ssize\_t** types as described  
7868 in **<sys/types.h>**.7869 The **<aio.h>** header shall define the **struct timespec** structure as described in **<time.h>**.7870 The **<aio.h>** header shall define the **sigevent** structure and **sigval** union as described in  
7871 **<signal.h>**.7872 The **<aio.h>** header shall define the following symbolic constants:7873 **AIO\_ALLDONE** A return value indicating that none of the requested operations could be  
7874 canceled since they are already complete.7875 **AIO\_CANCELED** A return value indicating that all requested operations have been  
7876 canceled.7877 **AIO\_NOTCANCELED**  
7878 A return value indicating that some of the requested operations could not  
7879 be canceled since they are in progress.7880 **LIO\_NOP** A *lio\_listio()* element operation option indicating that no transfer is  
7881 requested.7882 **LIO\_NOWAIT** A *lio\_listio()* synchronization operation indicating that the calling thread  
7883 is to continue execution while the *lio\_listio()* operation is being  
7884 performed, and no notification is given when the operation is complete.7885 **LIO\_READ** A *lio\_listio()* element operation option requesting a read.7886 **LIO\_WAIT** A *lio\_listio()* synchronization operation indicating that the calling thread  
7887 is to suspend until the *lio\_listio()* operation is complete.7888 **LIO\_WRITE** A *lio\_listio()* element operation option requesting a write.7889 The following shall be declared as functions and may also be defined as macros. Function  
7890 prototypes shall be provided.

7891	int	aio_cancel(int, struct aio_cb *);
7892	int	aio_error(const struct aio_cb *);
7893	FSC SIO int	aio_fsync(int, struct aio_cb *);
7894	int	aio_read(struct aio_cb *);
7895	ssize_t	aio_return(struct aio_cb *);
7896	int	aio_suspend(const struct aio_cb *const [], int,

```

7897         const struct timespec *);
7898     int     aio_write(struct aiocb *);
7899     int     lio_listio(int, struct aiocb *restrict const [restrict], int,
7900                struct sigevent *restrict);

```

7901 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>, <signal.h>, and <time.h>.

#### 7903 APPLICATION USAGE

7904 None.

#### 7905 RATIONALE

7906 None.

#### 7907 FUTURE DIRECTIONS

7908 None.

#### 7909 SEE ALSO

7910 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>

7911 XSH *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*, *aio\_return()*, *aio\_suspend()*, *aio\_write()*,  
7912 *fsync()*, *lio\_listio()*, *lseek()*, *read()*, *write()*

#### 7913 CHANGE HISTORY

7914 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 7915 Issue 6

7916 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7917 The description of the constants is expanded.

7918 The **restrict** keyword is added to the prototype for *lio\_listio()*.

#### 7919 Issue 7

7920 The <aio.h> header is moved from the Asynchronous Input and Output option to the Base.

7921 This reference page is clarified with respect to macros and symbolic constants, and type and structure declarations are added.

7923 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0038 [98] is applied.

7924 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0058 [579] is applied.

**7925 NAME**

7926           arpa/inet.h — definitions for internet operations

**7927 SYNOPSIS**

7928           #include <arpa/inet.h>

**7929 DESCRIPTION**

7930           The **<arpa/inet.h>** header shall define the **in\_port\_t** and **in\_addr\_t** types as described in **<netinet/in.h>** and the **socklen\_t** type as defined in **<sys/socket.h>**.

7932           The **<arpa/inet.h>** header shall define the **in\_addr** structure as described in **<netinet/in.h>**.

7933 IP6        The **<arpa/inet.h>** header shall define the **INET\_ADDRSTRLEN** and **INET6\_ADDRSTRLEN** macros as described in **<netinet/in.h>**.

7935           The following shall be declared as functions, or defined as macros, or both. If functions are declared, function prototypes shall be provided.

```
7937           uint32_t htonl(uint32_t);
7938           uint16_t htons(uint16_t);
7939           uint32_t ntohl(uint32_t);
7940           uint16_t ntohs(uint16_t);
```

7941           The **<arpa/inet.h>** header shall define the **uint32\_t** and **uint16\_t** types as described in **<inttypes.h>**.

7943           The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
7945 OB        in_addr_t     inet_addr(const char *);
7946           char         *inet_ntoa(struct in_addr);
7947           const char   *inet_ntop(int, const void *restrict, char *restrict,
7948                         socklen_t);
7949           int          inet_pton(int, const char *restrict, void *restrict);
```

7950           Inclusion of the **<arpa/inet.h>** header may also make visible all symbols from **<netinet/in.h>** and **<inttypes.h>**.

**7952 APPLICATION USAGE**

7953           None.

**7954 RATIONALE**

7955           None.

**7956 FUTURE DIRECTIONS**

7957           None.

**7958 SEE ALSO**

7959           **<endian.h>**, **<inttypes.h>**, **<netinet/in.h>**

7960           XSH *htonl()*, *inet\_addr()*, *inet\_ntop()*

**7961 CHANGE HISTORY**

7962           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7963           The **restrict** keyword is added to the prototypes for *inet\_ntop()* and *inet\_pton()*.

**7964 Issue 7**

7965           SD5-XBD-ERN-6 is applied.



7966 **Issue 8**

7967 Austin Group Defect 162 is applied, adding <endian.h> to the SEE ALSO section.

7968 Austin Group Defects 1101 and 1102 are applied, marking *inet\_addr()* and *inet\_ntoa()* as  
7969 obsolescent.

7970 Austin Group Defect 1290 is applied, adding a requirement for <arpa/inet.h> to define the  
7971 **socklen\_t** type.

7972 **NAME**7973 `assert.h` — verify program assertion7974 **SYNOPSIS**7975 `#include <assert.h>`7976 **DESCRIPTION**

7977 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
7978 conflict between the requirements described here and the ISO C standard is unintentional. This  
7979 volume of POSIX.1-2024 defers to the ISO C standard.

7980 The **<assert.h>** header shall define the `assert()` macro. It refers to the macro `NDEBUG` which is  
7981 not defined in the header. If `NDEBUG` is defined as a macro name before the inclusion of this  
7982 header, the `assert()` macro shall be defined simply as:

7983 

```
#define assert(ignore) ((void) 0)
```

7984 Otherwise, the macro behaves as described in `assert()`.

7985 The `assert()` macro shall be redefined according to the current state of `NDEBUG` each time  
7986 **<assert.h>** is included.

7987 The `assert()` macro shall be implemented as a macro, not as a function. If the macro definition is  
7988 suppressed in order to access an actual function, the behavior is undefined.

7989 **APPLICATION USAGE**

7990 None.

7991 **RATIONALE**

7992 None.

7993 **FUTURE DIRECTIONS**

7994 None.

7995 **SEE ALSO**7996 XSH [assert\(\)](#)7997 **CHANGE HISTORY**

7998 First released in Issue 1. Derived from Issue 1 of the SVID.

7999 **Issue 6**

8000 The definition of the `assert()` macro is changed for alignment with the ISO/IEC 9899:1999  
8001 standard.

8002 **NAME**  
 8003 complex.h — complex arithmetic

8004 **SYNOPSIS**  
 8005 #include <complex.h>

8006 **DESCRIPTION**

8007 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 8008 conflict between the requirements described here and the ISO C standard is unintentional. This  
 8009 volume of POSIX.1-2024 defers to the ISO C standard.

8010 Implementations shall not define the macro `__STDC_NO_COMPLEX__`, except for profile  
 8011 implementations that define `_POSIX_SUBPROFILE` (see [Section 2.1.5.1](#), on page 20) in  
 8012 `<unistd.h>`, which may define `__STDC_NO_COMPLEX__` and, if they do so, need not provide  
 8013 this header nor support any of its facilities.

8014 The `<complex.h>` header shall define the following macros:

- 8015 `complex` Expands to `_Complex`.
- 8016 `_Complex_I` Expands to a constant expression of type `const float _Complex`, with the value  
 8017 of the imaginary unit (that is, a number  $i$  such that  $i^2=-1$ ).
- 8018 `imaginary` Expands to `_Imaginary`.
- 8019 `_Imaginary_I` Expands to a constant expression of type `const float _Imaginary` with the  
 8020 value of the imaginary unit.
- 8021 `I` Expands to either `_Imaginary_I` or `_Complex_I`. If `_Imaginary_I` is not defined,  
 8022 `I` expands to `_Complex_I`.

8023 The macros `imaginary` and `_Imaginary_I` shall be defined if and only if the implementation  
 8024 MXC supports imaginary types. Implementations that support the IEC 60559 Complex Floating-Point  
 8025 option shall define the macros `imaginary` and `_Imaginary_I`, and the macro `I` shall expand to  
 8026 `_Imaginary_I`.

8027 An application may undefine and then, perhaps, redefine the `complex`, `imaginary`, and `I` macros.

8028 The following shall be defined as macros.

```
8029 double complex      Cmplx(double x, double y);
8030 float complex      Cmplxf(float x, float y);
8031 long double complex Cmplxl(long double x, long double y);
```

8032 The following shall be declared as functions and may also be defined as macros. Function  
 8033 prototypes shall be provided.

```
8034 double      cabs(double complex);
8035 float      cabsf(float complex);
8036 long double cabsl(long double complex);
8037 double complex cacos(double complex);
8038 float complex cacosf(float complex);
8039 double complex cacosh(double complex);
8040 float complex cacoshf(float complex);
8041 long double complex cacoshl(long double complex);
8042 long double complex cacosl(long double complex);
8043 double      carg(double complex);
8044 float      cargf(float complex);
8045 long double cargl(long double complex);
8046 double complex casin(double complex);
```

```
8047     float complex      casinl(float complex);
8048     double complex     casinh(double complex);
8049     float complex      casinhf(float complex);
8050     long double complex casinhl(long double complex);
8051     long double complex casinl(long double complex);
8052     double complex     catan(double complex);
8053     float complex      catanf(float complex);
8054     double complex     catanh(double complex);
8055     float complex      catanhf(float complex);
8056     long double complex catanhl(long double complex);
8057     long double complex catanl(long double complex);
8058     double complex     ccos(double complex);
8059     float complex      ccosf(float complex);
8060     double complex     ccosh(double complex);
8061     float complex      ccoshf(float complex);
8062     long double complex ccoshl(long double complex);
8063     long double complex ccosl(long double complex);
8064     double complex     cexp(double complex);
8065     float complex      cexpf(float complex);
8066     long double complex cexpl(long double complex);
8067     double             cimag(double complex);
8068     float             cimagf(float complex);
8069     long double       cimagl(long double complex);
8070     double complex     clog(double complex);
8071     float complex      clogf(float complex);
8072     long double complex clogl(long double complex);
8073     double complex     conj(double complex);
8074     float complex      conjf(float complex);
8075     long double complex conjl(long double complex);
8076     double complex     cpow(double complex, double complex);
8077     float complex      cpowf(float complex, float complex);
8078     long double complex cpowl(long double complex, long double complex);
8079     double complex     cproj(double complex);
8080     float complex      cprojf(float complex);
8081     long double complex cprojl(long double complex);
8082     double             creal(double complex);
8083     float             crealf(float complex);
8084     long double       creall(long double complex);
8085     double complex     csin(double complex);
8086     float complex      csinf(float complex);
8087     double complex     csinh(double complex);
8088     float complex      csinhf(float complex);
8089     long double complex csinhl(long double complex);
8090     long double complex csinl(long double complex);
8091     double complex     csqrt(double complex);
8092     float complex      csqrtf(float complex);
8093     long double complex csqrtl(long double complex);
8094     double complex     ctan(double complex);
8095     float complex      ctanf(float complex);
8096     double complex     ctanh(double complex);
8097     float complex      ctanhf(float complex);
8098     long double complex ctanhl(long double complex);
```

8099           long double complex   ctanl(long double complex);

8100 **APPLICATION USAGE**

8101           The <complex.h> header is optional in the ISO C standard but is mandated by POSIX.1-2024.  
8102           Note however that subprofiles can choose to make this header optional (see Section 2.1.5.1, on  
8103           page 20), and therefore application portability to subprofile implementations would benefit from  
8104           checking whether `__STDC_NO_COMPLEX__` is defined before inclusion of <complex.h>.

8105           Values are interpreted as radians, not degrees.

8106 **RATIONALE**

8107           The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the  
8108           identifier *i* for other purposes. The application can use a different identifier, say *j*, for the  
8109           imaginary unit by following the inclusion of the <complex.h> header with:

```
8110           #undef I
8111           #define j _Imaginary_I
```

8112           An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a  
8113           sufficiently convenient and more generally useful notation for imaginary terms. The  
8114           corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or  
8115           notational convenience will not result in widening types.

8116           On systems with imaginary types, the application has the ability to control whether use of the  
8117           macro `I` introduces an imaginary type, by explicitly defining `I` to be `_Imaginary_I` or `_Complex_I`.  
8118           Disallowing imaginary types is useful for some applications intended to run on  
8119           implementations without support for such types.

8120           The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

8121           The *cis*(*x*) function ( $\cos(x) + I\sin(x)$ ) was considered but rejected because its implementation is  
8122           easy and straightforward, even though some implementations could compute sine and cosine  
8123           more efficiently in tandem.

8124 **FUTURE DIRECTIONS**

8125           The following function names and the same names suffixed with *f* or *l* are reserved for future  
8126           use, and may be added to the declarations in the <complex.h> header.

```
8127           cerf()    cexpm1()  clog2()
8128           cerfc()  clog10()  clgamma()
8129           cexp2()  clog1p()  ctgamma()
```

8130 **SEE ALSO**

8131           XSH *CMPLX*(), *cabs*(), *cacos*(), *cacosh*(), *carg*(), *casin*(), *casinh*(), *catan*(), *catanh*(), *ccos*(), *ccosh*(),  
8132           *cexp*(), *cimag*(), *clog*(), *conj*(), *cpow*(), *cproj*(), *creal*(), *csin*(), *csinh*(), *csqrt*(), *ctan*(), *ctanh*()

8133 **CHANGE HISTORY**

8134           First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8135 **Issue 8**

8136           Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.

8137 **NAME**

8138 cpio.h — cpio archive values

8139 **SYNOPSIS**

8140 #include &lt;cpio.h&gt;

8141 **DESCRIPTION**8142 The **<cpio.h>** header shall define the symbolic constants needed by the *c\_mode* field of the *cpio*  
8143 archive format, with the names and values given in the following table:

8144	Name	Description	Value (Octal)
8145	C_IRUSR	Read by owner.	0000400
8146	C_IWUSR	Write by owner.	0000200
8147	C_IXUSR	Execute by owner.	0000100
8148	C_IRGRP	Read by group.	0000040
8149	C_IWGRP	Write by group.	0000020
8150	C_IXGRP	Execute by group.	0000010
8151	C_IROTH	Read by others.	0000004
8152	C_IWOTH	Write by others.	0000002
8153	C_IXOTH	Execute by others.	0000001
8154	C_ISUID	Set user ID.	0004000
8155	C_ISGID	Set group ID.	0002000
8156	C_ISVTX	On directories, restricted deletion flag.	0001000
8157	C_ISDIR	Directory.	0040000
8158	C_ISFIFO	FIFO.	0010000
8159	C_ISREG	Regular file.	0100000
8160	C_ISBLK	Block special.	0060000
8161	C_ISCHR	Character special.	0020000
8162	C_ISCTG	Reserved.	0110000
8163	C_ISLNK	Symbolic link.	0120000
8164	C_ISSOCK	Socket.	0140000

8165 The **<cpio.h>** header shall define the following symbolic constant as a string:

8166 MAGIC "070707"

8167 **APPLICATION USAGE**

8168 None.

8169 **RATIONALE**

8170 None.

8171 **FUTURE DIRECTIONS**

8172 None.

8173 **SEE ALSO**8174 XCU *pax*8175 **CHANGE HISTORY**8176 First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988  
8177 standard.8178 **Issue 6**8179 The SEE ALSO is updated to refer to *pax*.

8180 **Issue 7**

8181 The <cpio.h> header is moved from the XSI option to the Base.

8182 This reference page is clarified with respect to macros and symbolic constants.

8183 **NAME**8184 `ctype.h` — character types8185 **SYNOPSIS**8186 `#include <ctype.h>`8187 **DESCRIPTION**

8188 CX Some of the functionality described on this reference page extends the ISO C standard.  
8189 Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 496) to  
8190 enable the visibility of these symbols in this header.

8191 The **<ctype.h>** header shall define the **locale\_t** type as described in **<locale.h>**, representing a  
8192 locale object.

8193 The following shall be declared as functions and may also be defined as macros. Function  
8194 prototypes shall be provided for use with ISO C standard compilers.

```
8195 int isalnum(int);  
8196 CX int isalnum_l(int, locale_t);  
8197 int isalpha(int);  
8198 CX int isalpha_l(int, locale_t);  
8199 int isblank(int);  
8200 CX int isblank_l(int, locale_t);  
8201 int iscntrl(int);  
8202 CX int iscntrl_l(int, locale_t);  
8203 int isdigit(int);  
8204 CX int isdigit_l(int, locale_t);  
8205 int isgraph(int);  
8206 CX int isgraph_l(int, locale_t);  
8207 int islower(int);  
8208 CX int islower_l(int, locale_t);  
8209 int isprint(int);  
8210 CX int isprint_l(int, locale_t);  
8211 int ispunct(int);  
8212 CX int ispunct_l(int, locale_t);  
8213 int isspace(int);  
8214 CX int isspace_l(int, locale_t);  
8215 int isupper(int);  
8216 CX int isupper_l(int, locale_t);  
8217 int isxdigit(int);  
8218 CX int isxdigit_l(int, locale_t);  
8219 int tolower(int);  
8220 CX int tolower_l(int, locale_t);  
8221 int toupper(int);  
8222 CX int toupper_l(int, locale_t);
```



**8223 APPLICATION USAGE**

8224 None.

**8225 RATIONALE**

8226 None.

**8227 FUTURE DIRECTIONS**

8228 None.

**8229 SEE ALSO**

8230 <locale.h>

8231 XSH Section 2.2 (on page 496), *isalnum()*, *isalpha()*, *isblank()*, *isctrl()*, *isdigit()*, *isgraph()*,  
8232 *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*, *mbstowcs()*, *mbtowc()*,  
8233 *setlocale()*, *tolower()*, *toupper()*, *wcstombs()*, *wctomb()*

**8234 CHANGE HISTORY**

8235 First released in Issue 1. Derived from Issue 1 of the SVID.

**8236 Issue 6**

8237 Extensions beyond the ISO C standard are marked.

**8238 Issue 7**

8239 SD5-XBD-ERN-6 is applied, updating the wording regarding the function declarations for  
8240 consistency.

8241 The \*\_I() functions are added from The Open Group Technical Standard, 2006, Extended API Set  
8242 Part 4.

**8243 Issue 8**

8244 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

8245 **NAME**

8246 devctl.h — device control

8247 **SYNOPSIS**8248 DC `#include <devctl.h>`8249 **DESCRIPTION**8250 The **<devctl.h>** header shall define the **size\_t** type as described in **<sys/types.h>**.8251 The following shall be declared as a function and may also be defined as a macro. A function  
8252 prototype shall be provided.8253 `int posix_devctl(int, int, void *restrict, size_t, int *restrict);`8254 **APPLICATION USAGE**

8255 None.

8256 **RATIONALE**

8257 None.

8258 **FUTURE DIRECTIONS**

8259 None.

8260 **SEE ALSO**8261 [<sys/types.h>](#), [<termios.h>](#)8262 XSH *posix\_devctl()*8263 **CHANGE HISTORY**

8264 First released in Issue 8. Derived from POSIX.26.

8265 **NAME**

8266 dirent.h — format of directory entries

8267 **SYNOPSIS**

8268 #include <dirent.h>

8269 **DESCRIPTION**

8270 The internal format of directories is unspecified.

8271 The <dirent.h> header shall define the following type:

8272 **DIR** A type representing a directory stream. The **DIR** type may be an incomplete type.

8273 It shall also define the structure **dirent** which shall include the following members:

8274 ino\_t d\_ino File serial number.  
8275 char d\_name[] Filename string of entry.

8276 and the structure **posix\_dent** which shall include the following members:

8277 ino\_t d\_ino File serial number.  
8278 reflen\_t d\_reflen Length of this entry, including trailing  
8279 padding if necessary. See *posix\_getdents()*.  
8280 unsigned char d\_type File type or unknown-file-type indication.  
8281 char d\_name[] Filename string of this entry.

8282 The array *d\_name* in each of these structures is of unspecified size, but shall contain a filename of  
8283 at most {NAME\_MAX} bytes followed by a terminating null byte.

8284 The <dirent.h> header shall define the **ino\_t**, **reflen\_t**, **size\_t**, and **ssize\_t** types as described in  
8285 <sys/types.h>.

8286 The <dirent.h> header shall define the following symbolic constants for the file types and  
8287 unknown-file-type indicator returned in the *d\_type* member of the **posix\_dent** structure. The  
8288 values shall be distinct and shall be suitable for use in **#if** preprocessing directives:

8289 DT\_BLK Block special.  
8290 DT\_CHR Character special.  
8291 DT\_DIR Directory.  
8292 DT\_FIFO FIFO special.  
8293 DT\_LNK Symbolic link.  
8294 DT\_REG Regular.  
8295 DT\_SOCKET Socket.  
8296 DT\_UNKNOWN  
8297 Unknown file type.

8298 **TYM** The implementation may implement message queues, semaphores, shared memory objects or  
8299 typed memory objects as distinct file types. The following macros shall be provided to represent  
8300 these types. The values shall be distinct from each other and from the above symbolic constants  
8301 beginning with DT\_, except when a distinct file type is not implemented, in which case the  
8302 corresponding constant shall have a value that is never returned in *d\_type* by *posix\_getdents()*.  
8303 The values shall be suitable for use in **#if** preprocessing directives:

8304 DT\_MQ Message queue.

```

8305         DT_SEM    Semaphore.
8306         DT_SHM    Shared memory object.
8307 TYM    DT_TMO    Typed memory object.
8308         The following shall be declared as functions and may also be defined as macros. Function
8309         prototypes shall be provided.
8310         int        alphasort(const struct dirent **, const struct dirent **);
8311         int        closedir(DIR *);
8312         int        dirfd(DIR *);
8313         DIR        *fdopendir(int);
8314         DIR        *opendir(const char *);
8315         ssize_t    posix_getdents(int, void *, size_t, int);
8316         struct dirent *readdir(DIR *);
8317 OB      int        readdir_r(DIR *restrict, struct dirent *restrict,
8318         struct dirent **restrict);
8319         void        rewinddir(DIR *);
8320         int        scandir(const char *, struct dirent ***,
8321         int (*)(const struct dirent *),
8322         int (*)(const struct dirent **),
8323         const struct dirent **);
8324 XSI     void        seekdir(DIR *, long);
8325         long       telldir(DIR *);

```

**APPLICATION USAGE**

None.

**RATIONALE**

Information similar to that in the **<dirent.h>** header is contained in a file **<sys/dir.h>** in 4.2 BSD and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of POSIX.1-2024 is **struct direct**. The filename was changed because the name **<sys/dir.h>** was also used in earlier implementations to refer to definitions related to the older access method; this produced name conflicts. The name of the structure was changed because this volume of POSIX.1-2024 does not completely define what is in the structure, so it could be different on some implementations from **struct direct**.

The **posix\_dent** structure was based on existing structures used by traditional *getdents()* functions, but the name was changed because the existing structures differed in name and in their members. Some used the **dirent** structure but this is not required to include a *d\_type* member, which is the main advantage of using *posix\_getdents()* over *readdir()*. The *d\_reclen* member was included, even though some implementations return fixed-length entries and therefore do not need it, as almost all existing code that used *getdents()* used *d\_reclen* to iterate through the returned entries. Implementations that return fixed-length entries can simply set *d\_reclen* to that length in *posix\_getdents()*. The type **reclen\_t** for *d\_reclen* was introduced, instead of using **unsigned short**, so as not to create a requirement that  $\{\text{NAME\_MAX}\}$  cannot be greater than (a value somewhat smaller than)  $\{\text{SHRT\_MAX}\}$ .

Implementations are encouraged to define a **DT\_FORCE\_TYPE** symbolic constant for use in the *flags* argument to *posix\_getdents()*. See the RATIONALE for *posix\_getdents()*.

The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:

```
sizeof(d_name)
```

8350 is incorrect; use:  
8351 `strlen(d_name)`  
8352 instead.

8353 The array of **char** *d\_name* cannot be assumed to have a fixed size. Implementations may define  
8354 the *d\_name* array in the **dirent** and **posix\_dent** structures to have size 1, or size greater than  
8355 {NAME\_MAX}, or use a flexible array member, but in all cases the actual number of characters  
8356 used for *d\_name* is at least the length of the filename string including the terminating NUL byte.

8357 **FUTURE DIRECTIONS**  
8358 A future version of this standard may add a DT\_FORCE\_TYPE symbolic constant for use as  
8359 described in the RATIONALE for *posix\_getdents()*.

8360 **SEE ALSO**  
8361 [<sys/types.h>](#)

8362 XSH *alphasort()*, *closedir()*, *dirfd()*, *fdopendir()*, *posix\_getdents()*, *readdir()*, *rewinddir()*, *seekdir()*,  
8363 *telldir()*

8364 **CHANGE HISTORY**  
8365 First released in Issue 2.

8366 **Issue 5**  
8367 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8368 **Issue 6**  
8369 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir\_r()*.  
8370 The **restrict** keyword is added to the prototype for *readdir\_r()*.

8371 **Issue 7**  
8372 The *alphasort()*, *dirfd()*, and *scandir()* functions are added from The Open Group Technical  
8373 Standard, 2006, Extended API Set Part 1.  
8374 The *fdopendir()* function is added from The Open Group Technical Standard, 2006, Extended API  
8375 Set Part 2.  
8376 Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying the definition of the **DIR**  
8377 type.  
8378 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0039 [291], XBD/TC1-2008/0040 [291],  
8379 XBD/TC1-2008/0041 [291], and XBD/TC1-2008/0042 [206] are applied.

8380 **Issue 8**  
8381 Austin Group Defect 696 is applied, making *readdir\_r()* obsolescent.  
8382 Austin Group Defect 697 is applied, adding *posix\_getdents()*.

8383 **NAME**8384 `dlfcn.h` — dynamic linking8385 **SYNOPSIS**8386 `#include <dlfcn.h>`8387 **DESCRIPTION**8388 The **<dlfcn.h>** header shall define the **Dl\_info\_t** structure type, which shall include at least the  
8389 following members:

8390	<code>const char *dli_fname</code>	Pathname of mapped object file.
8391	<code>void *dli_fbase</code>	Base of mapped address range.
8392	<code>const char *dli_sname</code>	Symbol name or null pointer.
8393	<code>void *dli_saddr</code>	Symbol address or null pointer.

8394 The **<dlfcn.h>** header shall define at least the following symbolic constants for use in the  
8395 construction of a `dlopen()` *mode* argument:

8396	<code>RTLD_LAZY</code>	Relocations are performed at an implementation-defined time.
8397	<code>RTLD_NOW</code>	Relocations are performed when the object is loaded.
8398	<code>RTLD_GLOBAL</code>	All symbols are available for relocation processing of other modules.
8399	<code>RTLD_LOCAL</code>	All symbols are not made available for relocation processing by other 8400 modules.

8401 The following shall be declared as functions and may also be defined as macros. Function  
8402 prototypes shall be provided.

```
8403 int dladdr(const void *restrict, Dl_info_t *restrict);
8404 int dlclose(void *);
8405 char *dlderror(void);
8406 void *dlopen(const char *, int);
8407 void *dlsym(void *restrict, const char *restrict);
```

8408 **APPLICATION USAGE**

8409 None.

8410 **RATIONALE**

8411 None.

8412 **FUTURE DIRECTIONS**

8413 None.

8414 **SEE ALSO**8415 XSH `dladdr()`, `dlclose()`, `dlderror()`, `dlopen()`, `dlsym()`8416 **CHANGE HISTORY**

8417 First released in Issue 5.

8418 **Issue 6**8419 The **restrict** keyword is added to the prototype for `dlsym()`.8420 **Issue 7**8421 The **<dlfcn.h>** header is moved from the XSI option to the Base.

8422 This reference page is clarified with respect to macros and symbolic constants.

8423 **Issue 8**  
8424

Austin Group Defect 993 is applied, adding *dladdr()*.

8425 **NAME**8426            **endian.h** — system endianness8427 **SYNOPSIS**

8428            #include &lt;endian.h&gt;

8429 **DESCRIPTION**8430            The **<endian.h>** header shall define at least the following macros for use in determining host  
8431            byte order for integer types.8432            **BYTE\_ORDER**    This macro shall have a value equal to one of the \*\_ENDIAN macros in this  
8433            header.8434            **LITTLE\_ENDIAN**8435                    If **BYTE\_ORDER == LITTLE\_ENDIAN**, the host byte order is from least  
8436                    significant to most significant.8437            **BIG\_ENDIAN**    If **BYTE\_ORDER == BIG\_ENDIAN**, the host byte order is from most  
8438                    significant to least significant.8439            These macros shall be suitable for use in **#if** preprocessing directives. The macros **BIG\_ENDIAN**  
8440            and **LITTLE\_ENDIAN** shall have distinct values. Implementations may define other macros  
8441            with the **\_ENDIAN** suffix.8442            The following shall be declared as functions, or defined as macros, or both. If functions are  
8443            declared, function prototypes shall be provided.

8444            uint16\_t    be16toh(uint16\_t);

8445            uint32\_t    be32toh(uint32\_t);

8446            uint64\_t    be64toh(uint64\_t);

8447            uint16\_t    htobe16(uint16\_t);

8448            uint32\_t    htobe32(uint32\_t);

8449            uint64\_t    htobe64(uint64\_t);

8450            uint16\_t    htole16(uint16\_t);

8451            uint32\_t    htole32(uint32\_t);

8452            uint64\_t    htole64(uint64\_t);

8453            uint16\_t    le16toh(uint16\_t);

8454            uint32\_t    le32toh(uint32\_t);

8455            uint64\_t    le64toh(uint64\_t);

8456            The **<endian.h>** header shall define the **uint16\_t**, **uint32\_t**, and **uint64\_t** types as described in  
8457            **<stdint.h>**.8458            Inclusion of the **<endian.h>** header may also make visible all symbols from **<stdint.h>**.8459 **APPLICATION USAGE**

8460            None.

8461 **RATIONALE**8462            Many implementations also include **PDP\_ENDIAN** to indicate a byte ordering where each pair  
8463            of bytes is swapped. If **BIG\_ENDIAN** is defined as 4321, **PDP\_ENDIAN** would be 3412.  
8464            However, this scheme is not universal, and derives its name from an obsolete processor.8465 **FUTURE DIRECTIONS**

8466            None.



8467 **SEE ALSO**

8468 [<stdint.h>](#)

8469 XSH [be16toh\(\)](#), [htonl\(\)](#), [swab\(\)](#)

8470 **CHANGE HISTORY**

8471 First released in Issue 8.

8472 **NAME**8473 `errno.h` — system error numbers8474 **SYNOPSIS**8475 `#include <errno.h>`8476 **DESCRIPTION**

8477 CX Some of the functionality described on this reference page extends the ISO C standard. Any  
8478 conflict between the requirements described here and the ISO C standard is unintentional. This  
8479 volume of POSIX.1-2024 defers to the ISO C standard.

8480 The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

8481 The **<errno.h>** header shall provide a definition for the macro *errno*, which shall expand to a  
8482 modifiable lvalue of type **int** and thread local storage duration. If the macro definition is  
8483 suppressed in order to access an actual object, or a program defines an identifier with the name  
8484 *errno*, the behavior is undefined.

8485 The **<errno.h>** header shall define the following macros which shall expand to integer constant  
8486 expressions with type **int**, distinct positive values (except as noted below), and which shall be  
8487 suitable for use in **#if** preprocessing directives:

8488	[E2BIG]	Argument list too long.
8489	[EACCES]	Permission denied.
8490	[EADDRINUSE]	Address in use.
8491	[EADDRNOTAVAIL]	Address not available.
8492	[EAFNOSUPPORT]	Address family not supported.
8493	[EAGAIN]	Resource unavailable, try again (may be the same value as
8494		[EWOULDBLOCK]).
8495	[EALREADY]	Connection already in progress.
8496	[EBADF]	Bad file descriptor.
8497	[EBADMSG]	Bad message.
8498	[EBUSY]	Device or resource busy.
8499	[ECANCELED]	Operation canceled.
8500	[ECHILD]	No child processes.
8501	[ECONNABORTED]	Connection aborted.
8502	[ECONNREFUSED]	Connection refused.
8503	[ECONNRESET]	Connection reset.
8504	[EDEADLK]	Resource deadlock would occur.
8505	[EDESTADDRREQ]	Destination address required.
8506	[EDOM]	Mathematics argument out of domain of function.
8507	[EDQUOT]	Reserved.
8508	[EEXIST]	File exists.

8509	[EFAULT]	Bad address.
8510	[EFBIG]	File too large.
8511	[EHOSTUNREACH]	Host is unreachable.
8512	[EIDRM]	Identifier removed.
8513	[EILSEQ]	Illegal byte sequence.
8514	[EINPROGRESS]	Operation in progress.
8515	[EINTR]	Interrupted function.
8516	[EINVAL]	Invalid argument.
8517	[EIO]	I/O error.
8518	[EISCONN]	Socket is connected.
8519	[EISDIR]	Is a directory.
8520	[ELOOP]	Too many levels of symbolic links.
8521	[EMFILE]	File descriptor value too large.
8522	[EMLINK]	Too many hard links.
8523	[EMSGSIZE]	Message too large.
8524	[EMULTIHOP]	Reserved.
8525	[ENAMETOOLONG]	Filename too long.
8526	[ENETDOWN]	Network is down.
8527	[ENETRESET]	Connection aborted by network.
8528	[ENETUNREACH]	Network unreachable.
8529	[ENFILE]	Too many files open in system.
8530	[ENOBUFS]	No buffer space available.
8531	[ENODEV]	No such device.
8532	[ENOENT]	No such file or directory.
8533	[ENOEXEC]	Executable file format error.
8534	[ENOLCK]	No locks available.
8535	[ENOLINK]	Reserved.
8536	[ENOMEM]	Not enough space.
8537	[ENOMSG]	No message of the desired type.
8538	[ENOPROTOOPT]	Protocol not available.
8539	[ENOSPC]	No space left on device.
8540	[ENOSYS]	Functionality not supported.
8541	[ENOTCONN]	The socket is not connected.

8542	[ENOTDIR]	Not a directory or a symbolic link to a directory.
8543	[ENOTEMPTY]	Directory not empty.
8544	[ENOTRECOVERABLE]	
8545		State not recoverable.
8546	[ENOTSOCK]	Not a socket.
8547	[ENOTSUP]	Not supported (may be the same value as [EOPNOTSUPP]).
8548	[ENOTTY]	Inappropriate I/O control operation.
8549	[ENXIO]	No such device or address.
8550	[EOPNOTSUPP]	Operation not supported on socket (may be the same value as
8551		[ENOTSUP]).
8552	[EOVERFLOW]	Value too large to be stored in data type.
8553	[EOWNERDEAD]	Previous owner died.
8554	[EPERM]	Operation not permitted.
8555	[EPIPE]	Broken pipe.
8556	[EPROTO]	Protocol error.
8557	[EPROTONOSUPPORT]	
8558		Protocol not supported.
8559	[EPROTOTYPE]	Protocol wrong type for socket.
8560	[ERANGE]	Result too large.
8561	[EROFS]	Read-only file system.
8562	[ESOCKTNOSUPPORT]	
8563		Socket type not supported.
8564	[ESPIPE]	Invalid seek.
8565	[ESRCH]	No such process.
8566	[ESTALE]	Reserved.
8567	[ETIMEDOUT]	Connection timed out.
8568	[ETXTBSY]	Text file busy.
8569	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
8570	[EXDEV]	Improper hard link.

8571 **APPLICATION USAGE**

8572 Additional error numbers may be defined on conforming systems; see the System Interfaces  
8573 volume of POSIX.1-2024.

8574 **RATIONALE**

8575 None.

8576 **FUTURE DIRECTIONS**

8577 None.

8578 **SEE ALSO**8579 XSH [Section 2.3](#) (on page 507)8580 **CHANGE HISTORY**

8581 First released in Issue 1. Derived from Issue 1 of the SVID.

8582 **Issue 5**

8583 Updated for alignment with the POSIX Realtime Extension.

8584 **Issue 6**8585 The following new requirements on POSIX implementations derive from alignment with the  
8586 Single UNIX Specification:

- 8587
- The majority of the error conditions previously marked as extensions are now mandatory,  
8588 except for the STREAMS-related error conditions.

8589 Values for *errno* are now required to be distinct positive values rather than non-zero values. This  
8590 change is for alignment with the ISO/IEC 9899: 1999 standard.8591 **Issue 7**8592 Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and  
8593 [EOPNOTSUPP] to be the same values.8594 The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group  
8595 Technical Standard, 2006, Extended API Set Part 2.

8596 Functionality relating to the XSI STREAMS option is marked obsolescent.

8597 Functionality relating to the Threads option is moved to the Base.

8598 This reference page is clarified with respect to macros and symbolic constants.

8599 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0043 [324] is applied.

8600 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0059 [496] is applied.

8601 **Issue 8**

8602 Austin Group Defect 1067 is applied, adding [ESOCKTNOSUPPORT].

8603 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899: 2018 standard.

8604 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

8605 Austin Group Defect 1380 is applied, changing the descriptions of [EMLINK] and [EXDEV].

8606 **NAME**8607 `fcntl.h` — file control options8608 **SYNOPSIS**8609 `#include <fcntl.h>`8610 **DESCRIPTION**8611 The **<fcntl.h>** header shall define the `f_owner_ex` structure, which shall include at least the  
8612 following members:8613 `int type` Discriminator for `pid`.  
8614 `pid_t pid` Process ID or process group ID.8615 The **<fcntl.h>** header shall define the `flock` structure describing a file lock. It shall include the  
8616 following members:8617 `short l_type` Type of lock; `F_RDLCK`, `F_WRLCK`, `F_UNLCK`.  
8618 `short l_whence` Flag for starting offset.  
8619 `off_t l_start` Relative offset in bytes.  
8620 `off_t l_len` Size; if 0 then until EOF.  
8621 `pid_t l_pid` For a process-owned file lock, ignored on input or the process ID of the  
8622 owning process on output; for an OFD-owned file lock, zero on input or  
8623 `(pid_t)-1` on output.8624 The **<fcntl.h>** header shall define the `mode_t`, `off_t`, and `pid_t` types as described in  
8625 **<sys/types.h>**.8626 The **<fcntl.h>** header shall define the following symbolic constants for the `cmd` argument used  
8627 by `fcntl()`. The values shall be unique and shall be suitable for use in `#if` preprocessing  
8628 directives.8629 `F_DUPFD` Duplicate file descriptor.  
8630 `F_DUPFD_CLOEXEC`  
8631 Duplicate file descriptor with the close-on-*exec* flag `FD_CLOEXEC` set.  
8632 `F_DUPFD_CLOFORK`  
8633 Duplicate file descriptor with the close-on-*fork* flag `FD_CLOFORK` set.  
8634 `F_GETFD` Get file descriptor flags.  
8635 `F_SETFD` Set file descriptor flags.  
8636 `F_GETFL` Get file status flags and file access modes.  
8637 `F_SETFL` Set file status flags.  
8638 `F_GETLK` Get information about file locks.  
8639 `F_SETLK` Set a process-owned file lock.  
8640 `F_SETLKW` Set a process-owned file lock; wait if blocked.  
8641 `F_OFD_GETLK` Get information about file locks.  
8642 `F_OFD_SETLK` Set an OFD-owned file lock.  
8643 `F_OFD_SETLKW`  
8644 Set an OFD-owned file lock; wait if blocked.  
8645 `F_GETOWN` Get process or process group ID to receive SIGURG signals, via `int` type.

8646 F\_GETOWN\_EX Get process or process group ID to receive SIGURG signals, via `pid_t` type.

8647 F\_SETOWN Set process or process group ID to receive SIGURG signals, via `int` type.

8648 F\_SETOWN\_EX Set process or process group ID to receive SIGURG signals, via `pid_t` type.

8649 The <fcntl.h> header shall define the following symbolic constants used for the `fcntl()` file  
 8650 descriptor flags. The values shall be bitwise-distinct and shall be suitable for use in `#if`  
 8651 preprocessing directives.

8652 SPN FD\_CLOEXEC Close the file descriptor upon execution of an `exec` family function and in the  
 8653 new process image created by `posix_spawn()` or `posix_spawnnp()`.

8654 FD\_CLOFORK Close the file descriptor in any child process created from a process that has  
 8655 the file descriptor open; that is, the child shall not inherit the file descriptor.

8656 The <fcntl.h> header shall also define the following symbolic constants for the `l_type` argument  
 8657 used for record locking with `fcntl()`. The values shall be unique and shall be suitable for use in  
 8658 `#if` preprocessing directives.

8659 F\_RDLCK Shared or read lock.

8660 F\_UNLCK Unlock.

8661 F\_WRLCK Exclusive or write lock.

8662 The <fcntl.h> header shall also define the following symbolic constants for the `type` member of  
 8663 the `f_owner_ex` structure. The values shall be unique.

8664 F\_OWNER\_PID The `pid` member of `f_owner_ex` holds a process ID.

8665 F\_OWNER\_PGRP  
 8666 The `pid` member of `f_owner_ex` holds a process group ID.

8667 The <fcntl.h> header shall define the values used for `l_whence`, `SEEK_SET`, `SEEK_CUR`, and  
 8668 `SEEK_END` as described in <stdio.h>.

8669 The <fcntl.h> header shall define the following symbolic constants as file creation flags for use  
 8670 in the `oflag` value to `open()` and `openat()`. The values shall be bitwise-distinct and shall be  
 8671 suitable for use in `#if` preprocessing directives.

8672 O\_CLOEXEC Atomically set the FD\_CLOEXEC flag on the new file descriptor.

8673 O\_CLOFORK Atomically set the FD\_CLOFORK flag on the new file descriptor.

8674 O\_CREAT Create file if it does not exist.

8675 O\_DIRECTORY Fail if file is a non-directory file.

8676 O\_EXCL Exclusive use flag.

8677 O\_NOCTTY Do not assign controlling terminal.

8678 O\_NOFOLLOW Do not follow symbolic links.

8679 O\_TRUNC Truncate flag.

8680 O\_TTY\_INIT Set the `termios` structure terminal parameters to a state that provides  
 8681 conforming behavior; see Section 11.2 (on page 205).

8682 The O\_TTY\_INIT flag can have the value zero and in this case it need not be bitwise-distinct  
 8683 from the other flags.

8684 The <fcntl.h> header shall define the following symbolic constants for use as file status flags for

8685 *open()*, *openat()*, and *fcntl()*. The values shall be suitable for use in **#if** preprocessing directives.

8686 **O\_APPEND** Set append mode.

8687 SIO **O\_DSYNC** Write according to synchronized I/O data integrity completion.

8688 **O\_NONBLOCK** Non-blocking mode.

8689 SIO **O\_RSYNC** Synchronized read I/O operations.

8690 **O\_SYNC** Write according to synchronized I/O file integrity completion.

8691 The **<fcntl.h>** header shall define the following symbolic constant for use as the mask for file  
8692 access modes. The value shall be suitable for use in **#if** preprocessing directives.

8693 **O\_ACCMODE** Mask for file access modes.

8694 The **<fcntl.h>** header shall define the following symbolic constants for use as the file access  
8695 modes for *open()*, *openat()*, and *fcntl()*. The values shall be unique, except that **O\_EXEC** and  
8696 **O\_SEARCH** may have equal values. The values shall be suitable for use in **#if** preprocessing  
8697 directives.

8698 **O\_EXEC** Open for execute only (non-directory files). The result is unspecified if this  
8699 flag is applied to a directory.

8700 **O\_RDONLY** Open for reading only.

8701 **O\_RDWR** Open for reading and writing.

8702 **O\_SEARCH** Open directory for search only. The result is unspecified if this flag is applied  
8703 to a non-directory file.

8704 **O\_WRONLY** Open for writing only.

8705 The **<fcntl.h>** header shall define the symbolic constants for file modes for use as values of  
8706 **mode\_t** as described in **<sys/stat.h>**.

8707 The **<fcntl.h>** header shall define the following symbolic constant as a special value used in  
8708 place of a file descriptor for the *\*at()* functions which take a directory file descriptor as a  
8709 parameter:

8710 **AT\_FDCWD** Use the current working directory to determine the target of relative file paths.

8711 The **<fcntl.h>** header shall define the following symbolic constant as a value for the *flag* used by  
8712 *faccessat()*:

8713 **AT\_EACCESS** Check access using effective user and group ID.

8714 The **<fcntl.h>** header shall define the following symbolic constant as a value for the *flag* used by  
8715 *fstatat()*, *fchmodat()*, *fchownat()*, and *utimensat()*:

8716 **AT\_SYMLINK\_NOFOLLOW**  
8717 Do not follow symbolic links.

8718 The **<fcntl.h>** header shall define the following symbolic constant as a value for the *flag* used by  
8719 *linkat()*:

8720 **AT\_SYMLINK\_FOLLOW**  
8721 Follow symbolic link.



8722 The <fcntl.h> header shall define the following symbolic constant as a value for the flag used by  
 8723 *unlinkat()*:

8724 AT\_REMOVEDIR  
 8725 Remove directory instead of file.

8726 ADV The <fcntl.h> header shall define the following symbolic constants for the *advice* argument used  
 8727 by *posix\_fadvise()*:

8728 POSIX\_FADV\_DONTNEED  
 8729 The application expects that it will not access the specified data in the near future.

8730 POSIX\_FADV\_NOREUSE  
 8731 The application expects to access the specified data once and then not reuse it thereafter.

8732 POSIX\_FADV\_NORMAL  
 8733 The application has no advice to give on its behavior with respect to the specified data. It is  
 8734 the default characteristic if no advice is given for an open file.

8735 POSIX\_FADV\_RANDOM  
 8736 The application expects to access the specified data in a random order.

8737 POSIX\_FADV\_SEQUENTIAL  
 8738 The application expects to access the specified data sequentially from lower offsets to higher  
 8739 offsets.

8740 POSIX\_FADV\_WILLNEED  
 8741 The application expects to access the specified data in the near future.

8742 The following shall be declared as functions and may also be defined as macros. Function  
 8743 prototypes shall be provided.

```
8744 int creat(const char *, mode_t);
8745 int fcntl(int, int, ...);
8746 int open(const char *, int, ...);
8747 int openat(int, const char *, int, ...);
8748 ADV int posix_fadvise(int, off_t, off_t, int);
8749 int posix_fallocate(int, off_t, off_t);
```

8750 Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and  
 8751 <unistd.h>.

8752 **APPLICATION USAGE**

8753 Although no existing implementation defines AT\_SYMLINK\_FOLLOW and  
 8754 AT\_SYMLINK\_NOFOLLOW as the same numeric value, POSIX.1-2024 does not prohibit that as  
 8755 the two constants are not used with the same interfaces.

8756 **RATIONALE**

8757 While many of the symbolic constants introduced in the <fcntl.h> header do not strictly need to  
 8758 be used in #if preprocessor directives, widespread historic practice has defined them as macros  
 8759 that are usable in such constructs, and examination of existing applications has shown that they  
 8760 are occasionally used in such a way. Therefore it was decided to retain this requirement on an  
 8761 implementation in POSIX.1-2024.

**8762 FUTURE DIRECTIONS**

8763 A future version of this standard may add an O\_NOCLOBBER file creation flag. See the  
8764 FUTURE DIRECTIONS section for *open()*.

**8765 SEE ALSO**

8766 [<stdio.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#), [<unistd.h>](#)

8767 XSH *creat()*, *exec*, *fcntl()*, *futimens()*, *open()*, *posix\_fadvise()*, *posix\_fallocate()*, *posix\_madvise()*

**8768 CHANGE HISTORY**

8769 First released in Issue 1. Derived from Issue 1 of the SVID.

**8770 Issue 5**

8771 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

**8772 Issue 6**

8773 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

8774 • O\_DSYNC and O\_RSYNC are marked as part of the Synchronized Input and Output  
8775 option.

8776 The following new requirements on POSIX implementations derive from alignment with the  
8777 Single UNIX Specification:

8778 • The definition of the **mode\_t**, **off\_t**, and **pid\_t** types is mandated.

8779 The F\_GETOWN and F\_SETOWN values are added for sockets.

8780 The *posix\_fadvise()*, *posix\_fallocate()*, and *posix\_madvise()* functions are added for alignment with  
8781 IEEE Std 1003.1d-1999.

8782 IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix\_madvise()* to  
8783 [<sys/mman.h>](#).

8784 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/18 is applied, updating the prototypes for  
8785 *posix\_fadvise()* and *posix\_fallocate()* to be large file-aware, using **off\_t** instead of **size\_t**.

**8786 Issue 7**

8787 Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O\_TTY\_INIT flag.

8788 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the  
8789 FD\_CLOEXEC flag atomically at *open()*, and adding the F\_DUPFD\_CLOEXEC flag.

8790 The *openat()* function is added from The Open Group Technical Standard, 2006, Extended API  
8791 Set Part 2.

8792 Additional flags are added to support *faccessat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*,  
8793 *open()*, *openat()*, and *unlinkat()*.

8794 This reference page is clarified with respect to macros and symbolic constants.

8795 Changes are made related to support for finegrained timestamps.

8796 Changes are made to allow a directory to be opened for searching.

8797 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0044 [274] and XBD/TC1-2008/0045  
8798 [78,432] are applied.

8799 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0060 [847] is applied.

8800 **Issue 8**

8801 Austin Group Defect 768 is applied, adding OFD-owned file locks.

8802 Austin Group Defect 1016 is applied, changing the FUTURE DIRECTIONS section.

8803 Austin Group Defect 1274 is applied, adding the **f\_owner\_ex** structure and related symbolic constants.

8805 Austin Group Defect 1318 is applied, adding FD\_CLOFORK and O\_CLOFORK, and changing O\_CLOEXEC.

8807 Austin Group Defect 1351 is applied, adding F\_DUPFD\_CLOFORK.

8808 **NAME**8809 `fenv.h` — floating-point environment8810 **SYNOPSIS**8811 `#include <fenv.h>`8812 **DESCRIPTION**

8813 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 8814 conflict between the requirements described here and the ISO C standard is unintentional. This  
 8815 volume of POSIX.1-2024 defers to the ISO C standard.

8816 The **<fenv.h>** header shall define the following data types through **typedef**:

8817 **fenv\_t** Represents the entire floating-point environment. The floating-point environment  
 8818 refers collectively to any floating-point status flags and control modes supported  
 8819 by the implementation.

8820 **fexcept\_t** Represents the floating-point status flags collectively, including any status the  
 8821 implementation associates with the flags. A floating-point status flag is a system  
 8822 variable whose value is set (but never cleared) when a floating-point exception is  
 8823 raised, which occurs as a side-effect of exceptional floating-point arithmetic to  
 8824 provide auxiliary information. A floating-point control mode is a system variable  
 8825 whose value may be set by the user to affect the subsequent behavior of floating-  
 8826 point arithmetic.

8827 The **<fenv.h>** header shall define each of the following macros if and only if the implementation  
 8828 supports the floating-point exception by means of the floating-point functions *feclearexcept()*,  
 8829 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. The defined macros shall  
 8830 expand to integer constant expressions with values that are bitwise-distinct.

8831 `FE_DIVBYZERO`  
 8832 `FE_INEXACT`  
 8833 `FE_INVALID`  
 8834 `FE_OVERFLOW`  
 8835 `FE_UNDERFLOW`

8836 MX If the implementation supports the IEC 60559 Floating-Point option, all five macros shall be  
 8837 defined. Additional implementation-defined floating-point exceptions with macros beginning  
 8838 with `FE_` and an uppercase letter may also be specified by the implementation.

8839 The **<fenv.h>** header shall define the macro `FE_ALL_EXCEPT` as the bitwise-inclusive OR of all  
 8840 floating-point exception macros defined by the implementation, if any. If no such macros are  
 8841 defined, then the macro `FE_ALL_EXCEPT` shall be defined as zero.

8842 The **<fenv.h>** header shall define each of the following macros if and only if the implementation  
 8843 supports getting and setting the represented rounding direction by means of the *fegetround()*  
 8844 and *fesetround()* functions. The defined macros shall expand to integer constant expressions  
 8845 whose values are distinct non-negative values.

8846 `FE_DOWNWARD`  
 8847 `FE_TONEAREST`  
 8848 `FE_TOWARDZERO`  
 8849 `FE_UPWARD`

8850 MX If the implementation supports the IEC 60559 Floating-Point option, all four macros shall be  
 8851 defined. Additional implementation-defined rounding directions with macros beginning with  
 8852 `FE_` and an uppercase letter may also be specified by the implementation.

8853 The <fenv.h> header shall define the following macro, which represents the default floating-  
 8854 point environment (that is, the one installed at program startup) and has type pointer to const-  
 8855 qualified **fenv\_t**. It can be used as an argument to the functions within the <fenv.h> header that  
 8856 manage the floating-point environment.

8857 `FE_DFL_ENV`

8858 The following shall be declared as functions and may also be defined as macros. Function  
 8859 prototypes shall be provided.

```
8860 int feclearexcept(int);
8861 int fegetenv(fenv_t *);
8862 int fegetexceptflag(fexcept_t *, int);
8863 int fegetround(void);
8864 int feholdexcept(fenv_t *);
8865 int feraiseexcept(int);
8866 int fesetenv(const fenv_t *);
8867 int fesetexceptflag(const fexcept_t *, int);
8868 int fesetround(int);
8869 int fetestexcept(int);
8870 int feupdateenv(const fenv_t *);
```

8871 The `FENV_ACCESS` pragma provides a means to inform the implementation when an  
 8872 application might access the floating-point environment to test floating-point status flags or run  
 8873 under non-default floating-point control modes. The pragma shall occur either outside external  
 8874 declarations or preceding all explicit declarations and statements inside a compound statement.  
 8875 When outside external declarations, the pragma takes effect from its occurrence until another  
 8876 `FENV_ACCESS` pragma is encountered, or until the end of the translation unit. When inside a  
 8877 compound statement, the pragma takes effect from its occurrence until another `FENV_ACCESS`  
 8878 pragma is encountered (including within a nested compound statement), or until the end of the  
 8879 compound statement; at the end of a compound statement the state for the pragma is restored to  
 8880 its condition just before the compound statement. If this pragma is used in any other context, the  
 8881 behavior is undefined. If part of an application tests floating-point status flags, sets floating-  
 8882 point control modes, or runs under non-default mode settings, but was translated with the state  
 8883 for the `FENV_ACCESS` pragma off, the behavior is undefined. The default state (on or off) for  
 8884 the pragma is implementation-defined. (When execution passes from a part of the application  
 8885 translated with `FENV_ACCESS` off to a part translated with `FENV_ACCESS` on, the state of the  
 8886 floating-point status flags is unspecified and the floating-point control modes have their default  
 8887 settings.)

8888 **APPLICATION USAGE**

8889 This header is designed to support the floating-point exception status flags and directed-  
 8890 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-  
 8891 point state information. Also it is designed to facilitate code portability among all systems.

8892 Certain application programming conventions support the intended model of use for the  
 8893 floating-point environment:

- 8894 • A function call does not alter its caller’s floating-point control modes, clear its caller’s  
 8895 floating-point status flags, nor depend on the state of its caller’s floating-point status flags  
 8896 unless the function is so documented.
- 8897 • A function call is assumed to require default floating-point control modes, unless its  
 8898 documentation promises otherwise.

8899           • A function call is assumed to have the potential for raising floating-point exceptions,  
8900           unless its documentation promises otherwise.

8901           With these conventions, an application can safely assume default floating-point control modes  
8902           (or be unaware of them). The responsibilities associated with accessing the floating-point  
8903           environment fall on the application that does so explicitly.

8904           Even though the rounding direction macros may expand to constants corresponding to the  
8905           values of FLT\_ROUNDS, they are not required to do so.

8906           For example:

```
8907           #include <fenv.h>
8908           void f(double x)
8909           {
8910               #pragma STDC FENV_ACCESS ON
8911               void g(double);
8912               void h(double);
8913               /* ... */
8914               g(x + 1);
8915               h(x + 1);
8916               /* ... */
8917           }
```

8918           If the function *g()* might depend on status flags set as a side-effect of the first *x+1*, or if the  
8919           second *x+1* might depend on control modes set as a side-effect of the call to function *g()*, then  
8920           the application shall contain an appropriately placed invocation as follows:

```
8921           #pragma STDC FENV_ACCESS ON
```

## 8922   **RATIONALE**

### 8923   **The `feexcept_t` Type**

8924           **`feexcept_t`** does not have to be an integer type. Its values must be obtained by a call to  
8925           *fegetexceptflag()*, and cannot be created by logical operations from the exception macros. An  
8926           implementation might simply implement **`feexcept_t`** as an **`int`** and use the representations  
8927           reflected by the exception macros, but is not required to; other representations might contain  
8928           extra information about the exceptions. **`feexcept_t`** might be a **`struct`** with a member for each  
8929           exception (that might hold the address of the first or last floating-point instruction that caused  
8930           that exception). The ISO C standard makes no claims about the internals of an **`feexcept_t`**, and so  
8931           the user cannot inspect it.

### 8932   **Exception and Rounding Macros**

8933           Macros corresponding to unsupported modes and rounding directions are not defined by the  
8934           implementation and must not be defined by the application. An application might use **`#ifndef`**  
8935           to test for this.

## 8936   **FUTURE DIRECTIONS**

8937           None.

## 8938   **SEE ALSO**

8939           XSH *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*, *fegetround()*, *feholdexcept()*, *feraiseexcept()*,  
8940           *fetestexcept()*, *feupdateenv()*

8941 **CHANGE HISTORY**

8942 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8943 The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*,  
8944 *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the  
8945 ISO/IEC 9899:1999 standard, Defect Report 202.

8946 **Issue 7**

8947 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #37 (SD5-XBD-ERN-49) is applied.

8948 ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #36 is applied.

8949 SD5-XBD-ERN-48 and SD5-XBD-ERN-69 are applied.

8950 This reference page is clarified with respect to macros and symbolic constants.

8951 **NAME**

8952 float.h — floating types

8953 **SYNOPSIS**

8954 #include &lt;float.h&gt;

8955 **DESCRIPTION**

8956 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 8957 conflict between the requirements described here and the ISO C standard is unintentional. This  
 8958 volume of POSIX.1-2024 defers to the ISO C standard.

8959 The characteristics of floating types are defined in terms of a model that describes a  
 8960 representation of floating-point numbers and values that provide information about an  
 8961 implementation's floating-point arithmetic.

8962 The following parameters are used to define the model for each floating-point type:

8963 *s* Sign ( $\pm 1$ ).

8964 *b* Base or radix of exponent representation (an integer  $> 1$ ).

8965 *e* Exponent (an integer between a minimum  $e_{\min}$  and a maximum  $e_{\max}$ ).

8966 *p* Precision (the number of base-*b* digits in the significand).

8967  $f_k$  Non-negative integers less than *b* (the significand digits).

8968 A floating-point number *x* is defined by the following model:

$$x = sb^e \sum_{k=1}^p f_k b^{-k}, e_{\min} \leq e \leq e_{\max}$$

8969 In addition to normalized floating-point numbers ( $f_1 > 0$  if  $x \neq 0$ ), floating types may be able to  
 8970 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ( $x \neq 0$ ,  
 8971  $e = e_{\min}$ ,  $f_1 = 0$ ) and unnormalized floating-point numbers ( $x \neq 0$ ,  $e > e_{\min}$ ,  $f_1 = 0$ ), and values that are  
 8972 not floating-point numbers, such as infinities and NaNs. A *NaN* is an encoding signifying Not-a-  
 8973 Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a  
 8974 floating-point exception; a *signaling NaN* generally raises a floating-point exception when  
 8975 occurring as an arithmetic operand.

8976 An implementation may give zero and non-numeric values, such as infinities and NaNs, a sign,  
 8977 or may leave them unsigned. Wherever such values are unsigned, any requirement in  
 8978 POSIX.1-2024 to retrieve the sign shall produce an unspecified sign and any requirement to set  
 8979 the sign shall be ignored.

8980 The accuracy of the floating-point operations ('+', '-', '\*', '/') and of the functions in  
 8981 **<math.h>** and **<complex.h>** that return floating-point results is implementation-defined, as is  
 8982 the accuracy of the conversion between floating-point internal representations and string  
 8983 representations performed by the functions in **<stdio.h>**, **<stdlib.h>**, and **<wchar.h>**. The  
 8984 implementation may state that the accuracy is unknown.

8985 All integer values in the **<float.h>** header, except FLT\_ROUNDS, shall be constant expressions  
 8986 suitable for use in **#if** preprocessing directives; all floating values shall be constant expressions.  
 8987 All except DECIMAL\_DIG, FLT\_EVAL\_METHOD, FLT\_RADIX, and FLT\_ROUNDS have  
 8988 separate names for all three floating-point types. The floating-point model representation is  
 8989 provided for all values except FLT\_EVAL\_METHOD and FLT\_ROUNDS.

8990 The rounding mode for floating-point addition is characterized by the implementation-defined



8991 value of FLT\_ROUNDS:

8992 -1 Indeterminable.

8993 0 Toward zero.

8994 1 To nearest.

8995 2 Toward positive infinity.

8996 3 Toward negative infinity.

8997 All other values for FLT\_ROUNDS characterize implementation-defined rounding behavior.

8998 The values of operations with floating operands and values subject to the usual arithmetic

8999 conversions and of floating constants are evaluated to a format whose range and precision may

9000 be greater than required by the type. The use of evaluation formats is characterized by the

9001 implementation-defined value of FLT\_EVAL\_METHOD:

9002 -1 Indeterminable.

9003 0 Evaluate all operations and constants just to the range and precision of the type.

9004 1 Evaluate operations and constants of type **float** and **double** to the range and precision of

9005 the **double** type; evaluate **long double** operations and constants to the range and precision

9006 of the **long double** type.

9007 2 Evaluate all operations and constants to the range and precision of the **long double** type.

9008 All other negative values for FLT\_EVAL\_METHOD characterize implementation-defined

9009 behavior.

9010 The presence or absence of subnormal numbers is characterized by the implementation-defined

9011 values of FLT\_HAS\_SUBNORM, DBL\_HAS\_SUBNORM, and LDBL\_HAS\_SUBNORM:

9012 -1 Indeterminable.

9013 0 Absent (type does not support subnormal numbers).

9014 1 Present (type does support subnormal numbers).

9015 **Note:** Characterization as indeterminable is intended if floating-point operations do not consistently

9016 interpret subnormal representations as zero, nor as non-zero. Characterization as absent is

9017 intended if no floating-point operations produce subnormal results from non-subnormal inputs,

9018 even if the type format includes representations of subnormal numbers.

9019 The <float.h> header shall define the following values as constant expressions with

9020 implementation-defined values that are greater or equal in magnitude (absolute value) to those

9021 shown, with the same sign.

9022 • Radix of exponent representation,  $b$ .

9023 FLT\_RADIX 2

9024 • Number of base-FLT\_RADIX digits in the floating-point significand,  $p$ .

9025 FLT\_MANT\_DIG

9026 DBL\_MANT\_DIG

9027 LDBL\_MANT\_DIG

9028 • Number of decimal digits,  $n$ , such that any floating-point number with  $p$  radix  $b$  digits can

9029 be rounded to a floating-point number with  $n$  decimal digits and back again without

9030 change to the value.

$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lceil 1 + p \log_{10} b \rceil & \text{otherwise} \end{cases}$$

9031 FLT\_DECIMAL\_DIG 6  
 9032 DBL\_DECIMAL\_DIG 10  
 9033 LDBL\_DECIMAL\_DIG 10

- Number of decimal digits,  $n$ , such that any floating-point number in the widest supported floating type with  $p_{\max}$  radix  $b$  digits can be rounded to a floating-point number with  $n$  decimal digits and back again without change to the value.

$$\begin{cases} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lceil 1 + p_{\max} \log_{10} b \rceil & \text{otherwise} \end{cases}$$

9037 DECIMAL\_DIG 10

- Number of decimal digits,  $q$ , such that any floating-point number with  $q$  decimal digits can be rounded into a floating-point number with  $p$  radix  $b$  digits and back again without change to the  $q$  decimal digits.

$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lceil (p - 1) \log_{10} b \rceil & \text{otherwise} \end{cases}$$

9041 FLT\_DIG 6  
 9042 DBL\_DIG 10  
 9043 LDBL\_DIG 10

- Minimum negative integer such that FLT\_RADIX raised to that power minus 1 is a normalized floating-point number,  $e_{\min}$ .

9046 FLT\_MIN\_EXP  
 9047 DBL\_MIN\_EXP  
 9048 LDBL\_MIN\_EXP

- Minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers.

$$\lceil \log_{10} b^{e_{\min} - 1} \rceil$$

9051 FLT\_MIN\_10\_EXP -37  
 9052 DBL\_MIN\_10\_EXP -37  
 9053 LDBL\_MIN\_10\_EXP -37  
 9054 • Maximum integer such that FLT\_RADIX raised to that power minus 1 is a representable  
 9055 finite floating-point number,  $e_{\max}$ .  
 9056 FLT\_MAX\_EXP  
 9057 DBL\_MAX\_EXP  
 9058 LDBL\_MAX\_EXP  
 9059 CX Additionally, FLT\_MAX\_EXP shall be at least as large as FLT\_MANT\_DIG,  
 9060 DBL\_MAX\_EXP shall be at least as large as DBL\_MANT\_DIG, and LDBL\_MAX\_EXP shall  
 9061 be at least as large as LDBL\_MANT\_DIG; which has the effect that FLT\_MAX, DBL\_MAX,  
 9062 and LDBL\_MAX are integral.  
 9063 • Maximum integer such that 10 raised to that power is in the range of representable finite  
 9064 floating-point numbers.

$$\left\lfloor \log_{10}((1 - b^{-p}) b^{e_{\max}}) \right\rfloor$$

9065 FLT\_MAX\_10\_EXP +37  
 9066 DBL\_MAX\_10\_EXP +37  
 9067 LDBL\_MAX\_10\_EXP +37

9068 The <float.h> header shall define the following values as constant expressions with  
 9069 implementation-defined values that are greater than or equal to those shown:

- 9070 • Maximum representable finite floating-point number.

$$(1 - b^{-p}) b^{e_{\max}}$$

9071 FLT\_MAX 1E+37  
 9072 DBL\_MAX 1E+37  
 9073 LDBL\_MAX 1E+37

9074 The <float.h> header shall define the following values as constant expressions with  
 9075 implementation-defined (positive) values that are less than or equal to those shown:

- 9076 • The difference between 1 and the least value greater than 1 that is representable in the  
 9077 given floating-point type,  $b^{1-p}$ .

9078 FLT\_EPSILON 1E-5  
 9079 DBL\_EPSILON 1E-9  
 9080 LDBL\_EPSILON 1E-9

- 9081 • Minimum normalized positive floating-point number,  $b^{e_{\min} - 1}$ .

9082 FLT\_MIN 1E-37

9083 DBL\_MIN 1E-37

9084 LDBL\_MIN 1E-37

9085 • Minimum positive floating-point number.

9086 FLT\_TRUE\_MIN 1E-37

9087 DBL\_TRUE\_MIN 1E-37

9088 LDBL\_TRUE\_MIN 1E-37

9089 **Note:** If the presence or absence of subnormal numbers is indeterminable, then the value is  
9090 intended to be a positive number no greater than the minimum normalized positive  
9091 number for the type.

## 9092 APPLICATION USAGE

9093 None.

## 9094 RATIONALE

9095 All known hardware floating-point formats satisfy the property that the exponent range is larger  
9096 than the number of digits in the significand. The ISO C standard permits a floating-point format  
9097 where this property is not true, such that the largest finite value would not be integral; however,  
9098 it is unlikely that there will ever be hardware support for such a floating-point format, and it  
9099 introduces boundary cases that portable programs should not have to be concerned with (for  
9100 example, a non-integral DBL\_MAX means that *ceil()* would have to worry about overflow).  
9101 Therefore, this standard imposes an additional requirement that the largest representable finite  
9102 value is integral.

## 9103 FUTURE DIRECTIONS

9104 The formula for calculating FLT\_MAX, DBL\_MAX, and LDBL\_MAX is expected to change in the  
9105 next revision of the ISO C standard such that it only applies if the values are normalized.

## 9106 SEE ALSO

9107 [<complex.h>](#), [<math.h>](#), [<stdio.h>](#), [<stdlib.h>](#), [<wchar.h>](#)

## 9108 CHANGE HISTORY

9109 First released in Issue 4. Derived from the ISO C standard.

### 9110 Issue 6

9111 The description of the operations with floating-point values is updated for alignment with the  
9112 ISO/IEC 9899:1999 standard.

### 9113 Issue 7

9114 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #4 (SD5-XBD-ERN-50) and #5  
9115 (SD5-XBD-ERN-51) are applied.

9116 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0046 [346] and XBD/TC1-2008/0047  
9117 [346] are applied.

### 9118 Issue 8

9119 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.

9120 Austin Group Defect 1752 is applied, changing “the number of mantissa digits” to “the number  
9121 of digits in the significand”.

9122 Austin Group Defect 1754 is applied, changing the FUTURE DIRECTIONS section.

9123 **NAME**

9124 `fmtmsg.h` — message display structures

9125 **SYNOPSIS**

9126 XSI `#include <fmtmsg.h>`

9127 **DESCRIPTION**

9128 The <fmtmsg.h> header shall define the following symbolic constants:

- 9129 `MM_HARD`           Source of the condition is hardware.
- 9130 `MM_SOFT`           Source of the condition is software.
- 9131 `MM_FIRM`           Source of the condition is firmware.
- 9132 `MM_APPL`           Condition detected by application.
- 9133 `MM_UTIL`           Condition detected by utility.
- 9134 `MM_OPSYS`          Condition detected by operating system.
- 9135 `MM_RECOVER`       Recoverable error.
- 9136 `MM_NRECOV`       Non-recoverable error.
- 9137 `MM_HALT`           Error causing application to halt.
- 9138 `MM_ERROR`          Application has encountered a non-fatal fault.
- 9139 `MM_WARNING`       Application has detected unusual non-error condition.
- 9140 `MM_INFO`           Informative message.
- 9141 `MM_NOSEV`          No severity level provided for the message.
- 9142 `MM_PRINT`          Display message on standard error.
- 9143 `MM_CONSOLE`       Display message on system console.

9144 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The  
 9145 <fmtmsg.h> header shall define the symbolic constants in the **Identifier** column, which shall  
 9146 have the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
<i>label</i>	<b>char *</b>	(char*)0	MM_NULLLBL
<i>severity</i>	<b>int</b>	0	MM_NULLSEV
<i>class</i>	<b>long</b>	0L	MM_NULLMC
<i>text</i>	<b>char *</b>	(char*)0	MM_NULLTXT
<i>action</i>	<b>char *</b>	(char*)0	MM_NULLACT
<i>tag</i>	<b>char *</b>	(char*)0	MM_NULLTAG

9154 The <fmtmsg.h> header shall also define the following symbolic constants for use as return  
 9155 values for *fmtmsg()*:

- 9156 `MM_OK`            The function succeeded.
- 9157 `MM_NOTOK`        The function failed completely.
- 9158 `MM_NOMSG`        The function was unable to generate a message on standard error, but  
 9159 otherwise succeeded.

9160           MM\_NOCON           The function was unable to generate a console message, but otherwise  
9161                                   succeeded.

9162           The following shall be declared as a function and may also be defined as a macro. A function  
9163           prototype shall be provided.

```
9164           int fmtmsg(long, const char *, int,  
9165           const char *, const char *, const char *);
```

9166   **APPLICATION USAGE**

9167           None.

9168   **RATIONALE**

9169           None.

9170   **FUTURE DIRECTIONS**

9171           None.

9172   **SEE ALSO**

9173           XSH *fmtmsg()*

9174   **CHANGE HISTORY**

9175           First released in Issue 4, Version 2.

9176   **Issue 7**

9177           This reference page is clarified with respect to macros and symbolic constants.

9178 **NAME**

9179 fnmatch.h — filename-matching types

9180 **SYNOPSIS**

9181 #include <fnmatch.h>

9182 **DESCRIPTION**

9183 The <fnmatch.h> header shall define the following symbolic constants:

9184 FNM\_NOMATCH The string does not match the specified pattern.

9185 FNM\_PATHNAME <slash> in *string* only matches <slash> in *pattern*.

9186 FNM\_PERIOD Leading <period> in *string* only matches <period> in *pattern*.

9187 FNM\_NOESCAPE Disable backslash escaping.

9188 FNM\_CASEFOLD Compare *string* and *pattern* in a case-insensitive manner. See [Section 4.1](#)  
9189 (on page 95).

9190 FNM\_IGNORECASE Equivalent to FNM\_CASEFOLD.

9191 The following shall be declared as a function and may also be defined as a macro. A function  
9192 prototype shall be provided.

9193 int fnmatch(const char \*, const char \*, int);

9194 **APPLICATION USAGE**

9195 None.

9196 **RATIONALE**

9197 None.

9198 **FUTURE DIRECTIONS**

9199 None.

9200 **SEE ALSO**

9201 XSH [fnmatch\(\)](#)

9202 **CHANGE HISTORY**

9203 First released in Issue 4. Derived from the ISO POSIX-2 standard.

9204 **Issue 6**

9205 The FNM\_NOSYS constant is marked obsolescent.

9206 **Issue 7**

9207 The obsolescent FNM\_NOSYS constant is removed.

9208 This reference page is clarified with respect to macros and symbolic constants.

9209 **Issue 8**

9210 Austin Group Defect 1031 is applied, adding FNM\_CASEFOLD and FNM\_IGNORECASE.

9211 The description of FNM\_PERIOD is updated to eliminate the use of “must”.

9212 **NAME**

9213 ftw.h — file tree traversal

9214 **SYNOPSIS**9215 XSI `#include <ftw.h>`9216 **DESCRIPTION**9217 The **<ftw.h>** header shall define the **FTW** structure, which shall include at least the following  
9218 members:9219 `int base`  
9220 `int level`9221 The **<ftw.h>** header shall define the following symbolic constants for use as values of the third  
9222 argument to the application-supplied function that is passed as the second argument to *nftw()*:9223 **FTW\_F** Non-directory file.  
9224 **FTW\_D** Directory.  
9225 **FTW\_DNR** Directory without read permission.  
9226 **FTW\_DP** Directory with subdirectories visited.  
9227 **FTW\_NS** Unknown type; *stat()* failed.  
9228 **FTW\_SL** Symbolic link.  
9229 **FTW\_SLN** Symbolic link that names a nonexistent file.9230 The **<ftw.h>** header shall define the following symbolic constants for use as values of the fourth  
9231 argument to *nftw()*:9232 **FTW\_PHYS** Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows  
9233 links but does not walk down any path that crosses itself.  
9234 **FTW\_MOUNT** The walk only reports files that have the same device ID as the starting  
9235 directory and does not descend below directories that have a different  
9236 device ID than the starting directory.  
9237 **FTW\_XDEV** The walk does not descend below directories that have a different device  
9238 ID than the starting directory.  
9239 **FTW\_DEPTH** All subdirectories are visited before the directory itself.  
9240 **FTW\_CHDIR** The walk changes to each directory before reading it.9241 The following shall be declared as a function and may also be defined as a macro. A function  
9242 prototype shall be provided.9243 `int nftw(const char *, int (*)(const char *, const struct stat *,`  
9244 `int, struct FTW *), int, int);`9245 The **<ftw.h>** header shall define the **stat** structure and the symbolic names for *st\_mode* and the  
9246 file type test macros as described in **<sys/stat.h>**.9247 Inclusion of the **<ftw.h>** header may also make visible all symbols from **<sys/stat.h>**.



9248 **APPLICATION USAGE**

9249 None.

9250 **RATIONALE**

9251 None.

9252 **FUTURE DIRECTIONS**

9253 None.

9254 **SEE ALSO**9255 [<sys/stat.h>](#)9256 XSH *nftw()*9257 **CHANGE HISTORY**

9258 First released in Issue 1. Derived from Issue 1 of the SVID.

9259 **Issue 5**

9260 A description of FTW\_DP is added.

9261 **Issue 7**9262 The *ftw()* function is marked obsolescent.

9263 This reference page is clarified with respect to macros and symbolic constants.

9264 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0048 [403] is applied.

9265 **Issue 8**

9266 Austin Group Defect 1133 is applied, adding FTW\_XDEV.

9267 Austin Group Defect 1210 is applied, changing the description of FTW\_MOUNT.

9268 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

9269 **NAME**

9270 glob.h — pathname pattern-matching types

9271 **SYNOPSIS**

9272 #include &lt;glob.h&gt;

9273 **DESCRIPTION**9274 The **<glob.h>** header shall define the structures and symbolic constants used by the *glob()*  
9275 function.9276 The **<glob.h>** header shall define the **glob\_t** structure type, which shall include at least the  
9277 following members:9278 size\_t gl\_pathc Count of paths matched by *pattern*.  
9279 char \*\*gl\_pathv Pointer to a list of matched pathnames.  
9280 size\_t gl\_offs Slots to reserve at the beginning of *gl\_pathv*.9281 The **<glob.h>** header shall define the **size\_t** type as described in **<sys/types.h>**.9282 The **<glob.h>** header shall define the following symbolic constants as values for the *flags*  
9283 argument:9284 GLOB\_APPEND Append generated pathnames to those previously obtained.  
9285 GLOB\_DOOFFS Specify how many null pointers to add to the beginning of *gl\_pathv*.  
9286 GLOB\_ERR Cause *glob()* to return on error.  
9287 GLOB\_MARK Each pathname that is a directory that matches *pattern* has a <slash>  
9288 appended.  
9289 GLOB\_NOCHECK If *pattern* does not match any pathname, then return a list consisting of  
9290 only *pattern*.  
9291 GLOB\_NOESCAPE Disable backslash escaping.  
9292 GLOB\_NOSORT Do not sort the pathnames returned.9293 The **<glob.h>** header shall define the following symbolic constants as error return values:9294 GLOB\_ABORTED The scan was stopped because GLOB\_ERR was set or (\*errfunc)()  
9295 returned non-zero.  
9296 GLOB\_NOMATCH The pattern does not match any existing pathname, and  
9297 GLOB\_NOCHECK was not set in *flags*.  
9298 GLOB\_NOSPACE An attempt to allocate memory failed.9299 The following shall be declared as functions and may also be defined as macros. Function  
9300 prototypes shall be provided.9301 int glob(const char \*restrict, int, int (\*)(const char \*, int),  
9302 glob\_t \*restrict);  
9303 void globfree(glob\_t \*);

9304 **APPLICATION USAGE**

9305 None.

9306 **RATIONALE**

9307 None.

9308 **FUTURE DIRECTIONS**

9309 None.

9310 **SEE ALSO**9311 [<sys/types.h>](#)9312 XSH *glob()*9313 **CHANGE HISTORY**

9314 First released in Issue 4. Derived from the ISO POSIX-2 standard.

9315 **Issue 6**9316 The **restrict** keyword is added to the prototype for *glob()*.

9317 The GLOB\_NOSYS constant is marked obsolescent.

9318 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the *glob()*  
9319 prototype definition by removing the **restrict** qualifier from the function pointer argument.9320 **Issue 7**9321 SD5-XBD-ERN-56 is applied, adding a reference to [<sys/types.h>](#) for the **size\_t**

9322 The obsolescent GLOB\_NOSYS constant is removed.

9323 This reference page is clarified with respect to macros and symbolic constants.

9324 **NAME**

9325       grp.h — group structure

9326 **SYNOPSIS**

9327       #include &lt;grp.h&gt;

9328 **DESCRIPTION**9329       The **<grp.h>** header shall declare the **group** structure, which shall include the following  
9330       members:

9331       char     \*gr\_name   The name of the group.  
 9332       gid\_t    gr\_gid     Numerical group ID.  
 9333       char    \*\*gr\_mem    Pointer to a null-terminated array of character  
 9334       pointers to member names.

9335       The **<grp.h>** header shall define the **gid\_t** and **size\_t** types as described in **<sys/types.h>**.9336       The following shall be declared as functions and may also be defined as macros. Function  
9337       prototypes shall be provided.

```

9338 XSI     void            endgrent(void);
9339         struct group   *getgrent(void);
9340         struct group   *getgrgid(gid_t);
9341         int            getgrgid_r(gid_t, struct group *, char *,
9342                         size_t, struct group **);
9343         struct group   *getgrnam(const char *);
9344         int            getgrnam_r(const char *, struct group *, char *,
9345                         size_t , struct group **);
9346 XSI     void            setgrent(void);

```

9347 **APPLICATION USAGE**

9348       None.

9349 **RATIONALE**

9350       None.

9351 **FUTURE DIRECTIONS**

9352       None.

9353 **SEE ALSO**9354       [<sys/types.h>](#)9355       XSH *endgrent()*, *getgrgid()*, *getgrnam()*9356 **CHANGE HISTORY**

9357       First released in Issue 1.

9358 **Issue 5**

9359       The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

9360 **Issue 6**9361       The following new requirements on POSIX implementations derive from alignment with the  
9362       Single UNIX Specification:

- 9363       • The definition of **gid\_t** is mandated.
- 9364       • The *getgrgid\_r()* and *getgrnam\_r()* functions are marked as part of the Thread-Safe  
9365        Functions option.

9366 **Issue 7**

9367 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size\_t** type.

9368 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0049 [24] is applied.

9369 **NAME**

9370 iconv.h — codeset conversion facility

9371 **SYNOPSIS**

9372 #include &lt;iconv.h&gt;

9373 **DESCRIPTION**9374 The **<iconv.h>** header shall define the following types:9375 **iconv\_t** Identifies the conversion from one codeset to another.9376 **size\_t** As described in **<sys/types.h>**.9377 The following shall be declared as functions and may also be defined as macros. Function  
9378 prototypes shall be provided.9379 size\_t iconv(iconv\_t, char \*\*restrict, size\_t \*restrict,  
9380 char \*\*restrict, size\_t \*restrict);  
9381 int iconv\_close(iconv\_t);  
9382 iconv\_t iconv\_open(const char \*, const char \*);9383 **APPLICATION USAGE**

9384 None.

9385 **RATIONALE**

9386 None.

9387 **FUTURE DIRECTIONS**

9388 None.

9389 **SEE ALSO**9390 [<sys/types.h>](#)9391 XSH [iconv\(\)](#), [iconv\\_close\(\)](#), [iconv\\_open\(\)](#)9392 **CHANGE HISTORY**

9393 First released in Issue 4.

9394 **Issue 6**9395 The **restrict** keyword is added to the prototype for [iconv\(\)](#).9396 **Issue 7**9397 SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size\_t** type.9398 The **<iconv.h>** header is moved from the XSI option to the Base.

9399 **NAME**

9400 inttypes.h — fixed size integer types

9401 **SYNOPSIS**

9402 #include <inttypes.h>

9403 **DESCRIPTION**

9404 CX Some of the functionality described on this reference page extends the ISO C standard.  
 9405 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 9406 enable the visibility of these symbols in this header.

9407 The <inttypes.h> header shall include the <stdint.h> header.

9408 The <inttypes.h> header shall define at least the following types:

9409 **imaxdiv\_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

9410 CX **wchar\_t** As described in <stddef.h>.

9411 The <inttypes.h> header shall define the following macros. Each expands to a character string  
 9412 literal containing a conversion specifier, possibly modified by a length modifier, suitable for use  
 9413 within the *format* argument of a formatted input/output function when converting the  
 9414 corresponding integer type. These macros have the general form of PRI (character string literals  
 9415 for the *fprintf()* and *fwprintf()* family of functions) or SCN (character string literals for the  
 9416 *fscanf()* and *fwscanf()* family of functions), followed by the conversion specifier, followed by a  
 9417 name corresponding to a similar type name in <stdint.h>. In these names, *N* represents the  
 9418 width of the type as described in <stdint.h>. For example, *PRIdFAST32* can be used in a format  
 9419 string to print the value of an integer of type **int\_fast32\_t**.

9420 The *fprintf()* macros for signed integers are:

9421	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
9422	PRiN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR

9423 The *fprintf()* macros for unsigned integers are:

9424	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
9425	PRiUN	PRiULEASTN	PRiUFASTN	PRiUMAX	PRiUPTR
9426	PRiXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR
9427	PRiXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR

9428 The *fscanf()* macros for signed integers are:

9429	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
9430	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

9431 The *fscanf()* macros for unsigned integers are:

9432	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
9433	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
9434	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

9435 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* and  
 9436 *fwprintf()* macros shall be defined and the corresponding *fscanf()* and *fwscanf()* macros shall be  
 9437 defined unless the implementation does not have a suitable modifier for the type.

9438 The following shall be declared as functions and may also be defined as macros. Function  
 9439 prototypes shall be provided.

```
9440     intmax_t  imaxabs(intmax_t);
9441     imaxdiv_t imaxdiv(intmax_t, intmax_t);
9442     intmax_t  strtoumax(const char *restrict, char **restrict, int);
9443     uintmax_t strtoumax(const char *restrict, char **restrict, int);
9444     intmax_t  wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
9445     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
```

**9446 EXAMPLES**

```
9447     #include <inttypes.h>
9448     #include <wchar.h>
9449     int main(void)
9450     {
9451         uintmax_t i = UINTMAX_MAX; // This type always exists.
9452         wprintf(L"The largest integer value is %020"
9453             PRIxMAX "\n", i);
9454         return 0;
9455     }
```

**9456 APPLICATION USAGE**

9457 The purpose of **<inttypes.h>** is to provide a set of integer types whose definitions are consistent  
9458 across machines and independent of operating systems and other implementation  
9459 idiosyncrasies. It defines, through **typedef**, integer types of various sizes. Implementations are  
9460 free to **typedef** them as ISO C standard integer types or extensions that they support. Consistent  
9461 use of this header will greatly increase the portability of applications across platforms.

**9462 RATIONALE**

9463 The ISO/IEC 9899:1990 standard specified that the language should support four signed and  
9464 unsigned integer data types—**char**, **short**, **int**, and **long**—but placed very little requirement on  
9465 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and  
9466 not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits  
9467 to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to  
9468 assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems  
9469 for users who migrate from one system to another which assigns different sizes to integer types,  
9470 because the ISO C standard integer promotion rule can produce silent changes unexpectedly.  
9471 The need for defining an extended integer type increased with the introduction of 64-bit  
9472 systems.

**9473 FUTURE DIRECTIONS**

9474 Macro names beginning with PRI or SCN followed by any lowercase letter or 'x' may be added  
9475 to the macros defined in the **<inttypes.h>** header.

**9476 SEE ALSO**

9477 [<stddef.h>](#)

9478 XSH Section 2.2 (on page 496), [imaxabs\(\)](#), [imaxdiv\(\)](#), [strtoumax\(\)](#), [wcstoumax\(\)](#)

**9479 CHANGE HISTORY**

9480 First released in Issue 5.

**9481 Issue 6**

9482 The Open Group Base Resolution bwg97-006 is applied.

9483 This reference page is updated to align with the ISO/IEC 9899:1999 standard.



9484 **Issue 7**  
9485

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0050 [211] is applied.

9486 **NAME**

9487       iso646.h — alternative spellings

9488 **SYNOPSIS**

9489       #include &lt;iso646.h&gt;

9490 **DESCRIPTION**

9491 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
9492 conflict between the requirements described here and the ISO C standard is unintentional. This  
9493 volume of POSIX.1-2024 defers to the ISO C standard.

9494       The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the  
9495 corresponding tokens (on the right):

9496       and        &amp;&amp;

9497       and\_eq     &amp;=

9498       bitand     &amp;

9499       bitor      |

9500       compl     ~

9501       not        !

9502       not\_eq    !=

9503       or         ||

9504       or\_eq     |=

9505       xor       ^

9506       xor\_eq    ^=

9507 **APPLICATION USAGE**

9508       None.

9509 **RATIONALE**

9510       None.

9511 **FUTURE DIRECTIONS**

9512       None.

9513 **SEE ALSO**

9514       None.

9515 **CHANGE HISTORY**

9516       First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

9517 **NAME**

9518 langinfo.h — language information constants

9519 **SYNOPSIS**

9520 #include <langinfo.h>

9521 **DESCRIPTION**

9522 The <langinfo.h> header shall define the symbolic constants used to identify items of *langinfo*  
 9523 data (see *nl\_langinfo()*).

9524 The <langinfo.h> header shall define the **locale\_t** type as described in <locale.h>.

9525 The <langinfo.h> header shall define the **nl\_item** type as described in <nl\_types.h>.

9526 The <langinfo.h> header shall define the following symbolic constants with type **nl\_item**. The  
 9527 entries under **Category** indicate in which *setlocale()* category each item is defined.

Constant	Category	Meaning
CODESET	LC_CTYPE	Codeset name.
D_T_FMT	LC_TIME	String for formatting date and time.
D_FMT	LC_TIME	Date format string.
T_FMT	LC_TIME	Time format string.
T_FMT_AMPM	LC_TIME	Time format string using 12-hour clock format, if supported in the locale; if the 12-hour format is not supported, this shall be either an empty string or a string specifying a 24-hour clock format.
AM_STR	LC_TIME	Ante-meridiem affix; if AM_STR and PM_STR are both empty strings, the 12-hour format is not supported in the locale.
PM_STR	LC_TIME	Post-meridiem affix; if AM_STR and PM_STR are both empty strings, the 12-hour format is not supported in the locale.
DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
DAY_4	LC_TIME	Name of the fourth day of the week (for example, Wednesday).
DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
DAY_7	LC_TIME	Name of the seventh day of the week (for example, Saturday).
ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
MON_1	LC_TIME	Name of the first month of the year.
MON_2	LC_TIME	Name of the second month.
MON_3	LC_TIME	Name of the third month.
MON_4	LC_TIME	Name of the fourth month.
MON_5	LC_TIME	Name of the fifth month.
MON_6	LC_TIME	Name of the sixth month.

	Constant	Category	Meaning
9565			
9566	MON_7	LC_TIME	Name of the seventh month.
9567	MON_8	LC_TIME	Name of the eighth month.
9568	MON_9	LC_TIME	Name of the ninth month.
9569	MON_10	LC_TIME	Name of the tenth month.
9570	MON_11	LC_TIME	Name of the eleventh month.
9571	MON_12	LC_TIME	Name of the twelfth month.
9572	ALTMON_1	LC_TIME	Name of the alternative appropriate first month of the year.
9573	ALTMON_2	LC_TIME	Name of the alternative appropriate second month.
9574	ALTMON_3	LC_TIME	Name of the alternative appropriate third month.
9575	ALTMON_4	LC_TIME	Name of the alternative appropriate fourth month.
9576	ALTMON_5	LC_TIME	Name of the alternative appropriate fifth month.
9577	ALTMON_6	LC_TIME	Name of the alternative appropriate sixth month.
9578	ALTMON_7	LC_TIME	Name of the alternative appropriate seventh month.
9579	ALTMON_8	LC_TIME	Name of the alternative appropriate eighth month.
9580	ALTMON_9	LC_TIME	Name of the alternative appropriate ninth month.
9581	ALTMON_10	LC_TIME	Name of the alternative appropriate tenth month.
9582	ALTMON_11	LC_TIME	Name of the alternative appropriate eleventh month.
9583	ALTMON_12	LC_TIME	Name of the alternative appropriate twelfth month.
9584	ABMON_1	LC_TIME	Abbreviated name of the first month.
9585	ABMON_2	LC_TIME	Abbreviated name of the second month.
9586	ABMON_3	LC_TIME	Abbreviated name of the third month.
9587	ABMON_4	LC_TIME	Abbreviated name of the fourth month.
9588	ABMON_5	LC_TIME	Abbreviated name of the fifth month.
9589	ABMON_6	LC_TIME	Abbreviated name of the sixth month.
9590	ABMON_7	LC_TIME	Abbreviated name of the seventh month.
9591	ABMON_8	LC_TIME	Abbreviated name of the eighth month.
9592	ABMON_9	LC_TIME	Abbreviated name of the ninth month.
9593	ABMON_10	LC_TIME	Abbreviated name of the tenth month.
9594	ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
9595	ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
9596	ABALTMON_1	LC_TIME	Abbreviated alternative name of the first month of the year.
9597	ABALTMON_2	LC_TIME	Abbreviated alternative name of the second month.
9598	ABALTMON_3	LC_TIME	Abbreviated alternative name of the third month.
9599	ABALTMON_4	LC_TIME	Abbreviated alternative name of the fourth month.
9600	ABALTMON_5	LC_TIME	Abbreviated alternative name of the fifth month.
9601	ABALTMON_6	LC_TIME	Abbreviated alternative name of the sixth month.
9602	ABALTMON_7	LC_TIME	Abbreviated alternative name of the seventh month.
9603	ABALTMON_8	LC_TIME	Abbreviated alternative name of the eighth month.
9604	ABALTMON_9	LC_TIME	Abbreviated alternative name of the ninth month.
9605	ABALTMON_10	LC_TIME	Abbreviated alternative name of the tenth month.
9606	ABALTMON_11	LC_TIME	Abbreviated alternative name of the eleventh month.
9607	ABALTMON_12	LC_TIME	Abbreviated alternative name of the twelfth month.
9608	ERA	LC_TIME	Era description segments.
9609	ERA_D_FMT	LC_TIME	Era date format string.
9610	ERA_D_T_FMT	LC_TIME	Era date and time format string.
9611	ERA_T_FMT	LC_TIME	Era time format string.
9612	ALT_DIGITS	LC_TIME	Alternative symbols for digits.
9613	RADIXCHAR	LC_NUMERIC	Radix character.
9614	THOUSEP	LC_NUMERIC	Separator for thousands.
9615	YESEXPR	LC_MESSAGES	Affirmative response expression.

Constant	Category	Meaning
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Local currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string ("").

If the locale's values for **p\_cs\_precedes** and **n\_cs\_precedes** do not match, the value of *nl\_langinfo(CRNCYSTR)* and *nl\_langinfo\_l(CRNCYSTR,loc)* is unspecified.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
char *nl_langinfo(nl_item);
char *nl_langinfo_l(nl_item, locale_t);
```

Inclusion of the <langinfo.h> header may also make visible all symbols from <nl\_types.h>.

**APPLICATION USAGE**

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of *langinfo* data. In particular, the *strftime()* function should be used to access date and time information defined in category *LC\_TIME*. The *localeconv()* function should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and *CRNCYSTR*.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

Chapter 7 (on page 127), <locale.h>, <nl\_types.h>

XSH *nl\_langinfo()*, *localeconv()*, *strfmon()*, *strftime()*

**CHANGE HISTORY**

First released in Issue 2.

**Issue 5**

The constants *YESSTR* and *NOSTR* are marked *LEGACY*.

**Issue 6**

The constants *YESSTR* and *NOSTR* are removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to the "Meaning" column entry for the *CRNCYSTR* constant. This change is to accommodate historic practice.

**Issue 7**

The <langinfo.h> header is moved from the XSI option to the Base.

The *nl\_langinfo\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants, and a declaration for the *locale\_t* type is added.

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0051 [107] is applied.

9659 **Issue 8**

9660 Austin Group Defects 258 and 1166 are applied, adding the ALTMON\_*x* and ABALTMON\_*x*  
9661 symbolic constants.

9662 Austin Group Defect 1307 is applied, changing the AM\_STR, PM\_STR, and T\_FMT\_AMP  
9663 constants in relation to locales that do not support the 12-hour clock format.

9664 **NAME**  
9665 libgen.h — definitions for pattern matching functions

9666 **SYNOPSIS**

9667 XSI `#include <libgen.h>`

9668 **DESCRIPTION**

9669 The following shall be declared as functions and may also be defined as macros. Function  
9670 prototypes shall be provided.

9671 `char *basename(char *);`  
9672 `char *dirname(char *);`

9673 **APPLICATION USAGE**

9674 None.

9675 **RATIONALE**

9676 None.

9677 **FUTURE DIRECTIONS**

9678 None.

9679 **SEE ALSO**

9680 XSH *basename()*, *dirname()*

9681 **CHANGE HISTORY**

9682 First released in Issue 4, Version 2.

9683 **Issue 5**

9684 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first  
9685 argument is of type **char \*** rather than **const char \***.

9686 **Issue 6**

9687 The `__loc1` symbol and the *regcmp()* and *regex()* functions are removed.

9688 **NAME**

9689 libintl.h — international messaging

9690 **SYNOPSIS**

9691 #include &lt;libintl.h&gt;

9692 **DESCRIPTION**9693 The **<libintl.h>** header may define the macro TEXTDOMAINMAX. If defined, it shall have the  
9694 same value as {TEXTDOMAIN\_MAX} in **<limits.h>**.9695 The **<libintl.h>** header shall define the **locale\_t** type as described in **<locale.h>**.9696 The following shall be declared as functions and may also be defined as macros. Function  
9697 prototypes shall be provided.

```
9698 char *bindtextdomain(const char *, const char *);
9699 char *bind_textdomain_codeset(const char *, const char *);
9700 char *dcgettext(const char *, const char *, int);
9701 char *dcgettext_l(const char *, const char *, int, locale_t);
9702 char *dcngettext(const char *, const char *, const char *,
9703                 unsigned long int, int);
9704 char *dcngettext_l(const char *, const char *, const char *,
9705                   unsigned long int, int, locale_t);
9706 char *dgettext(const char *, const char *);
9707 char *dgettext_l(const char *, const char *, locale_t);
9708 char *dngettext(const char *, const char *, const char *,
9709                unsigned long int);
9710 char *dngettext_l(const char *, const char *, const char *,
9711                  unsigned long int, locale_t);
9712 char *gettext(const char *);
9713 char *gettext_l(const char *, locale_t);
9714 char *ngettext(const char *, const char *, unsigned long int);
9715 char *ngettext_l(const char *, const char *,
9716                 unsigned long int, locale_t);
9717 char *textdomain(const char *);
```

9718 **APPLICATION USAGE**

9719 None.

9720 **RATIONALE**9721 Some historical implementations defined TEXTDOMAINMAX in this header. This standard  
9722 instead defines {TEXTDOMAIN\_MAX} in **<limits.h>**. This was done to allow the maximum  
9723 length of a text domain name to vary depending on the filesystem type used to store message  
9724 catalogs. Implementations are allowed to continue to define TEXTDOMAINMAX in this header  
9725 as an extension to the standard (see XSH [Section 2.2.2](#), on page 498).9726 **FUTURE DIRECTIONS**

9727 None.

9728 **SEE ALSO**9729 [<locale.h>](#)9730 XSH *gettext*, *bindtextdomain()*



9731 **CHANGE HISTORY**  
9732 First released in Issue 8.

9733 **NAME**9734 `limits.h` — implementation-defined constants9735 **SYNOPSIS**9736 `#include <limits.h>`9737 **DESCRIPTION**

9738 CX Some of the functionality described on this reference page extends the ISO C standard.  
9739 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
9740 enable the visibility of these symbols in this header.

9741 Many of the symbols listed here are not defined by the ISO C standard. Such symbols are not  
9742 shown as CX shaded, except under the heading “Numerical Limits”.

9743 The **<limits.h>** header shall define macros and symbolic constants for various limits. Different  
9744 categories of limits are described below, representing various limits on resources that the  
9745 implementation imposes on applications. All macros and symbolic constants defined in this  
9746 header shall be suitable for use in **#if** preprocessing directives.

9747 Implementations may choose any appropriate value for each limit, provided it is not more  
9748 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names  
9749 beginning with `_POSIX` may be found in **<unistd.h>**.

9750 Applications should not assume any particular value for a limit. To achieve maximum  
9751 portability, an application should not require more resource than the Minimum Acceptable  
9752 Value quantity. However, an application wishing to avail itself of the full amount of a resource  
9753 available on an implementation may make use of the value given in **<limits.h>** on that  
9754 particular implementation, by using the macros and symbolic constants listed below. It should  
9755 be noted, however, that many of the listed limits are not invariant, and at runtime, the value of  
9756 the limit may differ from those given in this header, for the following reasons:

- 9757 • The limit is pathname-dependent.
- 9758 • The limit differs between the compile and runtime machines.
- 9759 • The limit has been changed at runtime by an application (see `setrlimit()`).

9760 For these reasons, an application can use the `fpathconf()`, `getrlimit()`, `pathconf()`, and `sysconf()`  
9761 functions to determine the actual value of a limit at runtime.

9762 The items in the list ending in `_MIN` give the most negative values that the mathematical types  
9763 are guaranteed to be capable of representing. Numbers of a more negative value may be  
9764 supported on some implementations, as indicated by the **<limits.h>** header on the  
9765 implementation, but applications requiring such numbers are not guaranteed to be portable to  
9766 all implementations. For positive constants ending in `_MIN`, this indicates the minimum  
9767 acceptable value.

9768 **Runtime Invariant Values (Possibly Indeterminate)**

9769 A definition of one of the symbolic constants in the following list shall be omitted from  
9770 **<limits.h>** on specific implementations where the corresponding value is equal to or greater  
9771 than the stated minimum, but is unspecified.

9772 This indetermination might depend on the amount of available memory space on a specific  
9773 instance of a specific implementation. The actual value supported by a specific instance shall be  
9774 provided by the `sysconf()` function.

9775 {AIO\_LISTIO\_MAX}  
 9776 Maximum number of I/O operations in a single list I/O call supported by the  
 9777 implementation.  
 9778 Minimum Acceptable Value: {\_POSIX\_AIO\_LISTIO\_MAX}

9779 {AIO\_MAX}  
 9780 Maximum number of outstanding asynchronous I/O operations supported by the  
 9781 implementation.  
 9782 Minimum Acceptable Value: {\_POSIX\_AIO\_MAX}

9783 {AIO\_PRIO\_DELTA\_MAX}  
 9784 The maximum amount by which a process can decrease its asynchronous I/O priority level  
 9785 from its own scheduling priority.  
 9786 Minimum Acceptable Value: 0

9787 {ARG\_MAX}  
 9788 Maximum length of argument to the *exec* functions including environment data.  
 9789 Minimum Acceptable Value: {\_POSIX\_ARG\_MAX}

9790 {ATEXIT\_MAX}  
 9791 Maximum number of functions that can be registered with *atexit()* or *at\_quick\_exit()*. The  
 9792 limit shall apply independently to each function.  
 9793 Minimum Acceptable Value: 32

9794 {CHILD\_MAX}  
 9795 Maximum number of simultaneous processes per real user ID.  
 9796 Minimum Acceptable Value: {\_POSIX\_CHILD\_MAX}

9797 {DELAYTIMER\_MAX}  
 9798 Maximum number of timer expiration overruns.  
 9799 Minimum Acceptable Value: {\_POSIX\_DELAYTIMER\_MAX}

9800 {HOST\_NAME\_MAX}  
 9801 Maximum length of a host name (not including the terminating null) as returned from the  
 9802 *gethostname()* function.  
 9803 Minimum Acceptable Value: {\_POSIX\_HOST\_NAME\_MAX}

9804 XSI {IOV\_MAX}  
 9805 Maximum number of *iovec* structures that one process has available for use with *readv()* or  
 9806 *writev()*.  
 9807 Minimum Acceptable Value: {\_XOPEN\_IOV\_MAX}

9808 {LOGIN\_NAME\_MAX}  
 9809 Maximum length of a login name.  
 9810 Minimum Acceptable Value: {\_POSIX\_LOGIN\_NAME\_MAX}

9811 MSG {MQ\_OPEN\_MAX}  
 9812 The maximum number of open message queue descriptors a process may hold.  
 9813 Minimum Acceptable Value: {\_POSIX\_MQ\_OPEN\_MAX}

9814 MSG {MQ\_PRIO\_MAX}  
 9815 The maximum number of message priorities supported by the implementation.  
 9816 Minimum Acceptable Value: {\_POSIX\_MQ\_PRIO\_MAX}

9817 {OPEN\_MAX}  
 9818 A value one greater than the maximum value that the system may assign to a newly-created  
 9819 file descriptor.  
 9820 Minimum Acceptable Value: {\_POSIX\_OPEN\_MAX}

9821           {PAGESIZE}  
9822            Size in bytes of a page.  
9823            Minimum Acceptable Value: 1

9824 XSI        {PAGE\_SIZE}  
9825            Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE\_SIZE} is defined, the other is  
9826            defined with the same value.

9827           {PTHREAD\_DESTRUCTOR\_ITERATIONS}  
9828            Maximum number of attempts made to destroy a thread's thread-specific data values on  
9829            thread exit.  
9830            Minimum Acceptable Value: {\_POSIX\_THREAD\_DESTRUCTOR\_ITERATIONS}

9831           {PTHREAD\_KEYS\_MAX}  
9832            Maximum number of data keys that can be created by a process.  
9833            Minimum Acceptable Value: {\_POSIX\_THREAD\_KEYS\_MAX}

9834           {PTHREAD\_STACK\_MIN}  
9835            Minimum size in bytes of thread stack storage.  
9836            Minimum Acceptable Value: 0

9837           {PTHREAD\_THREADS\_MAX}  
9838            Maximum number of threads that can be created per process.  
9839            Minimum Acceptable Value: {\_POSIX\_THREAD\_THREADS\_MAX}

9840           {RTSIG\_MAX}  
9841            Maximum number of realtime signals reserved for application use in this implementation.  
9842            Minimum Acceptable Value: {\_POSIX\_RTSIG\_MAX}

9843           {SEM\_NSEMS\_MAX}  
9844            Maximum number of semaphores that a process may have.  
9845            Minimum Acceptable Value: {\_POSIX\_SEM\_NSEMS\_MAX}

9846           {SEM\_VALUE\_MAX}  
9847            The maximum value a semaphore may have.  
9848            Minimum Acceptable Value: {\_POSIX\_SEM\_VALUE\_MAX}

9849           {SIGQUEUE\_MAX}  
9850            Maximum number of queued signals that a process may send and have pending at the  
9851            receiver(s) at any time.  
9852            Minimum Acceptable Value: {\_POSIX\_SIGQUEUE\_MAX}

9853 SS|TSP     {SS\_REPL\_MAX}  
9854            The maximum number of replenishment operations that may be simultaneously pending  
9855            for a particular sporadic server scheduler.  
9856            Minimum Acceptable Value: {\_POSIX\_SS\_REPL\_MAX}

9857           {STREAM\_MAX}  
9858            Maximum number of streams that one process can have open at one time. If defined, it has  
9859            the same value as {FOPEN\_MAX} (see <stdio.h>).  
9860            Minimum Acceptable Value: {\_POSIX\_STREAM\_MAX}

9861           {SYMLOOP\_MAX}  
9862            Maximum number of symbolic links that can be reliably traversed in the resolution of a  
9863            pathname in the absence of a loop.  
9864            Minimum Acceptable Value: {\_POSIX\_SYMLOOP\_MAX}

9865 {TIMER\_MAX}  
 9866 Maximum number of timers per process supported by the implementation.  
 9867 Minimum Acceptable Value: {\_POSIX\_TIMER\_MAX}

9868 {TTY\_NAME\_MAX}  
 9869 Maximum length of terminal device name.  
 9870 Minimum Acceptable Value: {\_POSIX\_TTY\_NAME\_MAX}

9871 {TZNAME\_MAX}  
 9872 Maximum number of bytes supported for the name of a timezone (not of the *TZ* variable).  
 9873 Minimum Acceptable Value: {\_POSIX\_TZNAME\_MAX}

9874 **Note:** The length given by {TZNAME\_MAX} does not include the quoting characters mentioned in  
 9875 [Section 8.3](#) (on page 174).

9876 **Pathname Variable Values**

9877 The values in the following list may be constants within an implementation or may vary from  
 9878 one pathname to another. For example, file systems or directories may have different  
 9879 characteristics.

9880 A definition of one of the symbolic constants in the following list shall be omitted from the  
 9881 <limits.h> header on specific implementations where the corresponding value is equal to or  
 9882 greater than the stated minimum, but where the value can vary depending on the file to which it  
 9883 is applied. The actual value supported for a specific pathname shall be provided by the  
 9884 *pathconf()* function.

9885 {FILESIZEBITS}  
 9886 Minimum number of bits needed to represent, as a signed integer value, the maximum size  
 9887 of a regular file allowed in the specified directory.  
 9888 Minimum Acceptable Value: 32

9889 {LINK\_MAX}  
 9890 Maximum number of links to a single file.  
 9891 Minimum Acceptable Value: {\_POSIX\_LINK\_MAX}

9892 {MAX\_CANON}  
 9893 Maximum number of bytes in a terminal canonical input line.  
 9894 Minimum Acceptable Value: {\_POSIX\_MAX\_CANON}

9895 {MAX\_INPUT}  
 9896 Minimum number of bytes for which space is available in a terminal input queue; therefore,  
 9897 the maximum number of bytes a conforming application may require to be typed as input  
 9898 before reading them.  
 9899 Minimum Acceptable Value: {\_POSIX\_MAX\_INPUT}

9900 {NAME\_MAX}  
 9901 Maximum number of bytes in a filename (not including the terminating null of a filename  
 9902 string).  
 9903 Minimum Acceptable Value: {\_POSIX\_NAME\_MAX}  
 9904 XSI Minimum Acceptable Value: {\_XOPEN\_NAME\_MAX}

9905 {PATH\_MAX}  
 9906 Maximum number of bytes the implementation stores as a pathname in a user-supplied  
 9907 buffer of unspecified size, including the terminating null character. Minimum number the  
 9908 implementation shall accept as the maximum number of bytes in a pathname.  
 9909 Minimum Acceptable Value: {\_POSIX\_PATH\_MAX}

9910	XSI	<b>Minimum Acceptable Value: {_XOPEN_PATH_MAX}</b>
9911		{PIPE_BUF}
9912		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
9913		Minimum Acceptable Value: {_POSIX_PIPE_BUF}
9914	ADV	<b>{POSIX_ALLOC_SIZE_MIN}</b>
9915		Minimum number of bytes of storage actually allocated for any portion of a file.
9916		Minimum Acceptable Value: Not specified.
9917	ADV	<b>{POSIX_REC_INCR_XFER_SIZE}</b>
9918		Recommended increment for file transfer sizes between the
9919		{POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
9920		Minimum Acceptable Value: Not specified.
9921	ADV	<b>{POSIX_REC_MAX_XFER_SIZE}</b>
9922		Maximum recommended file transfer size.
9923		Minimum Acceptable Value: Not specified.
9924	ADV	<b>{POSIX_REC_MIN_XFER_SIZE}</b>
9925		Minimum recommended file transfer size.
9926		Minimum Acceptable Value: Not specified.
9927	ADV	<b>{POSIX_REC_XFER_ALIGN}</b>
9928		Recommended file transfer buffer alignment.
9929		Minimum Acceptable Value: Not specified.
9930		{SYMLINK_MAX}
9931		Maximum number of bytes in a symbolic link.
9932		Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}
9933		{TEXTDOMAIN_MAX}
9934		Maximum length of a text domain name, not including the terminating null byte.
9935		Minimum Acceptable Value: {_POSIX_NAME_MAX} – 3
9936	XSI	<b>Minimum Acceptable Value: {_XOPEN_NAME_MAX} – 3</b>

### 9937 Runtime Inceasable Values

9938 The magnitude limitations in the following list shall be fixed by specific implementations. An  
 9939 application should assume that the value of the symbolic constant defined by **<limits.h>** in a  
 9940 specific implementation is the minimum that pertains whenever the application is run under  
 9941 that implementation. A specific instance of a specific implementation may increase the value  
 9942 relative to that supplied by **<limits.h>** for that implementation. The actual value supported by a  
 9943 specific instance shall be provided by the *sysconf()* function.

9944		<b>{BC_BASE_MAX}</b>
9945		Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
9946		Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}
9947		<b>{BC_DIM_MAX}</b>
9948		Maximum number of elements permitted in an array by the <i>bc</i> utility.
9949		Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}
9950		<b>{BC_SCALE_MAX}</b>
9951		Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
9952		Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

9953 {BC\_STRING\_MAX}  
 9954 Maximum length of a string constant accepted by the *bc* utility.  
 9955 Minimum Acceptable Value: {\_POSIX2\_BC\_STRING\_MAX}

9956 {CHARCLASS\_NAME\_MAX}  
 9957 Maximum number of bytes in a character class name.  
 9958 Minimum Acceptable Value: {\_POSIX2\_CHARCLASS\_NAME\_MAX}

9959 {COLL\_WEIGHTS\_MAX}  
 9960 Maximum number of weights that can be assigned to an entry of the *LC\_COLLATE* **order**  
 9961 keyword in the locale definition file; see [Chapter 7](#) (on page 127).  
 9962 Minimum Acceptable Value: {\_POSIX2\_COLL\_WEIGHTS\_MAX}

9963 {EXPR\_NEST\_MAX}  
 9964 Maximum number of expressions that can be nested within parentheses by the *expr* utility.  
 9965 Minimum Acceptable Value: {\_POSIX2\_EXPR\_NEST\_MAX}

9966 {LINE\_MAX}  
 9967 Unless otherwise noted, the maximum length, in bytes, of a utility’s input line (either  
 9968 standard input or another file), when the utility is described as processing text files. The  
 9969 length includes room for the trailing <newline>.  
 9970 Minimum Acceptable Value: {\_POSIX2\_LINE\_MAX}

9971 {NGROUPS\_MAX}  
 9972 Maximum number of simultaneous supplementary group IDs per process.  
 9973 Minimum Acceptable Value: {\_POSIX\_NGROUPS\_MAX}

9974 {RE\_DUP\_MAX}  
 9975 Maximum number of repeated occurrences of a BRE or ERE interval expression; see [Section](#)  
 9976 [9.3.6](#) (on page 185) and [Section 9.4.6](#) (on page 189).  
 9977 Minimum Acceptable Value: {\_POSIX\_RE\_DUP\_MAX}

9978 **Maximum Values**

9979 The <limits.h> header shall define the following symbolic constants with the values shown.  
 9980 These are the most restrictive values for certain features on an implementation. A conforming  
 9981 implementation shall provide values no larger than these values. A conforming application shall  
 9982 not require a smaller value for correct operation.

9983 {\_POSIX\_CLOCKRES\_MIN}  
 9984 The resolution of the CLOCK\_REALTIME and CLOCK\_MONOTONIC clocks, in  
 9985 nanoseconds.  
 9986 Value: 20 000 000

9987 **Minimum Values**

9988 The <limits.h> header shall define the following symbolic constants with the values shown.  
 9989 These are the most restrictive values for certain features on an implementation conforming to  
 9990 this volume of POSIX.1-2024. Related symbolic constants are defined elsewhere in this volume  
 9991 of POSIX.1-2024 which reflect the actual implementation and which need not be as restrictive.  
 9992 For each of these limits, a conforming implementation shall provide a value at least this large or  
 9993 shall have no limit. A strictly conforming application shall not require a larger value for correct  
 9994 operation.

9995 {\_POSIX\_AIO\_LISTIO\_MAX}  
 9996 The number of I/O operations that can be specified in a list I/O call.  
 9997 Value: 2

9998		{_POSIX_AIO_MAX}
9999		The number of outstanding asynchronous I/O operations.
10000		Value: 1
10001		{_POSIX_ARG_MAX}
10002		Maximum length of argument to the <i>exec</i> functions including environment data.
10003		Value: 4 096
10004		{_POSIX_CHILD_MAX}
10005		Maximum number of simultaneous processes per real user ID.
10006		Value: 25
10007		{_POSIX_DELAYTIMER_MAX}
10008		The number of timer expiration overruns.
10009		Value: 32
10010		{_POSIX_HOST_NAME_MAX}
10011		Maximum length of a host name (not including the terminating null) as returned from the <i>gethostname()</i> function.
10012		Value: 255
10013		
10014		{_POSIX_LINK_MAX}
10015		Maximum number of links to a single file.
10016		Value: 8
10017		{_POSIX_LOGIN_NAME_MAX}
10018		The size of the storage required for a login name, in bytes (including the terminating null).
10019		Value: 9
10020		{_POSIX_MAX_CANON}
10021		Maximum number of bytes in a terminal canonical input queue.
10022		Value: 255
10023		{_POSIX_MAX_INPUT}
10024		Maximum number of bytes allowed in a terminal input queue.
10025		Value: 255
10026	MSG	{_POSIX_MQ_OPEN_MAX}
10027		The number of message queues that can be open for a single process.
10028		Value: 8
10029	MSG	{_POSIX_MQ_PRIO_MAX}
10030		The maximum number of message priorities supported by the implementation.
10031		Value: 32
10032		{_POSIX_NAME_MAX}
10033		Maximum number of bytes in a filename (not including the terminating null of a filename string).
10034		Value: 14
10035		
10036		{_POSIX_NGROUPS_MAX}
10037		Maximum number of simultaneous supplementary group IDs per process.
10038		Value: 8
10039		{_POSIX_OPEN_MAX}
10040		A value one greater than the maximum value that the system may assign to a newly-created file descriptor.
10041		Value: 20
10042		



10043		{_POSIX_PATH_MAX}
10044		Minimum number the implementation shall accept as the maximum number of bytes in a
10045		pathname.
10046		Value: 256
10047		{_POSIX_PIPE_BUF}
10048		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
10049		Value: 512
10050		{_POSIX_RE_DUP_MAX}
10051		Maximum number of repeated occurrences of a BRE or ERE interval expression; see <a href="#">Section</a>
10052		<a href="#">9.3.6</a> (on page 185) and <a href="#">Section 9.4.6</a> (on page 189).
10053		Value: 255
10054		{_POSIX_RTSIG_MAX}
10055		The number of realtime signal numbers reserved for application use.
10056		Value: 8
10057		{_POSIX_SEM_NSEMS_MAX}
10058		The number of semaphores that a process may have.
10059		Value: 256
10060		{_POSIX_SEM_VALUE_MAX}
10061		The maximum value a semaphore may have.
10062		Value: 32 767
10063		{_POSIX_SIGQUEUE_MAX}
10064		The number of queued signals that a process may send and have pending at the receiver(s)
10065		at any time.
10066		Value: 32
10067		{_POSIX_SSIZE_MAX}
10068		The value that can be stored in an object of type <b>ssize_t</b> .
10069		Value: 32 767
10070	SS TSP	{_POSIX_SS_REPL_MAX}
10071		The number of replenishment operations that may be simultaneously pending for a
10072		particular sporadic server scheduler.
10073		Value: 4
10074		{_POSIX_STREAM_MAX}
10075		The number of streams that one process can have open at one time.
10076		Value: 8
10077		{_POSIX_SYMLINK_MAX}
10078		The number of bytes in a symbolic link.
10079		Value: 255
10080		{_POSIX_SYMLINK_MAX}
10081		The number of symbolic links that can be traversed in the resolution of a pathname in the
10082		absence of a loop.
10083		Value: 8
10084		{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
10085		The number of attempts made to destroy a thread's thread-specific data values on thread
10086		exit.
10087		Value: 4

10088	{_POSIX_THREAD_KEYS_MAX}
10089	The number of data keys per process.
10090	Value: 128
10091	{_POSIX_THREAD_THREADS_MAX}
10092	The number of threads per process.
10093	Value: 64
10094	{_POSIX_TIMER_MAX}
10095	The per-process number of timers.
10096	Value: 32
10097	{_POSIX_TTY_NAME_MAX}
10098	The size of the storage required for a terminal device name, in bytes (including the terminating null).
10099	
10100	Value: 9
10101	{_POSIX_TZNAME_MAX}
10102	Maximum number of bytes supported for the name of a timezone (not of the <i>TZ</i> variable).
10103	Value: 6
10104	<b>Note:</b> The length given by {_POSIX_TZNAME_MAX} does not include the quoting characters
10105	mentioned in <a href="#">Section 8.3</a> (on page 174).
10106	{_POSIX2_BC_BASE_MAX}
10107	Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
10108	Value: 99
10109	{_POSIX2_BC_DIM_MAX}
10110	Maximum number of elements permitted in an array by the <i>bc</i> utility.
10111	Value: 2 048
10112	{_POSIX2_BC_SCALE_MAX}
10113	Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
10114	Value: 99
10115	{_POSIX2_BC_STRING_MAX}
10116	Maximum length of a string constant accepted by the <i>bc</i> utility.
10117	Value: 1 000
10118	{_POSIX2_CHARCLASS_NAME_MAX}
10119	Maximum number of bytes in a character class name.
10120	Value: 14
10121	{_POSIX2_COLL_WEIGHTS_MAX}
10122	Maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b>
10123	keyword in the locale definition file; see <a href="#">Chapter 7</a> (on page 127).
10124	Value: 2
10125	{_POSIX2_EXPR_NEST_MAX}
10126	Maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.
10127	Value: 32
10128	{_POSIX2_LINE_MAX}
10129	Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
10130	standard input or another file), when the utility is described as processing text files. The
10131	length includes room for the trailing <newline>.
10132	Value: 2 048

10133            {`_POSIX2_RE_DUP_MAX`}

10134            Maximum number of repeated occurrences of a BRE or ERE interval expression; see [Section 9.3.6](#) (on page 185) and [Section 9.4.6](#) (on page 189).

10135            Value: 255

10136

10137 XSI         {`_XOPEN_IOV_MAX`}

10138            Maximum number of `iovec` structures that one process has available for use with `readv()` or `writev()`.

10139            Value: 16

10140

10141 XSI         {`_XOPEN_NAME_MAX`}

10142            Maximum number of bytes in a filename (not including the terminating null of a filename string).

10143            Value: 255

10144

10145 XSI         {`_XOPEN_PATH_MAX`}

10146            Minimum number the implementation shall accept as the maximum number of bytes in a pathname.

10147            Value: 1 024

10148

10149         **Numerical Limits**

10150         The <limits.h> header shall define the following macros and, except for {CHAR\_BIT}, {LONG\_BIT}, {MB\_LEN\_MAX}, and {WORD\_BIT}, they shall be replaced by expressions that have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions.

10151

10152         If an object of type `char` can hold negative values, the value of {CHAR\_MIN} shall be the same as that of {SCHAR\_MIN} and the value of {CHAR\_MAX} shall be the same as that of {SCHAR\_MAX}. Otherwise, the value of {CHAR\_MIN} shall be 0 and the value of {CHAR\_MAX} shall be the same as that of {UCHAR\_MAX}.

10153

10154         {CHAR\_BIT}

10155            Number of bits in a type `char`.

10156            Value: 8

10157

10158         {CHAR\_MAX}

10159            Maximum value for an object of type `char`.

10160            Value: {UCHAR\_MAX} or {SCHAR\_MAX}

10161

10162         {CHAR\_MIN}

10163            Minimum value for an object of type `char`.

10164            Value: {SCHAR\_MIN} or 0

10165

10166         {INT\_MAX}

10167            Maximum value for an object of type `int`.

10168            Minimum Acceptable Value: 2 147 483 647

10169

10170         {INT\_MIN}

10171            Minimum value for an object of type `int`.

10172            Maximum Acceptable Value: -2 147 483 648

10173

10174         {LLONG\_MAX}

10175            Maximum value for an object of type `long long`.

10176            Minimum Acceptable Value: +9 223 372 036 854 775 807

10176		{LLONG_MIN}
10177		Minimum value for an object of type <b>long long</b> .
10178	CX	Maximum Acceptable Value: -9 223 372 036 854 775 808
10179	CX	{LONG_BIT}
10180		Number of bits in an object of type <b>long</b> .
10181		Minimum Acceptable Value: 32
10182		{LONG_MAX}
10183		Maximum value for an object of type <b>long</b> .
10184		Minimum Acceptable Value: +2 147 483 647
10185		{LONG_MIN}
10186		Minimum value for an object of type <b>long</b> .
10187	CX	Maximum Acceptable Value: -2 147 483 648
10188		{MB_LEN_MAX}
10189		Maximum number of bytes in a character, for any supported locale.
10190		Minimum Acceptable Value: 1
10191		{SCHAR_MAX}
10192		Maximum value for an object of type <b>signed char</b> .
10193	CX	Value: +127
10194		{SCHAR_MIN}
10195		Minimum value for an object of type <b>signed char</b> .
10196	CX	Value: -128
10197		{SHRT_MAX}
10198		Maximum value for an object of type <b>short</b> .
10199		Minimum Acceptable Value: +32 767
10200		{SHRT_MIN}
10201		Minimum value for an object of type <b>short</b> .
10202	CX	Maximum Acceptable Value: -32 768
10203	CX	{SSIZE_MAX}
10204		Maximum value for an object of type <b>ssize_t</b> .
10205		Minimum Acceptable Value: {_POSIX_SSIZE_MAX}
10206		{UCHAR_MAX}
10207		Maximum value for an object of type <b>unsigned char</b> .
10208	CX	Value: 255
10209		{UINT_MAX}
10210		Maximum value for an object of type <b>unsigned</b> .
10211	CX	Minimum Acceptable Value: 4 294 967 295
10212		{ULLONG_MAX}
10213		Maximum value for an object of type <b>unsigned long long</b> .
10214		Minimum Acceptable Value: 18 446 744 073 709 551 615
10215		{ULONG_MAX}
10216		Maximum value for an object of type <b>unsigned long</b> .
10217		Minimum Acceptable Value: 4 294 967 295
10218		{USHRT_MAX}
10219		Maximum value for an object of type <b>unsigned short</b> .
10220		Minimum Acceptable Value: 65 535

10221	CX	<b>{WORD_BIT}</b>
10222		Number of bits in an object of type <b>int</b> .
10223		Minimum Acceptable Value: 32
10224		<b>Other Invariant Values</b>
10225		The <limits.h> header shall define the following symbolic constants:
10226		<b>{GETENTROPY_MAX}</b>
10227		The maximum value of the <i>length</i> argument in calls to the <i>getentropy()</i> function.
10228		Minimum Acceptable Value: 256
10229		<b>{NL_ARGMAX}</b>
10230		Maximum value of <i>n</i> in conversion specifications using the "%n\$" sequence in calls to the <i>printf()</i> and <i>scanf()</i> families of functions.
10231		
10232		Minimum Acceptable Value: 9
10233	XSI	<b>{NL_LANGMAX}</b>
10234		Maximum number of bytes in a <i>LANG</i> name.
10235		Minimum Acceptable Value: 14
10236		<b>{NL_MSGMAX}</b>
10237		Maximum message number.
10238		Minimum Acceptable Value: 32 767
10239		<b>{NL_SETMAX}</b>
10240		Maximum set number.
10241		Minimum Acceptable Value: 255
10242		<b>{NL_TEXTMAX}</b>
10243		Maximum number of bytes in a message string.
10244		Minimum Acceptable Value: {_POSIX2_LINE_MAX}
10245		<b>{NSIG_MAX}</b>
10246		Maximum possible return value of <i>sysconf(_SC_NSIG)</i> . See XSH <i>sysconf()</i> . The value of
10247		{NSIG_MAX} shall be no greater than the number of signals that the <b>sigset_t</b> type (see
10248		<signal.h>) is capable of representing, ignoring any restrictions imposed by <i>sigfillset()</i> or
10249		<i>sigaddset()</i> .
10250	XSI	<b>{NZERO}</b>
10251		Default process priority.
10252		Minimum Acceptable Value: 20

10253 **APPLICATION USAGE**

10254 None.

10255 **RATIONALE**

10256 A request was made to reduce the value of {\_POSIX\_LINK\_MAX} from the value of 8 specified  
 10257 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request  
 10258 for several reasons:

- 10259 • They wanted to avoid making any changes to the standard that could break conforming  
 10260 applications, and the requested change could have that effect.
- 10261 • The use of multiple hard links to a file cannot always be replaced with use of symbolic  
 10262 links. Symbolic links are semantically different from hard links in that they associate a  
 10263 pathname with another pathname rather than a pathname with a file. This has  
 10264 implications for access control, file permanence, and transparency.

10265 • The original standard developers had considered the issue of allowing for  
10266 implementations that did not in general support hard links, and decided that this would  
10267 reduce consensus on the standard.

10268 Systems that support historical versions of the development option of the ISO POSIX-2 standard  
10269 retain the name `{_POSIX2_RE_DUP_MAX}` as an alias for `{_POSIX_RE_DUP_MAX}`.

10270 `{NSIG_MAX}`

10271 Some historical implementations provided compile-time constants `NSIG` or `SIGMAX` to  
10272 define the maximum number of signals the implementation supported, but these values did  
10273 not necessarily reflect the number of signals that could be handled using a `sigset_t`. With  
10274 the addition of real-time signals and the desire by some applications to be able to allocate  
10275 additional real-time signals at run-time, neither of these constants provided a useable,  
10276 portable value. `{NSIG_MAX}` was added to the standard to allow applications to determine  
10277 the maximum number of signals that an implementation will support based on the size of  
10278 the `sigset_t` type (defined in **<signal.h>**).

10279 `{PATH_MAX}`

10280 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the  
10281 definition of `pathname` and the description of `{PATH_MAX}`, allowing application  
10282 developers to allocate either `{PATH_MAX}` or `{PATH_MAX}+1` bytes. The inconsistency has  
10283 been removed by correction to the `{PATH_MAX}` definition to include the null character.  
10284 With this change, applications that previously allocated `{PATH_MAX}` bytes will continue to  
10285 succeed.

10286 `{SYMLINK_MAX}`

10287 This symbol refers to space for data that is stored in the file system, as opposed to  
10288 `{PATH_MAX}` which is the length of a name that can be passed to a function. In some  
10289 existing implementations, the pathnames pointed to by symbolic links are stored in the  
10290 *inodes* of the links, so it is important that `{SYMLINK_MAX}` not be constrained to be as large  
10291 as `{PATH_MAX}`.

10292 The maximum values for `{SCHAR_MIN}`, `{SHRT_MIN}`, `{LONG_MIN}` and `{LLONG_MIN}`  
10293 differ from the ISO C standard because POSIX.1 requires two's complement representation for  
10294 the corresponding integer types. The maximum value for `{INT_MIN}` differs both for that reason  
10295 and because POSIX.1 requires that `int` has a width of at least 32 bits. See also the RATIONALE  
10296 section for **<stdint.h>**.

#### 10297 **FUTURE DIRECTIONS**

10298 None.

#### 10299 **SEE ALSO**

10300 [Chapter 7](#) (on page 127), **<stdint.h>**, **<stdio.h>**, **<unistd.h>**

10301 [XSH Section 2.2](#) (on page 496), `fpathconf()`, `getrlimit()`, `sysconf()`

#### 10302 **CHANGE HISTORY**

10303 First released in Issue 1.

#### 10304 **Issue 5**

10305 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
10306 Threads Extension.

10307 `{FILESIZEBITS}` is added for the Large File Summit extensions.

10308 The minimum acceptable values for `{INT_MAX}`, `{INT_MIN}`, and `{UINT_MAX}` are changed to  
10309 make 32-bit values the minimum requirement.

10310 The entry is restructured to improve readability.

10311 **Issue 6**

10312 The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR\_MIN},  
 10313 {INT\_MIN}, {LONG\_MIN}, {SCHAR\_MIN}, and {SHRT\_MIN} that these are maximum  
 10314 acceptable values.

10315 The following new requirements on POSIX implementations derive from alignment with the  
 10316 Single UNIX Specification:

- 10317 • The minimum value for {CHILD\_MAX} is 25. This is a FIPS requirement.
- 10318 • The minimum value for {OPEN\_MAX} is 20. This is a FIPS requirement.
- 10319 • The minimum value for {NGROUPS\_MAX} is 8. This is also a FIPS requirement.

10320 Symbolic constants are added for {\_POSIX\_SYMLINK\_MAX}, {\_POSIX\_SYMLOOP\_MAX},  
 10321 {\_POSIX\_RE\_DUP\_MAX}, {RE\_DUP\_MAX}, {SYMLOOP\_MAX}, and {SYMLINK\_MAX}.

10322 The following values are added for alignment with IEEE Std 1003.1d-1999:

10323 {\_POSIX\_SS\_REPL\_MAX}  
 10324 {SS\_REPL\_MAX}  
 10325 {POSIX\_ALLOC\_SIZE\_MIN}  
 10326 {POSIX\_REC\_INCR\_XFER\_SIZE}  
 10327 {POSIX\_REC\_MAX\_XFER\_SIZE}  
 10328 {POSIX\_REC\_MIN\_XFER\_SIZE}  
 10329 {POSIX\_REC\_XFER\_ALIGN}

10330 Reference to CLOCK\_MONOTONIC is added in the description of {\_POSIX\_CLOCKRES\_MIN}  
 10331 for alignment with IEEE Std 1003.1j-2000.

10332 The constants {LLONG\_MIN}, {LLONG\_MAX}, and {ULLONG\_MAX} are added for alignment  
 10333 with the ISO/IEC 9899:1999 standard.

10334 The following values are added for alignment with IEEE Std 1003.1q-2000:

10335 {\_POSIX\_TRACE\_EVENT\_NAME\_MAX}  
 10336 {\_POSIX\_TRACE\_NAME\_MAX}  
 10337 {\_POSIX\_TRACE\_SYS\_MAX}  
 10338 {\_POSIX\_TRACE\_USER\_EVENT\_MAX}  
 10339 {TRACE\_EVENT\_NAME\_MAX}  
 10340 {TRACE\_NAME\_MAX}  
 10341 {TRACE\_SYS\_MAX}  
 10342 {TRACE\_USER\_EVENT\_MAX}

10343 The new limits {\_XOPEN\_NAME\_MAX} and {\_XOPEN\_PATH\_MAX} are added as minimum  
 10344 values for {PATH\_MAX} and {NAME\_MAX} limits on XSI-conformant systems.

10345 The LEGACY symbols {PASS\_MAX} and {TMP\_MAX} are removed.

10346 The values for the limits {CHAR\_BIT}, {SCHAR\_MAX}, and {UCHAR\_MAX} are now required  
 10347 to be 8, +127, and 255, respectively.

10348 The value for the limit {CHAR\_MAX} is now {UCHAR\_MAX} or {SCHAR\_MAX}.

10349 The value for the limit {CHAR\_MIN} is now {SCHAR\_MIN} or zero.

10350 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of  
 10351 {\_POSIX\_CHILD\_MAX} from 6 to 25. This is for FIPS 151-2 alignment.

10352 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/19 is applied, updating the values for  
10353 {INT\_MAX}, {UINT\_MAX}, and {INT\_MIN} to be CX extensions over the ISO C standard, and  
10354 correcting {WORD\_BIT} from 16 to 32.

10355 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/20 is applied, removing  
10356 {CHARCLASS\_NAME\_MAX} from the “Other Invariant Values” section (it also occurs under  
10357 “Runtime Increaseable Values”).

**Issue 7**

10358 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.

10360 Austin Group Interpretation 1003.1-2001 #173 is applied, updating the descriptions of  
10361 {TRACE\_EVENT\_NAME\_MAX} and {TRACE\_NAME\_MAX} to not include the terminating  
10362 null.

10363 SD5-XBD-ERN-36 is applied, changing the description of {RE\_DUP\_MAX}.

10364 SD5-XBD-ERN-90 is applied.

10365 {NL\_NMAX} is removed; it should have been removed in Issue 6.

10366 The Trace option values are marked obsolescent.

10367 The {ATEXIT\_MAX}, {LONG\_BIT}, {NL\_MSGMAX}, {NL\_SETMAX}, {NL\_TEXTMAX}, and  
10368 {WORD\_BIT} values are moved from the XSI option to the Base.

10369 The AIO\_\* and \_POSIX\_AIO\_\* values are moved from the Asynchronous Input and Output  
10370 option to the Base.

10371 The {\_POSIX\_RTSIG\_MAX}, {\_POSIX\_SIGQUEUE\_MAX}, {RTSIG\_MAX}, and  
10372 {SIGQUEUE\_MAX} values are moved from the Realtime Signals Extension option to the Base.

10373 Functionality relating to the Threads and Timers options is moved to the Base.

10374 This reference page is clarified with respect to macros and symbolic constants.

10375 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0052 [108], XBD/TC1-2008/0053 [291],  
10376 XBD/TC1-2008/0054 [182,427], XBD/TC1-2008/0055 [291], XBD/TC1-2008/0056 [371],  
10377 XBD/TC1-2008/0057 [291], XBD/TC1-2008/0058 [108], and XBD/TC1-2008/0059 [291] are  
10378 applied.

10379 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0061 [666] is applied.

**Issue 8**

10380 Austin Group Defect 741 is applied, adding {NSIG\_MAX}.

10382 Austin Group Defect 1108 is applied, changing the maximum allowed value for all signed  
10383 integer minimum limits.

10384 Austin Group Defect 1122 is applied, adding {TEXTDOMAIN\_MAX}.

10385 Austin Group Defect 1134 is applied, adding {GETENTROPY\_MAX}.

10386 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.

10387 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

10388 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

10389 Austin Group Defect 1446 is applied, changing the introductory paragraphs of the  
10390 DESCRIPTION to include mention of *setrlimit()* and *getrlimit()*.



10391 **NAME**

10392 locale.h — category macros

10393 **SYNOPSIS**

10394 #include <locale.h>

10395 **DESCRIPTION**

10396 CX Some of the functionality described on this reference page extends the ISO C standard.  
 10397 Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 496) to  
 10398 enable the visibility of these symbols in this header.

10399 The <locale.h> header shall define the **Iconv** structure, which shall include at least the following  
 10400 members. (See the definitions of *LC\_MONETARY* in [Section 7.3.3](#) (on page 147) and [Section 7.3.4](#)  
 10401 (on page 151).)

- 10402 char \*currency\_symbol
- 10403 char \*decimal\_point
- 10404 char frac\_digits
- 10405 char \*grouping
- 10406 char \*int\_curr\_symbol
- 10407 char int\_frac\_digits
- 10408 char int\_n\_cs\_precedes
- 10409 char int\_n\_sep\_by\_space
- 10410 char int\_n\_sign\_posn
- 10411 char int\_p\_cs\_precedes
- 10412 char int\_p\_sep\_by\_space
- 10413 char int\_p\_sign\_posn
- 10414 char \*mon\_decimal\_point
- 10415 char \*mon\_grouping
- 10416 char \*mon\_thousands\_sep
- 10417 char \*negative\_sign
- 10418 char n\_cs\_precedes
- 10419 char n\_sep\_by\_space
- 10420 char n\_sign\_posn
- 10421 char \*positive\_sign
- 10422 char p\_cs\_precedes
- 10423 char p\_sep\_by\_space
- 10424 char p\_sign\_posn
- 10425 char \*thousands\_sep

10426 The <locale.h> header shall define NULL (as described in <stddef.h>) and at least the following  
 10427 as macros:

- 10428 LC\_ALL
- 10429 LC\_COLLATE
- 10430 LC\_CTYPE
- 10431 CX LC\_MESSAGES
- 10432 LC\_MONETARY
- 10433 LC\_NUMERIC
- 10434 LC\_TIME

10435 which shall expand to integer constant expressions with distinct values for use as the first  
 10436 argument to the *setlocale()* function.

10437 Additional macro definitions, beginning with the characters *LC\_* and an uppercase letter, may  
 10438 also be specified by the implementation.

10439 CX The **<locale.h>** header shall contain at least the following macros representing bitmasks for use  
 10440 with the *newlocale()* function for each supported locale category:

10441 LC\_COLLATE\_MASK  
 10442 LC\_CTYPE\_MASK  
 10443 LC\_MESSAGES\_MASK  
 10444 LC\_MONETARY\_MASK  
 10445 LC\_NUMERIC\_MASK  
 10446 LC\_TIME\_MASK

10447 In addition, a macro to set the bits for all categories set shall be defined:

10448 LC\_ALL\_MASK

10449 The **<locale.h>** header shall define LC\_GLOBAL\_LOCALE, a special locale object descriptor  
 10450 used by the *duplocale()* and *uselocale()* functions.

10451 The **<locale.h>** header shall define the **locale\_t** type, representing a locale object.

10452 The following shall be declared as functions and may also be defined as macros. Function  
 10453 prototypes shall be provided for use with ISO C standard compilers.

```
10454 CX locale_t      duplocale(locale_t);
10455 void          freelocale(locale_t);
10456 const char    *getlocalename_l(int, locale_t);
10457 struct lconv  *localeconv(void);
10458 CX locale_t    newlocale(int, const char *, locale_t);
10459 char          *setlocale(int, const char *);
10460 CX locale_t    uselocale (locale_t);
```

#### 10461 APPLICATION USAGE

10462 None.

#### 10463 RATIONALE

10464 It is suggested that each category macro name for use in *setlocale()* have a corresponding macro  
 10465 name ending in *\_MASK* for use in *newlocale()*.

#### 10466 FUTURE DIRECTIONS

10467 None.

#### 10468 SEE ALSO

10469 [Chapter 8](#) (on page 167), **<stddef.h>**

10470 XSH *duplocale()*, *freelocale()*, *getlocalename\_l()*, *localeconv()*, *newlocale()*, *setlocale()*, *uselocale()*

#### 10471 CHANGE HISTORY

10472 First released in Issue 3.

10473 Included for alignment with the ISO C standard.

#### 10474 Issue 6

10475 The **lconv** structure is expanded with new members (**int\_n\_cs\_precedes**, **int\_n\_sep\_by\_space**,  
 10476 **int\_n\_sign\_posn**, **int\_p\_cs\_precedes**, **int\_p\_sep\_by\_space**, and **int\_p\_sign\_posn**) for alignment  
 10477 with the ISO/IEC 9899:1999 standard.

10478 Extensions beyond the ISO C standard are marked.

10479 **Issue 7**

10480 The *duplocale()*, *freelocale()*, *newlocale()*, and *uselocale()* functions are added from The Open  
10481 Group Technical Standard, 2006, Extended API Set Part 4.

10482 This reference page is clarified with respect to macros and symbolic constants.

10483 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0060 [301,427] is applied.

10484 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0062 [781] is applied.

10485 **Issue 8**

10486 Austin Group Defect 1220 is applied, adding *getlocalename\_1()*.

10487 **NAME**

10488           math.h — mathematical declarations

10489 **SYNOPSIS**

10490           #include &lt;math.h&gt;

10491 **DESCRIPTION**

10492 CX       Some of the functionality described on this reference page extends the ISO C standard.  
10493       Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 496) to  
10494       enable the visibility of these symbols in this header.

10495       The **<math.h>** header shall define at least the following types:

10496       **float\_t**           A real-floating type at least as wide as **float**.

10497       **double\_t**          A real-floating type at least as wide as **double**, and at least as wide as **float\_t**.

10498       If FLT\_EVAL\_METHOD equals 0, **float\_t** and **double\_t** shall be **float** and **double**, respectively; if  
10499       FLT\_EVAL\_METHOD equals 1, they shall both be **double**; if FLT\_EVAL\_METHOD equals 2,  
10500       they shall both be **long double**; for other values of FLT\_EVAL\_METHOD, they are otherwise  
10501       implementation-defined.

10502       The **<math.h>** header shall define the following macros, where real-floating indicates that the  
10503       argument shall be an expression of real-floating type:

```
10504       int fpclassify(real-floating x);  
10505       int isfinite(real-floating x);  
10506       int isgreater(real-floating x, real-floating y);  
10507       int isgreaterequal(real-floating x, real-floating y);  
10508       int isinf(real-floating x);  
10509       int isless(real-floating x, real-floating y);  
10510       int islessequal(real-floating x, real-floating y);  
10511       int islessgreater(real-floating x, real-floating y);  
10512       int isnan(real-floating x);  
10513       int isnormal(real-floating x);  
10514       int isunordered(real-floating x, real-floating y);  
10515       int signbit(real-floating x);
```

10516 XSI      The **<math.h>** header shall define the following symbolic constants. Where the constant name  
10517      ends in 'l' (lower case ell), the values shall have type **long double**; otherwise the values shall  
10518      have type **double**. The values shall be accurate to the precision of their type. If the  
10519      implementation supports FLT\_EVAL\_METHOD values other than 0 or 1, the values shall either  
10520      include an explicit cast for that type, or be expressed as hexadecimal floating constants.

	Double	Long Double	Value
10521	M_E	M_El	Value of $e$
10522	M_EGAMMA	M_EGAMMAl	Value of $\gamma$ , Euler-Mascheroni constant
10523	M_LOG2E	M_LOG2El	Value of $\log_2 e$
10524	M_LOG10E	M_LOG10El	Value of $\log_{10} e$
10525	M_LN2	M_LN2l	Value of $\log_e 2$
10526	M_LN10	M_LN10l	Value of $\log_e 10$
10527	M_PHI	M_PHIl	Value of $\phi$ , $(1 + \sqrt{5})/2$ , golden ratio constant
10528	M_PI	M_PIl	Value of $\pi$
10529	M_PI_2	M_PI_2l	Value of $\pi/2$
10530	M_PI_4	M_PI_4l	Value of $\pi/4$
10531	M_1_PI	M_1_PIl	Value of $1/\pi$
10532	M_1_SQRTPI	M_1_SQRTPIl	Value of $1/\sqrt{\pi}$
10533	M_2_PI	M_2_PIl	Value of $2/\pi$
10534	M_2_SQRTPI	M_2_SQRTPIl	Value of $2/\sqrt{\pi}$
10535	M_SQRT2	M_SQRT2l	Value of $\sqrt{2}$
10536	M_SQRT3	M_SQRT3l	Value of $\sqrt{3}$
10537	M_SQRT1_2	M_SQRT1_2l	Value of $1/\sqrt{2}$
10538	M_SQRT1_3	M_SQRT1_3l	Value of $1/\sqrt{3}$
10539			

10540 The <math.h> header shall define the following macros:

- 10541 HUGE\_VAL      A positive **double** constant expression, not necessarily representable as a
- 10542                    **float**. Used as an error value returned by the mathematics library.
- 10543                    HUGE\_VAL evaluates to +infinity on systems supporting IEEE Std 754-1985.
  
- 10544 HUGE\_VALF      A positive **float** constant expression. Used as an error value returned by the
- 10545                    mathematics library. HUGE\_VALF evaluates to +infinity on systems
- 10546                    supporting IEEE Std 754-1985.
  
- 10547 HUGE\_VALL      A positive **long double** constant expression. Used as an error value returned
- 10548                    by the mathematics library. HUGE\_VALL evaluates to +infinity on systems
- 10549                    supporting IEEE Std 754-1985.
  
- 10550 INFINITY        A constant expression of type **float** representing positive or unsigned infinity,
- 10551                    if available; else a positive constant of type **float** that overflows at translation
- 10552                    time.
  
- 10553 NAN             A constant expression of type **float** representing a quiet NaN. This macro is
- 10554                    only defined if the implementation supports quiet NaNs for the **float** type.

10555 The following macros shall be defined for number classification. They represent the mutually-  
 10556 exclusive kinds of floating-point values. They expand to integer constant expressions with  
 10557 distinct values. Additional implementation-defined floating-point classifications, with macro  
 10558 definitions beginning with FP\_ and an uppercase letter, may also be specified by the  
 10559 implementation.

- 10560                    FP\_INFINITE
- 10561                    FP\_NAN
- 10562                    FP\_NORMAL
- 10563                    FP\_SUBNORMAL
- 10564                    FP\_ZERO

10565 The following optional macros indicate whether the *fma()* family of functions are fast compared

10566 with direct code:

```
10567     FP_FAST_FMA
10568     FP_FAST_FMAF
10569     FP_FAST_FMAL
```

10570 If defined, the `FP_FAST_FMA` macro shall expand to the integer constant 1 and shall indicate  
 10571 that the `fma()` function generally executes about as fast as, or faster than, a multiply and an add  
 10572 of **double** operands. If undefined, the speed of execution is unspecified. The other macros have  
 10573 the equivalent meaning for the **float** and **long double** versions.

10574 The following macros shall expand to integer constant expressions whose values are returned by  
 10575 `ilogb(x)` if  $x$  is zero or NaN, respectively. The value of `FP_ILOGB0` shall be either `{INT_MIN}` or  
 10576 `-{INT_MAX}`. The value of `FP_ILOGBNAN` shall be either `{INT_MAX}` or `{INT_MIN}`.

```
10577     FP_ILOGB0
10578     FP_ILOGBNAN
```

10579 The following macros shall expand to the integer constants 1 and 2, respectively;

```
10580     MATH_ERRNO
10581     MATH_ERREXCEPT
```

10582 The following macro shall expand to an expression that has type **int** and the value  
 10583 `MATH_ERRNO`, `MATH_ERREXCEPT`, or the bitwise-inclusive OR of both:

```
10584     math_errhandling
```

10585 The value of `math_errhandling` is constant for the duration of the program. It is unspecified  
 10586 whether `math_errhandling` is a macro or an identifier with external linkage. If a macro definition  
 10587 is suppressed or a program defines an identifier with the name `math_errhandling`, the behavior  
 10588 is undefined. If the expression `(math_errhandling & MATH_ERREXCEPT)` can be non-zero, the  
 10589 implementation shall define the macros `FE_DIVBYZERO`, `FE_INVALID`, and `FE_OVERFLOW` in  
 10590 **<fenv.h>**.

10591 The following shall be declared as functions and may also be defined as macros. Function  
 10592 prototypes shall be provided.

```
10593     double      acos(double);
10594     float       acosf(float);
10595     double      acosh(double);
10596     float       acoshf(float);
10597     long double acoshl(long double);
10598     long double acosl(long double);
10599     double      asin(double);
10600     float       asinf(float);
10601     double      asinh(double);
10602     float       asinhf(float);
10603     long double asinhl(long double);
10604     long double asinl(long double);
10605     double      atan(double);
10606     double      atan2(double, double);
10607     float       atan2f(float, float);
10608     long double atan2l(long double, long double);
10609     float       atanf(float);
```

```
10610     double      atanh(double);
10611     float       atanhf(float);
10612     long double  atanhl(long double);
10613     long double  atanl(long double);
10614     double      cbrt(double);
10615     float       cbrtf(float);
10616     long double  cbrtl(long double);
10617     double      ceil(double);
10618     float       ceilf(float);
10619     long double  ceill(long double);
10620     double      copysign(double, double);
10621     float       copysignf(float, float);
10622     long double  copysignl(long double, long double);
10623     double      cos(double);
10624     float       cosf(float);
10625     double      cosh(double);
10626     float       coshf(float);
10627     long double  coshl(long double);
10628     long double  cosl(long double);
10629     double      erf(double);
10630     double      erfc(double);
10631     float       erfcf(float);
10632     long double  erfcl(long double);
10633     float       erff(float);
10634     long double  erfl(long double);
10635     double      exp(double);
10636     double      exp2(double);
10637     float       exp2f(float);
10638     long double  exp2l(long double);
10639     float       expf(float);
10640     long double  expl(long double);
10641     double      expm1(double);
10642     float       expm1f(float);
10643     long double  expm1l(long double);
10644     double      fabs(double);
10645     float       fabsf(float);
10646     long double  fabsl(long double);
10647     double      fdim(double, double);
10648     float       fdimf(float, float);
10649     long double  fdiml(long double, long double);
10650     double      floor(double);
10651     float       floorf(float);
10652     long double  floorl(long double);
10653     double      fma(double, double, double);
10654     float       fmaf(float, float, float);
10655     long double  fmal(long double, long double, long double);
10656     double      fmax(double, double);
10657     float       fmaxf(float, float);
10658     long double  fmaxl(long double, long double);
10659     double      fmin(double, double);
10660     float       fminf(float, float);
10661     long double  fminl(long double, long double);
```

```

10662     double      fmod(double, double);
10663     float       fmodf(float, float);
10664     long double  fmodl(long double, long double);
10665     double      frexp(double, int *);
10666     float       frexpf(float, int *);
10667     long double  frexpl(long double, int *);
10668     double      hypot(double, double);
10669     float       hypotf(float, float);
10670     long double  hypotl(long double, long double);
10671     int         ilogb(double);
10672     int         ilogbf(float);
10673     int         ilogbl(long double);
10674 XSI     double      j0(double);
10675     double      j1(double);
10676     double      jn(int, double);
10677     double      ldexp(double, int);
10678     float       ldexpf(float, int);
10679     long double  ldexpl(long double, int);
10680     double      lgamma(double);
10681     float       lgammaf(float);
10682     long double  lgammal(long double);
10683     long long   llrint(double);
10684     long long   llrintf(float);
10685     long long   llrintl(long double);
10686     long long   llround(double);
10687     long long   llroundf(float);
10688     long long   llroundl(long double);
10689     double      log(double);
10690     double      log10(double);
10691     float       log10f(float);
10692     long double  log10l(long double);
10693     double      log1p(double);
10694     float       log1pf(float);
10695     long double  log1pl(long double);
10696     double      log2(double);
10697     float       log2f(float);
10698     long double  log2l(long double);
10699     double      logb(double);
10700     float       logbf(float);
10701     long double  logbl(long double);
10702     float       logf(float);
10703     long double  logl(long double);
10704     long        lrint(double);
10705     long        lrintf(float);
10706     long        lrintl(long double);
10707     long        lround(double);
10708     long        lroundf(float);
10709     long        lroundl(long double);
10710     double      modf(double, double *);
10711     float       modff(float, float *);
10712     long double  modfl(long double, long double *);
10713     double      nan(const char *);

```



```

10714 float nanf(const char *);
10715 long double nanl(const char *);
10716 double nearbyint(double);
10717 float nearbyintf(float);
10718 long double nearbyintl(long double);
10719 double nextafter(double, double);
10720 float nextafterf(float, float);
10721 long double nextafterl(long double, long double);
10722 double nexttoward(double, long double);
10723 float nexttowardf(float, long double);
10724 long double nexttowardl(long double, long double);
10725 double pow(double, double);
10726 float powf(float, float);
10727 long double powl(long double, long double);
10728 double remainder(double, double);
10729 float remainderf(float, float);
10730 long double remainderl(long double, long double);
10731 double remquo(double, double, int *);
10732 float remquof(float, float, int *);
10733 long double remquol(long double, long double, int *);
10734 double rint(double);
10735 float rintf(float);
10736 long double rintl(long double);
10737 double round(double);
10738 float roundf(float);
10739 long double roundl(long double);
10740 double scalbn(double, long);
10741 float scalbnf(float, long);
10742 long double scalbnl(long double, long);
10743 double scalbn(double, int);
10744 float scalbnf(float, int);
10745 long double scalbnl(long double, int);
10746 double sin(double);
10747 float sinf(float);
10748 double sinh(double);
10749 float sinhlf(float);
10750 long double sinhl(long double);
10751 long double sinl(long double);
10752 double sqrt(double);
10753 float sqrtf(float);
10754 long double sqrtl(long double);
10755 double tan(double);
10756 float tanf(float);
10757 double tanh(double);
10758 float tanhlf(float);
10759 long double tanhl(long double);
10760 long double tanl(long double);
10761 double tgamma(double);
10762 float tgammaf(float);
10763 long double tgamma1(long double);
10764 double trunc(double);
10765 float truncf(float);

```

```

10766     long double trunc1(long double);
10767 XSI     double      y0(double);
10768     double      y1(double);
10769     double      yn(int, double);

```

10770 The following external variable shall be defined:

```

10771 XSI     extern int  signgam;

```

10772 The behavior of each of the functions defined in **<math.h>** is specified in the System Interfaces  
 10773 volume of POSIX.1-2024 for all representable values of its input arguments, except where stated  
 10774 otherwise. Each function shall execute as if it were a single operation without generating any  
 10775 externally visible exceptional conditions.

#### 10776 APPLICATION USAGE

10777 The FP\_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is  
 10778 off) the implementation to contract expressions. Each pragma can occur either outside external  
 10779 declarations or preceding all explicit declarations and statements inside a compound statement.  
 10780 When outside external declarations, the pragma takes effect from its occurrence until another  
 10781 FP\_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a  
 10782 compound statement, the pragma takes effect from its occurrence until another FP\_CONTRACT  
 10783 pragma is encountered (including within a nested compound statement), or until the end of the  
 10784 compound statement; at the end of a compound statement the state for the pragma is restored to  
 10785 its condition just before the compound statement. If this pragma is used in any other context, the  
 10786 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

#### 10787 RATIONALE

10788 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type  
 10789 **double**. All the names formed by appending 'f' or 'l' to a name in **<math.h>** were reserved  
 10790 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999  
 10791 standard provided for all three versions of math functions.

10792 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their  
 10793 capability is available through *sprintf()*.

10794 The requirement for an explicit cast or representation via hexadecimal floating constants is to  
 10795 guarantee that even when FLT\_EVAL\_METHOD is neither 0 nor 1, the expression  
 10796 (double)M\_PI == M\_PI will always hold true. Earlier versions of this standard did not make  
 10797 this requirement, because they lacked the 'l' (lower case ell) versions of the constants and were  
 10798 allowed to provide M\_PI with long double precision depending on FLT\_EVAL\_METHOD.

#### 10799 FUTURE DIRECTIONS

10800 None.

#### 10801 SEE ALSO

10802 **<float.h>**, **<stddef.h>**, **<sys/types.h>**

10803 XSH Section 2.2 (on page 496), *acos()*, *acosh()*, *asin()*, *asinh()*, *atan()*, *atan2()*, *atanh()*, *cbrt()*,  
 10804 *ceil()*, *copysign()*, *cos()*, *cosh()*, *erf()*, *erfc()*, *exp()*, *exp2()*, *expm1()*, *fabs()*, *fdim()*, *floor()*, *fma()*,  
 10805 *fmax()*, *fmin()*, *fmod()*, *fpclassify()*, *frexp()*, *hypot()*, *ilogb()*, *isfinite()*, *isgreater()*, *isinf()*, *isnan()*,  
 10806 *isnormal()*, *isunordered()*, *j0()*, *ldexp()*, *lgamma()*, *llrint()*, *llround()*, *log()*, *log10()*, *log1p()*, *log2()*,  
 10807 *logb()*, *lrint()*, *lround()*, *modf()*, *nan()*, *nearbyint()*, *nextafter()*, *pow()*, *remainder()*, *remquo()*,  
 10808 *rint()*, *round()*, *scalbln()*, *signbit()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *tgamma()*, *trunc()*, *y0()*

10809 **CHANGE HISTORY**

10810 First released in Issue 1.

10811 **Issue 6**

10812 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

10813 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/21 is applied, making it clear that the  
10814 meaning of the FP\_FAST\_FMA macro is unspecified if the macro is undefined.10815 **Issue 7**10816 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #47 (SD5-XBD-ERN-52) is applied,  
10817 clarifying the wording of the FP\_FAST\_FMA macro.

10818 The MAXFLOAT constant is marked obsolescent.

10819 This reference page is clarified with respect to macros and symbolic constants.

10820 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0063 [801] and XBD/TC2-2008/0064  
10821 [801] are applied.10822 **Issue 8**10823 Austin Group Defect 828 is applied, adding **long double** counterparts with names ending in  
10824 'l' to the **double** constants whose names begin with "M\_", and changing the representation  
10825 requirements for those constants.

10826 Austin Group Defect 1302 is applied, changing ``provides'' to ``provided''.

10827 Austin Group Defect 1330 is applied, removing the obsolescent MAXFLOAT.

10828 Austin Group Defect 1503 is applied, adding M\_1\_SQRTPI, M\_EGAMMA, M\_PHI, M\_SQRT1\_3,  
10829 and M\_SQRT3.

10830 **NAME**

10831           monetary.h — monetary types

10832 **SYNOPSIS**

10833           #include &lt;monetary.h&gt;

10834 **DESCRIPTION**10835           The **<monetary.h>** header shall define the **locale\_t** type as described in **<locale.h>**.10836           The **<monetary.h>** header shall define the **size\_t** type as described in **<stddef.h>**.10837           The **<monetary.h>** header shall define the **ssize\_t** type as described in **<sys/types.h>**.10838           The following shall be declared as functions and may also be defined as macros. Function  
10839           prototypes shall be provided for use with ISO C standard compilers.

10840           ssize\_t strfmon(char \*restrict, size\_t, const char \*restrict, ...);

10841           ssize\_t strfmon\_l(char \*restrict, size\_t, locale\_t,

10842                         const char \*restrict, ...);

10843 **APPLICATION USAGE**

10844           None.

10845 **RATIONALE**

10846           None.

10847 **FUTURE DIRECTIONS**

10848           None.

10849 **SEE ALSO**10850           [<locale.h>](#), [<stddef.h>](#), [<sys/types.h>](#)10851           XSH *strfmon()*10852 **CHANGE HISTORY**

10853           First released in Issue 4.

10854 **Issue 6**10855           The **restrict** keyword is added to the prototype for *strfmon()*.10856 **Issue 7**10857           The **<monetary.h>** header is moved from the XSI option to the Base.10858           The *strfmon\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
10859           Set Part 4.10860           A declaration for the **locale\_t** type is added.

10861 **NAME**10862           mqueue.h — message queues (**REALTIME**)10863 **SYNOPSIS**

10864 MSG       #include &lt;mqueue.h&gt;

10865 **DESCRIPTION**10866           The <mqueue.h> header shall define the **mqd\_t** type, which is used for message queue  
10867 descriptors. This is not an array type.10868           The <mqueue.h> header shall define the **size\_t** and **ssize\_t** types as described in <sys/types.h>.10869           The <mqueue.h> header shall define the **struct timespec** structure as described in <time.h>.10870           The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which  
10871 are described in the <signal.h> header.10872           The <mqueue.h> header shall define the **mq\_attr** structure, which is used in getting and setting  
10873 the attributes of a message queue. Attributes are initially set when the message queue is created.  
10874 An **mq\_attr** structure shall have at least the following fields:

10875	long	mq_flags	Message queue flags.
10876	long	mq_maxmsg	Maximum number of messages.
10877	long	mq_msgsize	Maximum message size.
10878	long	mq_curmsgs	Number of messages currently queued.

10879           The <mqueue.h> header shall define **O\_RDONLY**, **O\_WRONLY**, **O\_RDWR**, **O\_CREAT**, **O\_EXCL**,  
10880 and **O\_NONBLOCK** as described in <fcntl.h>.10881           The following shall be declared as functions and may also be defined as macros. Function  
10882 prototypes shall be provided.

```

10883 int      mq_close(mqd_t);
10884 int      mq_getattr(mqd_t, struct mq_attr *);
10885 int      mq_notify(mqd_t, const struct sigevent *);
10886 mqd_t    mq_open(const char *, int, ...);
10887 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
10888 int      mq_send(mqd_t, const char *, size_t, unsigned);
10889 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
10890                   struct mq_attr *restrict);
10891 ssize_t  mq_timedreceive(mqd_t, char *restrict, size_t,
10892                        unsigned *restrict, const struct timespec *restrict);
10893 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
10894                     const struct timespec *);
10895 int      mq_unlink(const char *);

```

10896           Inclusion of the <mqueue.h> header may make visible symbols defined in the headers  
10897 <fcntl.h>, <signal.h>, and <time.h>.

**10898 APPLICATION USAGE**

10899 None.

**10900 RATIONALE**

10901 None.

**10902 FUTURE DIRECTIONS**

10903 None.

**10904 SEE ALSO**

10905 [<fcntl.h>](#), [<signal.h>](#), [<sys/types.h>](#), [<time.h>](#)

10906 XSH [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#), [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#),  
10907 [mq\\_unlink\(\)](#)

**10908 CHANGE HISTORY**

10909 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**10910 Issue 6**

10911 The **<mqueue.h>** header is marked as part of the Message Passing option.

10912 The [mq\\_timedreceive\(\)](#) and [mq\\_timedsend\(\)](#) functions are added for alignment with IEEE Std  
10913 1003.1d-1999.

10914 The **restrict** keyword is added to the prototypes for [mq\\_setattr\(\)](#) and [mq\\_timedreceive\(\)](#).

**10915 Issue 7**

10916 Type and structure declarations are added.

**10917 Issue 8**

10918 Austin Group Defect 593 is applied, adding **O\_RDONLY**, **O\_WRONLY**, **O\_RDWR**, **O\_CREAT**,  
10919 **O\_EXCL**, and **O\_NONBLOCK**.

10920 Austin Group Defect 1282 is applied, removing the requirement for **<mqueue.h>** to define  
10921 **pthread\_attr\_t**.

10922 **NAME**

10923 ndbm.h — definitions for ndbm database operations

10924 **SYNOPSIS**10925 XSI `#include <ndbm.h>`10926 **DESCRIPTION**10927 The <ndbm.h> header shall define the **datum** type as a structure, which shall include at least the  
10928 following members:10929 `void *dptr` A pointer to the application's data.  
10930 `size_t dsize` The size of the object pointed to by *dptr*.10931 The <ndbm.h> header shall define the **size\_t** type as described in <stddef.h>.10932 The <ndbm.h> header shall define the **DBM** type.10933 The <ndbm.h> header shall define the following symbolic constants as possible values for the  
10934 *store\_mode* argument to *dbm\_store()*:10935 **DBM\_INSERT** Insertion of new entries only.10936 **DBM\_REPLACE** Allow replacing existing entries.10937 The following shall be declared as functions and may also be defined as macros. Function  
10938 prototypes shall be provided.10939 `int dbm_clearerr(DBM *);`  
10940 `void dbm_close(DBM *);`  
10941 `int dbm_delete(DBM *, datum);`  
10942 `int dbm_error(DBM *);`  
10943 `datum dbm_fetch(DBM *, datum);`  
10944 `datum dbm_firstkey(DBM *);`  
10945 `datum dbm_nextkey(DBM *);`  
10946 `DBM *dbm_open(const char *, int, mode_t);`  
10947 `int dbm_store(DBM *, datum, datum, int);`10948 The <ndbm.h> header shall define the **mode\_t** type through **typedef**, as described in  
10949 <sys/types.h>.10950 **APPLICATION USAGE**

10951 None.

10952 **RATIONALE**

10953 None.

10954 **FUTURE DIRECTIONS**

10955 None.

10956 **SEE ALSO**

10957 &lt;stddef.h&gt;, &lt;sys/types.h&gt;

10958 XSH *dbm\_clearerr()*10959 **CHANGE HISTORY**

10960 First released in Issue 4, Version 2.

10961 **Issue 5**

10962 References to the definitions of **size\_t** and **mode\_t** are added to the DESCRIPTION.

10963 **Issue 7**

10964 This reference page is clarified with respect to macros and symbolic constants.



10965 **NAME**

10966 net/if.h — sockets local interfaces

10967 **SYNOPSIS**

10968 #include &lt;net/if.h&gt;

10969 **DESCRIPTION**10970 The <net/if.h> header shall define the **if\_nameindex** structure, which shall include at least the  
10971 following members:10972 unsigned if\_index Numeric index of the interface.  
10973 char \*if\_name Null-terminated name of the interface.10974 The <net/if.h> header shall define the following symbolic constant for the length of a buffer  
10975 containing an interface name (including the terminating NULL character):10976 **IF\_NAMESIZE** Interface name length.10977 The following shall be declared as functions and may also be defined as macros. Function  
10978 prototypes shall be provided.10979 void if\_freenameindex(struct if\_nameindex \*);  
10980 char \*if\_indextoname(unsigned, char \*);  
10981 struct if\_nameindex \*if\_nameindex(void);  
10982 unsigned if\_nametoindex(const char \*);10983 **APPLICATION USAGE**

10984 None.

10985 **RATIONALE**

10986 None.

10987 **FUTURE DIRECTIONS**

10988 None.

10989 **SEE ALSO**10990 XSH [if\\_freenameindex\(\)](#), [if\\_indextoname\(\)](#), [if\\_nameindex\(\)](#), [if\\_nametoindex\(\)](#)10991 **CHANGE HISTORY**

10992 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10993 **Issue 7**

10994 This reference page is clarified with respect to macros and symbolic constants.

10995 **NAME**

10996 netdb.h — definitions for network database operations

10997 **SYNOPSIS**

10998 #include &lt;netdb.h&gt;

10999 **DESCRIPTION**11000 The **<netdb.h>** header shall define the **hostent** structure, which shall include at least the  
11001 following members:

11002	char	*h_name	Official name of the host.
11003	char	**h_aliases	A pointer to an array of pointers to 11004 alternative host names, terminated by a 11005 null pointer.
11006	int	h_addrtype	Address type.
11007	int	h_length	The length, in bytes, of the address.
11008	char	**h_addr_list	A pointer to an array of pointers to network 11009 addresses (in network byte order) for the host, 11010 terminated by a null pointer.

11011 The **<netdb.h>** header shall define the **netent** structure, which shall include at least the  
11012 following members:

11013	char	*n_name	Official, fully-qualified (including the 11014 domain) name of the host.
11015	char	**n_aliases	A pointer to an array of pointers to 11016 alternative network names, terminated by a 11017 null pointer.
11018	int	n_addrtype	The address type of the network.
11019	uint32_t	n_net	The network number, in host byte order.

11020 The **<netdb.h>** header shall define the **uint32\_t** type as described in **<inttypes.h>**.11021 The **<netdb.h>** header shall define the **protoent** structure, which shall include at least the  
11022 following members:

11023	char	*p_name	Official name of the protocol.
11024	char	**p_aliases	A pointer to an array of pointers to 11025 alternative protocol names, terminated by 11026 a null pointer.
11027	int	p_proto	The protocol number.

11028 The **<netdb.h>** header shall define the **servent** structure, which shall include at least the  
11029 following members:

11030	char	*s_name	Official name of the service.
11031	char	**s_aliases	A pointer to an array of pointers to 11032 alternative service names, terminated by 11033 a null pointer.
11034	int	s_port	A value which, when converted to <b>uint16_t</b> , 11035 yields the port number in network byte order 11036 at which the service resides.
11037	char	*s_proto	The name of the protocol to use when 11038 contacting the service.

11039 The **<netdb.h>** header shall define the **IPPORT\_RESERVED** symbolic constant with the value of  
11040 the highest reserved Internet port number.

11041 **Address Information Structure**

11042 The <netdb.h> header shall define the **addrinfo** structure, which shall include at least the  
 11043 following members:

11044	int	ai_flags	Input flags.
11045	int	ai_family	Address family of socket.
11046	int	ai_socktype	Socket type.
11047	int	ai_protocol	Protocol of socket.
11048	socklen_t	ai_addrlen	Length of socket address.
11049	struct sockaddr	*ai_addr	Socket address of socket.
11050	char	*ai_canonname	Canonical name of service location.
11051	struct addrinfo	*ai_next	Pointer to next in list.

11052 The **addrinfo** structure shall not include any additional members which have a floating-point  
 11053 type if an object of that type with an all-bits-zero representation does not have the value 0.0.

11054 MX Implementations that define `__STDC_IEC_559__` are required to treat the all-zero bit pattern for  
 11055 a floating point object as a representation of 0.0, and may therefore have floating-point type  
 11056 members.

11057 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-  
 11058 distinct integer constants for use in the *ai\_flags* field of the **addrinfo** structure:

11059	AI_PASSIVE	Socket address is intended for <i>bind()</i> .
11060	AI_CANONNAME	Request for canonical name.
11061	AI_NUMERICHOST	Return numeric host address as name.
11062	AI_NUMERICSERV	Inhibit service name resolution.
11063	AI_V4MAPPED	If no IPv6 addresses are found, query for IPv4 addresses and return them 11064 to the caller as IPv4-mapped IPv6 addresses.
11065	AI_ALL	Query for both IPv4 and IPv6 addresses.
11066	AI_ADDRCONFIG	Query for IPv4 addresses only when an IPv4 address is configured; query 11067 for IPv6 addresses only when an IPv6 address is configured.

11068 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-  
 11069 distinct integer constants for use in the *flags* argument to *getnameinfo()*:

11070	NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
11071	NI_NUMERICHOST	The numeric form of the node's address is returned instead of its name.
11072	NI_NAMEREQD	Return an error if the node's name cannot be located in the database.
11073	NI_NUMERICSERV	The numeric form of the service address is returned instead of its name.
11074	NI_NUMERICSCOPE	
11075		For IPv6 addresses, the numeric form of the scope identifier is returned 11076 instead of its name.
11077	NI_DGRAM	Indicates that the service is a datagram service (SOCK_DGRAM).

11078 **Address Information Errors**

11079 The **<netdb.h>** header shall define the following symbolic constants for use as error values for  
 11080 *getaddrinfo()* and *getnameinfo()*. The values shall be suitable for use in **#if** preprocessing  
 11081 directives.

11082 **EAI\_AGAIN** The name could not be resolved at this time. Future attempts may  
 11083 succeed.

11084 **EAI\_BADFLAGS** The flags had an invalid value.

11085 **EAI\_FAIL** A non-recoverable error occurred.

11086 **EAI\_FAMILY** The address family was not recognized or the address length was invalid  
 11087 for the specified family.

11088 **EAI\_MEMORY** There was a memory allocation failure.

11089 **EAI\_NONAME** The name does not resolve for the supplied parameters.

11090 **NI\_NAMEREQD** is set and the host's name cannot be located, or both  
 11091 *nodename* and *servertime* were null.

11092 **EAI\_SERVICE** The service passed was not recognized for the specified socket type.

11093 **EAI\_SOCKTYPE** The intended socket type was not recognized.

11094 **EAI\_SYSTEM** A system error occurred. The error code can be found in *errno*.

11095 **EAI\_OVERFLOW** An argument buffer overflowed.

11096 The following shall be declared as functions and may also be defined as macros. Function  
 11097 prototypes shall be provided.

```

11098 void          endhostent(void);
11099 void          endnetent(void);
11100 void          endprotoent(void);
11101 void          endservent(void);
11102 void          freeaddrinfo(struct addrinfo *);
11103 const char    *gai_strerror(int);
11104 int           getaddrinfo(const char *restrict, const char *restrict,
11105                          const struct addrinfo *restrict,
11106                          struct addrinfo **restrict);
11107 struct hostent *gethostent(void);
11108 int           getnameinfo(const struct sockaddr *restrict, socklen_t,
11109                          char *restrict, socklen_t, char *restrict,
11110                          socklen_t, int);
11111 struct netent *getnetbyaddr(uint32_t, int);
11112 struct netent *getnetbyname(const char *);
11113 struct netent *getnetent(void);
11114 struct protoent *getprotobyname(const char *);
11115 struct protoent *getprotobynumber(int);
11116 struct protoent *getprotoent(void);
11117 struct servent *getservbyname(const char *, const char *);
11118 struct servent *getservbyport(int, const char *);
11119 struct servent *getservent(void);
11120 void          sethostent(int);
11121 void          setnetent(int);
11122 void          setprotoent(int);

```

11123           void                           setserverent(int);

11124           The <netdb.h> header shall define the **socklen\_t** type through **typedef**, as described in  
11125           <sys/socket.h>.

11126           Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>,  
11127           <sys/socket.h>, and <inttypes.h>.

#### 11128 APPLICATION USAGE

11129           The requirement that **addrinfo** does not include any additional members which have a floating-  
11130           point type if an object of that type with an all-bits-zero representation does not have the value  
11131           0.0 is to allow initialization of an **addrinfo** *hints* structure (see XSH *freeaddrinfo()*) using:

```
11132           struct addrinfo hints;  
11133           memset(&hints, 0, sizeof hints);
```

11134           as an alternative to the use of default initialization.

#### 11135 RATIONALE

11136           None.

#### 11137 FUTURE DIRECTIONS

11138           None.

#### 11139 SEE ALSO

11140           <inttypes.h>, <netinet/in.h>, <sys/socket.h>

11141           XSH *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *freeaddrinfo()*, *gai\_strerror()*,  
11142           *getnameinfo()*

#### 11143 CHANGE HISTORY

11144           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

11145           The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for  
11146           *gai\_strerror()* from **char \*** to **const char \***. This is for coordination with the IPnG Working Group.

11147           IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the  
11148           NI\_NUMERICSCOPE macro and correcting the *getnameinfo()* function prototype. These changes  
11149           are for alignment with IPv6.

#### 11150 Issue 7

11151           SD5-XBD-ERN-14 is applied, changing the description of the *s\_port* member of the **servent**  
11152           structure.

11153           The obsolescent *h\_errno* external integer, and the obsolescent *gethostbyaddr()* and *gethostbyname()*  
11154           functions are removed, along with the HOST\_NOT\_FOUND, NO\_DATA, NO\_RECOVERY, and  
11155           TRY\_AGAIN macros.

11156           This reference page is clarified with respect to macros and symbolic constants.

#### 11157 Issue 8

11158           Austin Group Defect 940 is applied, requiring that the **addrinfo** structure does not include any  
11159           additional members which have a floating-point type if an object of that type with an all-bits-  
11160           zero representation does not have the value 0.0.

11161           Austin Group Defect 1289 is applied, removing some redundant text from the DESCRIPTION.

11162           Austin Group Defect 1327 is applied, changing ``flags'' to ``ai\_flags''.

11163 **NAME**

11164           netinet/in.h — Internet address family

11165 **SYNOPSIS**

11166       #include <netinet/in.h>

11167 **DESCRIPTION**

11168       The <netinet/in.h> header shall define the following types:

11169       **in\_port\_t**   Equivalent to the type **uint16\_t** as described in <inttypes.h>.

11170       **in\_addr\_t**   Equivalent to the type **uint32\_t** as described in <inttypes.h>.

11171       The <netinet/in.h> header shall define the **sa\_family\_t** type as described in <sys/socket.h>.

11172       The <netinet/in.h> header shall define the **uint8\_t** and **uint32\_t** types as described in <inttypes.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.

11175       The <netinet/in.h> header shall define the **in\_addr** structure, which shall include at least the following member:

11177       in\_addr\_t   s\_addr

11178       The <netinet/in.h> header shall define the **sockaddr\_in** structure, which shall include at least the following members:

11180	sa_family_t	sin_family	AF_INET.
11181	in_port_t	sin_port	Port number.
11182	struct in_addr	sin_addr	IP address.

11183       The *sin\_port* and *sin\_addr* members shall be in network byte order. If the *sin\_port* value passed to *bind()* is zero, the port number bound to the socket shall be one chosen by the implementation from an implementation-defined port range to produce an unused local address.

11186       The **sockaddr\_in** structure is used to store addresses for the Internet address family. Pointers to this type shall be cast by applications to **struct sockaddr \*** for use with socket functions.

11188 IP6       The <netinet/in.h> header shall define the **in6\_addr** structure, which shall include at least the following member:

11190       uint8\_t s6\_addr[16]

11191       This array is used to contain a 128-bit IPv6 address, stored in network byte order.

11192       The <netinet/in.h> header shall define the **sockaddr\_in6** structure, which shall include at least the following members:

11194	sa_family_t	sin6_family	AF_INET6.
11195	in_port_t	sin6_port	Port number.
11196	uint32_t	sin6_flowinfo	IPv6 traffic class and flow information.
11197	struct in6_addr	sin6_addr	IPv6 address.
11198	uint32_t	sin6_scope_id	Set of interfaces for a scope.

11199       The **sockaddr\_in6** structure shall not include any additional members which have a floating-point type if an object of that type with an all-bits-zero representation does not have the value 0.0.

11202 IP6 MX       Implementations that define `__STDC_IEC_559__` are required to treat the all-zero bit pattern for a floating point object as a representation of 0.0, and may therefore have floating-point type members.

11205 IP6 The *sin6\_port* and *sin6\_addr* members shall be in network byte order. If the *sin6\_port* value  
 11206 passed to *bind()* is zero, the port number bound to the socket shall be one chosen by the  
 11207 implementation from an implementation-defined port range to produce an unused local  
 11208 address.

11209 Prior to calling a function in this standard which reads values from a **sockaddr\_in6** structure (for  
 11210 example, *bind()* or *connect()*), the application shall ensure that all members of the structure,  
 11211 including any additional non-standard members, if any, are initialized. If the **sockaddr\_in6**  
 11212 structure has a non-standard member, and that member has a value other than the value that  
 11213 would result from default initialization, the behavior of any function in this standard that reads  
 11214 values from the **sockaddr\_in6** structure is implementation-defined. All functions in this  
 11215 standard that return data in a **sockaddr\_in6** structure (for example, *getaddrinfo()* or *accept()*)  
 11216 shall initialize the structure in a way that meets the above requirements, and shall ensure that  
 11217 each non-standard member, if any, has a value that produces the same behavior as default  
 11218 initialization would in all functions in this standard which read values from a **sockaddr\_in6**  
 11219 structure.

11220 The *sin6\_scope\_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the  
 11221 scope of the address carried in the *sin6\_addr* field. For a link scope *sin6\_addr*, the application  
 11222 shall ensure that *sin6\_scope\_id* is a link index. For a site scope *sin6\_addr*, the application shall  
 11223 ensure that *sin6\_scope\_id* is a site index. The mapping of *sin6\_scope\_id* to an interface or set of  
 11224 interfaces is implementation-defined.

11225 The <netinet/in.h> header shall declare the following external variable:

```
11226 const struct in6_addr in6addr_any
```

11227 This variable is initialized by the system to contain the wildcard IPv6 address. The  
 11228 <netinet/in.h> header also defines the IN6ADDR\_ANY\_INIT macro. This macro shall be  
 11229 constant at compile time and can be used to initialize a variable of type **struct in6\_addr** to the  
 11230 IPv6 wildcard address.

11231 The <netinet/in.h> header shall declare the following external variable:

```
11232 const struct in6_addr in6addr_loopback
```

11233 This variable is initialized by the system to contain the loopback IPv6 address. The  
 11234 <netinet/in.h> header also defines the IN6ADDR\_LOOPBACK\_INIT macro. This macro shall be  
 11235 constant at compile time and can be used to initialize a variable of type **struct in6\_addr** to the  
 11236 IPv6 loopback address.

11237 The <netinet/in.h> header shall define the **ipv6\_mreq** structure, which shall include at least the  
 11238 following members:

```
11239 struct in6_addr  ipv6mr_multiaddr  IPv6 multicast address.  

    11240 unsigned        ipv6mr_interface  Interface index.
```

11241 The <netinet/in.h> header shall define the following symbolic constants for use as values of the  
 11242 *level* argument of *getsockopt()* and *setsockopt()*:

11243	IPPROTO_IP	Internet protocol.
11244 IP6	IPPROTO_IPV6	Internet Protocol Version 6.
11245	IPPROTO_ICMP	Control message protocol.

11246	RS	IPPROTO_RAW	Raw IP Packets Protocol.
11247		IPPROTO_TCP	Transmission control protocol.
11248		IPPROTO_UDP	User datagram protocol.
11249		The <netinet/in.h> header shall define the following symbolic constant for use as a local address in the structure passed to <i>bind()</i> :	
11250			
11251		INADDR_ANY	IPv4 wildcard address.
11252		The <netinet/in.h> header shall define the following symbolic constant for use as a destination address in the structures passed to <i>connect()</i> , <i>sendmsg()</i> , and <i>sendto()</i> :	
11253			
11254		INADDR_BROADCAST	IPv4 broadcast address.
11255		The <netinet/in.h> header shall define the following symbolic constant, with the value specified, to help applications declare buffers of the proper size to store IPv4 addresses in string form:	
11256			
11257			
11258		INET_ADDRSTRLEN	16. Length of the string form for IP.
11259		The <i>htonl()</i> , <i>htons()</i> , <i>ntohl()</i> , and <i>ntohs()</i> functions shall be available as described in <arpa/inet.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.	
11260			
11261			
11262	IP6	The <netinet/in.h> header shall define the following symbolic constant, with the value specified, to help applications declare buffers of the proper size to store IPv6 addresses in string form:	
11263			
11264			
11265		INET6_ADDRSTRLEN	46. Length of the string form for IPv6.
11266	IP6	The <netinet/in.h> header shall define the following symbolic constants, with distinct integer values, for use in the <i>option_name</i> argument in the <i>getsockopt()</i> or <i>setsockopt()</i> functions at protocol level IPPROTO_IPV6:	
11267			
11268			
11269		IPV6_JOIN_GROUP	Join a multicast group.
11270		IPV6_LEAVE_GROUP	Quit a multicast group.
11271		IPV6_MULTICAST_HOPS	
11272			Multicast hop limit.
11273		IPV6_MULTICAST_IF	Interface to use for outgoing multicast packets.
11274		IPV6_MULTICAST_LOOP	
11275			Multicast packets are delivered back to the local application.
11276		IPV6_UNICAST_HOPS	Unicast hop limit.
11277		IPV6_V6ONLY	Restrict AF_INET6 socket to IPv6 communications only.
11278		The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses. Each macro is of type <b>int</b> and takes a single argument of type <b>const struct in6_addr *</b> :	
11279			
11280		IN6_IS_ADDR_UNSPECIFIED	
11281			Unspecified address.
11282		IN6_IS_ADDR_LOOPBACK	
11283			Loopback address.



11284	<b>IN6_IS_ADDR_MULTICAST</b>
11285	Multicast address.
11286	<b>IN6_IS_ADDR_LINKLOCAL</b>
11287	Unicast link-local address.
11288	<b>IN6_IS_ADDR_SITELOCAL</b>
11289	Unicast site-local address.
11290	<b>IN6_IS_ADDR_V4MAPPED</b>
11291	IPv4 mapped address.
11292	<b>IN6_IS_ADDR_V4COMPAT</b>
11293	IPv4-compatible address.
11294	<b>IN6_IS_ADDR_MC_NODELOCAL</b>
11295	Multicast node-local address.
11296	<b>IN6_IS_ADDR_MC_LINKLOCAL</b>
11297	Multicast link-local address.
11298	<b>IN6_IS_ADDR_MC_SITELOCAL</b>
11299	Multicast site-local address.
11300	<b>IN6_IS_ADDR_MC_ORGLOCAL</b>
11301	Multicast organization-local address.
11302	<b>IN6_IS_ADDR_MC_GLOBAL</b>
11303	Multicast global address.

#### 11304 **APPLICATION USAGE**

11305 Although applications are required to initialize all members (including any non-standard ones)  
 11306 of a **sockaddr\_in6** structure, the same is not required for the **sockaddr\_in** structure, since  
 11307 historically many applications only initialized the standard members. Despite this, applications  
 11308 are encouraged to initialize **sockaddr\_in** structures in a manner similar to the required  
 11309 initialization of **sockaddr\_in6** structures.

11310 The requirement that **sockaddr\_in6** does not include any additional members which have a  
 11311 floating-point type if an object of that type with an all-bits-zero representation does not have the  
 11312 value 0.0 is to allow initialization of a **sockaddr\_in6** structure using:

```
11313 struct sockaddr_in6 sa;  
11314 memset(&sa, 0, sizeof sa);
```

11315 as an alternative to the use of default initialization.

#### 11316 **RATIONALE**

11317 The INADDR\_ANY and INADDR\_BROADCAST values are byte-order-neutral and thus their  
 11318 byte order is not specified. Many implementations have additional constants as extensions, such  
 11319 as INADDR\_LOOPBACK, that are not byte-order-neutral. Traditionally, these constants are in  
 11320 host byte order, requiring the use of *htonl()* when using them in a **sockaddr\_in** structure.

#### 11321 **FUTURE DIRECTIONS**

11322 None.

11323 **SEE ALSO**

11324 Section 4.13 (on page 99), <arpa/inet.h>, <inttypes.h>, <sys/socket.h>  
11325 XSH *bind()*, *connect()*, *getsockopt()*, *htonl()*, *sendmsg()*, *sendto()*, *setsockopt()*

11326 **CHANGE HISTORY**

11327 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.  
11328 The *sin\_zero* member was removed from the **sockaddr\_in** structure as per The Open Group Base  
11329 Resolution bwg2001-004.  
11330 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to  
11331 the *in6addr\_any* and *in6addr\_loopback* external variables.  
11332 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/22 is applied, making it clear which  
11333 structure members are in network byte order.

11334 **Issue 7**

11335 This reference page is clarified with respect to macros and symbolic constants.  
11336 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0061 [355] is applied.  
11337 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0065 [934], XBD/TC2-2008/0066 [952],  
11338 XBD/TC2-2008/0067 [934], and XBD/TC2-2008/0068 [952] are applied.

11339 **Issue 8**

11340 Austin Group Defect 940 is applied, requiring that the **sockaddr\_in6** structure does not include  
11341 any additional members which have a floating-point type if an object of that type with an all-  
11342 bits-zero representation does not have the value 0.0.  
11343 Austin Group Defect 1068 is applied, specifying how a *sin\_port* or *sin6\_port* value of zero is  
11344 handled by *bind()*.  
11345 Austin Group Defect 1299 is applied, changing <netinet\_in.h> to <netinet/in.h>.

11346 **NAME**

11347           netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

11348 **SYNOPSIS**

11349           #include <netinet/tcp.h>

11350 **DESCRIPTION**

11351           The <netinet/tcp.h> header shall define the following symbolic constant for use as a socket option at the IPPROTO\_TCP level:

11353           TCP\_NODELAY   Avoid coalescing of small segments.

11354           The implementation need not allow the value of the option to be set via *setsockopt()* or retrieved via *getsockopt()*.

11356 **APPLICATION USAGE**

11357           None.

11358 **RATIONALE**

11359           None.

11360 **FUTURE DIRECTIONS**

11361           None.

11362 **SEE ALSO**

11363           <sys/socket.h>

11364           XSH *getsockopt()*, *setsockopt()*

11365 **CHANGE HISTORY**

11366           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

11367 **Issue 7**

11368           This reference page is clarified with respect to macros and symbolic constants.

11369 **NAME**11370 `nl_types.h` — data types11371 **SYNOPSIS**11372 `#include <nl_types.h>`11373 **DESCRIPTION**11374 The **<nl\_types.h>** header shall define at least the following types:11375 **nl\_catd** Used by the message catalog functions `catopen()`, `catgets()`, and `catclose()`  
11376 to identify a catalog descriptor.11377 **nl\_item** Used by `nl_langinfo()` to identify items of `langinfo` data. Values of objects  
11378 of type **nl\_item** are defined in **<langinfo.h>**.11379 The **<nl\_types.h>** header shall define at least the following symbolic constants:11380 **NL\_SETD** Used by `gencat` when no `$set` directive is specified in a message text source  
11381 file. This constant can be passed as the value of `set_id` on subsequent calls  
11382 to `catgets()` (that is, to retrieve messages from the default message set).  
11383 The value of **NL\_SETD** is implementation-defined.11384 **NL\_CAT\_LOCALE** Value that can be passed as the `oflag` argument to `catopen()` to request that  
11385 message catalog selection depends on the `LC_MESSAGES` locale category,  
11386 rather than directly on the `LANG` environment variable.11387 The following shall be declared as functions and may also be defined as macros. Function  
11388 prototypes shall be provided.11389 `int catclose(nl_catd);`  
11390 `char *catgets(nl_catd, int, int, const char *);`  
11391 `nl_catd catopen(const char *, int);`11392 **APPLICATION USAGE**

11393 None.

11394 **RATIONALE**

11395 None.

11396 **FUTURE DIRECTIONS**

11397 None.

11398 **SEE ALSO**11399 [<langinfo.h>](#)11400 XSH `catclose()`, `catgets()`, `catopen()`, `nl_langinfo()`11401 XCU `gencat`11402 **CHANGE HISTORY**

11403 First released in Issue 2.

11404 **Issue 7**11405 The **<nl\_types.h>** header is moved from the XSI option to the Base.

11406 This reference page is clarified with respect to macros and symbolic constants.

11407 **Issue 8**11408 The description of **NL\_CAT\_LOCALE** is updated to eliminate the use of “`must”.

11409 **NAME**

11410 poll.h — definitions for the poll() function

11411 **SYNOPSIS**

11412 #include <poll.h>

11413 **DESCRIPTION**

11414 The <poll.h> header shall define the **pollfd** structure, which shall include at least the following  
11415 members:

11416 int fd The following descriptor being polled.  
11417 short events The input event flags (see below).  
11418 short revents The output event flags (see below).

11419 The <poll.h> header shall define the following type through **typedef**:

11420 **nfds\_t** An unsigned integer type used for the number of file descriptors.

11421 The implementation shall support one or more programming environments in which the width  
11422 of **nfds\_t** is no greater than the width of type **long**. The names of these programming  
11423 environments can be obtained using the *confstr()* function or the *getconf* utility.

11424 The <poll.h> header shall define the **sigset\_t** type as described in <signal.h>.

11425 The <poll.h> header shall define the **timespec** structure as described in <time.h>.

11426 The <poll.h> header shall define the following symbolic constants, zero or more of which may  
11427 be OR'ed together to form the *events* or *revents* members in the **pollfd** structure:

11428 POLLIN Data other than high-priority data may be read without blocking.  
11429 POLLRDNORM Normal data may be read without blocking.  
11430 POLLRDBAND Priority data may be read without blocking.  
11431 POLLPRI High priority data may be read without blocking.  
11432 POLLOUT Normal data may be written without blocking.  
11433 POLLWRNORM Equivalent to POLLOUT.  
11434 POLLWRBAND Priority data may be written.  
11435 POLLERR An error has occurred (*revents* only).  
11436 POLLHUP Device has been disconnected (*revents* only).  
11437 POLLNVAL Invalid *fd* member (*revents* only).

11438 The significance and semantics of normal, priority, and high-priority data are file and device-  
11439 specific.

11440 The following shall be declared as functions and may also be defined as macros. Function  
11441 prototypes shall be provided.

```
11442 int poll(struct pollfd [], nfds_t, int);
11443 int ppoll(struct pollfd [], nfds_t, const struct timespec *restrict,
11444           const sigset_t *restrict);
```

11445 Inclusion of the <poll.h> header may make visible all symbols from the headers <signal.h> and  
11446 <time.h>.

11447 **APPLICATION USAGE**

11448 None.

11449 **RATIONALE**

11450 None.

11451 **FUTURE DIRECTIONS**

11452 None.

11453 **SEE ALSO**11454 XSH *confstr()*, *poll()*11455 XCU *getconf*11456 **CHANGE HISTORY**

11457 First released in Issue 4, Version 2.

11458 **Issue 6**11459 The description of the symbolic constants is updated to match the *poll()* function.11460 Text related to STREAMS has been moved to the *poll()* reference page.11461 A note is added to the DESCRIPTION regarding the significance and semantics of normal,  
11462 priority, and high-priority data.11463 **Issue 7**11464 The **<poll.h>** header is moved from the XSI option to the Base.11465 **Issue 8**11466 Austin Group Defect 1263 is applied, adding *ppoll()*, requiring **<poll.h>** to define **sigset\_t** and  
11467 **struct timespec**, and allowing **<poll.h>** to make visible all symbols from **<signal.h>** and  
11468 **<time.h>**.

11469 **NAME**

11470 pthread.h — threads

11471 **SYNOPSIS**

11472 #include <pthread.h>

11473 **DESCRIPTION**

11474 The <pthread.h> header shall define the following symbolic constants:

- 11475 PTHREAD\_BARRIER\_SERIAL\_THREAD
- 11476 PTHREAD\_CANCEL\_ASYNCHRONOUS
- 11477 PTHREAD\_CANCEL\_ENABLE
- 11478 PTHREAD\_CANCEL\_DEFERRED
- 11479 PTHREAD\_CANCEL\_DISABLE
- 11480 PTHREAD\_CANCELED
- 11481 PTHREAD\_CREATE\_DETACHED
- 11482 PTHREAD\_CREATE\_JOINABLE
- 11483 TPS PTHREAD\_EXPLICIT\_SCHED
- 11484 PTHREAD\_INHERIT\_SCHED
- 11485 PTHREAD\_MUTEX\_DEFAULT
- 11486 PTHREAD\_MUTEX\_ERRORCHECK
- 11487 PTHREAD\_MUTEX\_NORMAL
- 11488 PTHREAD\_MUTEX\_RECURSIVE
- 11489 PTHREAD\_MUTEX\_ROBUST
- 11490 PTHREAD\_MUTEX\_STALLED
- 11491 PTHREAD\_ONCE\_INIT
- 11492 RPI|TPI PTHREAD\_PRIO\_INHERIT
- 11493 MC1 PTHREAD\_PRIO\_NONE
- 11494 RPP|TPP PTHREAD\_PRIO\_PROTECT
- 11495 PTHREAD\_PROCESS\_SHARED
- 11496 PTHREAD\_PROCESS\_PRIVATE
- 11497 TPS PTHREAD\_SCOPE\_PROCESS
- 11498 PTHREAD\_SCOPE\_SYSTEM

11499 The <pthread.h> header shall define the following compile-time constant expressions valid as  
 11500 initializers for the following types:

	Name	Initializer for Type
11501	PTHREAD_COND_INITIALIZER	pthread_cond_t
11502	PTHREAD_MUTEX_INITIALIZER	pthread_mutex_t
11503	PTHREAD_RWLOCK_INITIALIZER	pthread_rwlock_t
11504		

11505 The <pthread.h> header shall define the following compile-time constant expression, valid as an  
 11506 initializer for **pthread\_t**, representing a value that shall not compare equal to the thread ID of  
 11507 any existing thread:

11508 PTHREAD\_NULL

11509 The <pthread.h> header shall define the **pthread\_attr\_t**, **pthread\_barrier\_t**,  
 11510 **pthread\_barrierattr\_t**, **pthread\_cond\_t**, **pthread\_condattr\_t**, **pthread\_key\_t**, **pthread\_mutex\_t**,  
 11511 **pthread\_mutexattr\_t**, **pthread\_once\_t**, **pthread\_rwlock\_t**, **pthread\_rwlockattr\_t**,  
 11512 **pthread\_spinlock\_t**, and **pthread\_t** types as described in <sys/types.h>.

11513 The following shall be declared as functions and may also be defined as macros. Function  
 11514 prototypes shall be provided.

```

11515 OB      int  pthread_atfork(void (*) (void), void (*) (void),
11516          void (*) (void));
11517      int  pthread_attr_destroy(pthread_attr_t *);
11518      int  pthread_attr_getdetachstate(const pthread_attr_t *, int *);
11519      int  pthread_attr_getguardsize(const pthread_attr_t *restrict,
11520          size_t *restrict);
11521 TPS      int  pthread_attr_getinheritsched(const pthread_attr_t *restrict,
11522          int *restrict);
11523      int  pthread_attr_getschedparam(const pthread_attr_t *restrict,
11524          struct sched_param *restrict);
11525 TPS      int  pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
11526          int *restrict);
11527      int  pthread_attr_getscope(const pthread_attr_t *restrict,
11528          int *restrict);
11529 TSA TSS   int  pthread_attr_getstack(const pthread_attr_t *restrict,
11530          void **restrict, size_t *restrict);
11531 TSS      int  pthread_attr_getstacksize(const pthread_attr_t *restrict,
11532          size_t *restrict);
11533      int  pthread_attr_init(pthread_attr_t *);
11534      int  pthread_attr_setdetachstate(pthread_attr_t *, int);
11535      int  pthread_attr_setguardsize(pthread_attr_t *, size_t);
11536 TPS      int  pthread_attr_setinheritsched(pthread_attr_t *, int);
11537      int  pthread_attr_setschedparam(pthread_attr_t *restrict,
11538          const struct sched_param *restrict);
11539 TPS      int  pthread_attr_setschedpolicy(pthread_attr_t *, int);
11540      int  pthread_attr_setscope(pthread_attr_t *, int);
11541 TSA TSS   int  pthread_attr_setstack(pthread_attr_t *, void *, size_t);
11542 TSS      int  pthread_attr_setstacksize(pthread_attr_t *, size_t);
11543      int  pthread_barrier_destroy(pthread_barrier_t *);
11544      int  pthread_barrier_init(pthread_barrier_t *restrict,
11545          const pthread_barrierattr_t *restrict, unsigned);
11546      int  pthread_barrier_wait(pthread_barrier_t *);
11547      int  pthread_barrierattr_destroy(pthread_barrierattr_t *);
11548 TSH      int  pthread_barrierattr_getpshared(
11549          const pthread_barrierattr_t *restrict, int *restrict);
11550      int  pthread_barrierattr_init(pthread_barrierattr_t *);
11551 TSH      int  pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
11552      int  pthread_cancel(pthread_t);
11553      int  pthread_cond_broadcast(pthread_cond_t *);
11554      int  pthread_cond_clockwait(pthread_cond_t *restrict,
11555          pthread_mutex_t *restrict, clockid_t,
11556          const struct timespec *restrict);
11557      int  pthread_cond_destroy(pthread_cond_t *);
11558      int  pthread_cond_init(pthread_cond_t *restrict,
11559          const pthread_condattr_t *restrict);
11560      int  pthread_cond_signal(pthread_cond_t *);
11561      int  pthread_cond_timedwait(pthread_cond_t *restrict,
11562          pthread_mutex_t *restrict, const struct timespec *restrict);
11563      int  pthread_cond_wait(pthread_cond_t *restrict,
11564          pthread_mutex_t *restrict);
11565      int  pthread_condattr_destroy(pthread_condattr_t *);
11566      int  pthread_condattr_getclock(const pthread_condattr_t *restrict,

```



```

11567         clockid_t *restrict);
11568 TSH int pthread_condattr_getpshared(const pthread_condattr_t *restrict,
11569         int *restrict);
11570 int pthread_condattr_init(pthread_condattr_t *);
11571 int pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
11572 TSH int pthread_condattr_setpshared(pthread_condattr_t *, int);
11573 int pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
11574         void *(*)(void*), void *restrict);
11575 int pthread_detach(pthread_t);
11576 int pthread_equal(pthread_t, pthread_t);
11577 _Noreturn void
11578         pthread_exit(void *);
11579 TCT int pthread_getcpuclockid(pthread_t, clockid_t *);
11580 TPS int pthread_getschedparam(pthread_t, int *restrict,
11581         struct sched_param *restrict);
11582 void *pthread_getspecific(pthread_key_t);
11583 int pthread_join(pthread_t, void **);
11584 int pthread_key_create(pthread_key_t *, void (*)(void*));
11585 int pthread_key_delete(pthread_key_t);
11586 int pthread_mutex_clocklock(pthread_mutex_t *restrict, clockid_t,
11587         const struct timespec *restrict);
11588 int pthread_mutex_consistent(pthread_mutex_t *);
11589 int pthread_mutex_destroy(pthread_mutex_t *);
11590 RPP|TPP int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
11591         int *restrict);
11592 int pthread_mutex_init(pthread_mutex_t *restrict,
11593         const pthread_mutexattr_t *restrict);
11594 int pthread_mutex_lock(pthread_mutex_t *);
11595 RPP|TPP int pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
11596         int *restrict);
11597 int pthread_mutex_timedlock(pthread_mutex_t *restrict,
11598         const struct timespec *restrict);
11599 int pthread_mutex_trylock(pthread_mutex_t *);
11600 int pthread_mutex_unlock(pthread_mutex_t *);
11601 int pthread_mutexattr_destroy(pthread_mutexattr_t *);
11602 RPP|TPP int pthread_mutexattr_getprioceiling(
11603         const pthread_mutexattr_t *restrict, int *restrict);
11604 MC1 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
11605         int *restrict);
11606 TSH int pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
11607         int *restrict);
11608 int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict,
11609         int *restrict);
11610 int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
11611         int *restrict);
11612 int pthread_mutexattr_init(pthread_mutexattr_t *);
11613 RPP|TPP int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
11614 MC1 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
11615 TSH int pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
11616 int pthread_mutexattr_setrobust(pthread_mutexattr_t *, int);
11617 int pthread_mutexattr_settype(pthread_mutexattr_t *, int);
11618 int pthread_once(pthread_once_t *, void (*)(void));

```

```

11619     int   pthread_rwlock_destroy(pthread_rwlock_t *);
11620     int   pthread_rwlock_init(pthread_rwlock_t *restrict,
11621                               const pthread_rwlockattr_t *restrict);
11622     int   pthread_rwlock_clockrdlock(pthread_rwlock_t *restrict,
11623                                     clockid_t, const struct timespec *restrict);
11624     int   pthread_rwlock_clockwrlock(pthread_rwlock_t *restrict,
11625                                     clockid_t, const struct timespec *restrict);
11626     int   pthread_rwlock_rdlock(pthread_rwlock_t *);
11627     int   pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
11628                                     const struct timespec *restrict);
11629     int   pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
11630                                     const struct timespec *restrict);
11631     int   pthread_rwlock_tryrdlock(pthread_rwlock_t *);
11632     int   pthread_rwlock_trywrlock(pthread_rwlock_t *);
11633     int   pthread_rwlock_unlock(pthread_rwlock_t *);
11634     int   pthread_rwlock_wrlock(pthread_rwlock_t *);
11635     int   pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
11636 TSH    int   pthread_rwlockattr_getpshared(
11637         const pthread_rwlockattr_t *restrict, int *restrict);
11638     int   pthread_rwlockattr_init(pthread_rwlockattr_t *);
11639 TSH    int   pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
11640     pthread_t
11641         pthread_self(void);
11642     int   pthread_setcancelstate(int, int *);
11643     int   pthread_setcanceltype(int, int *);
11644 TPS    int   pthread_setschedparam(pthread_t, int,
11645                                   const struct sched_param *);
11646     int   pthread_setschedprio(pthread_t, int);
11647     int   pthread_setspecific(pthread_key_t, const void *);
11648     int   pthread_spin_destroy(pthread_spinlock_t *);
11649     int   pthread_spin_init(pthread_spinlock_t *, int);
11650     int   pthread_spin_lock(pthread_spinlock_t *);
11651     int   pthread_spin_trylock(pthread_spinlock_t *);
11652     int   pthread_spin_unlock(pthread_spinlock_t *);
11653     void  pthread_testcancel(void);

```

11654 The following may be declared as functions, or defined as macros, or both. If functions are  
11655 declared, function prototypes shall be provided.

```

11656         pthread_cleanup_pop()
11657         pthread_cleanup_push()

```

11658 Inclusion of the **<pthread.h>** header shall make symbols defined in the headers **<sched.h>** and  
11659 **<time.h>** visible.

11660 **APPLICATION USAGE**

11661 None.

11662 **RATIONALE**

11663 Since **pthread\_t** is an opaque type, a definition of PTHREAD\_NULL was added to allow for a  
 11664 null value of that type. Some conforming definitions of PTHREAD\_NULL could be:

11665 For a pointer type:

11666 #define PTHREAD\_NULL ((pthread\_t) NULL)

11667 For an integer type:

11668 #define PTHREAD\_NULL ((pthread\_t) -42)

11669 For a struct type:

11670 #define PTHREAD\_NULL ((const pthread\_t){ .\_\_foo = -1 })

11671 **FUTURE DIRECTIONS**

11672 None.

11673 **SEE ALSO**

11674 &lt;sched.h&gt;, &lt;sys/types.h&gt;, &lt;time.h&gt;

11675 XSH *pthread\_atfork()*, *pthread\_attr\_destroy()*, *pthread\_attr\_getdetachstate()*,  
 11676 *pthread\_attr\_getguardsize()*, *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedparam()*,  
 11677 *pthread\_attr\_getschedpolicy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getstack()*,  
 11678 *pthread\_attr\_getstacksize()*, *pthread\_barrier\_destroy()*, *pthread\_barrier\_wait()*,  
 11679 *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_getpshared()*, *pthread\_cancel()*,  
 11680 *pthread\_cleanup\_pop()*, *pthread\_cond\_broadcast()*, *pthread\_cond\_clockwait()*, *pthread\_cond\_destroy()*,  
 11681 *pthread\_condattr\_destroy()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_getpshared()*,  
 11682 *pthread\_create()*, *pthread\_detach()*, *pthread\_equal()*, *pthread\_exit()*, *pthread\_getcpuclockid()*,  
 11683 *pthread\_getschedparam()*, *pthread\_getspecific()*, *pthread\_join()*, *pthread\_key\_create()*,  
 11684 *pthread\_key\_delete()*, *pthread\_mutex\_clocklock()*, *pthread\_mutex\_consistent()*,  
 11685 *pthread\_mutex\_destroy()*, *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_lock()*,  
 11686 *pthread\_mutexattr\_destroy()*, *pthread\_mutexattr\_getprioceiling()*, *pthread\_mutexattr\_getprotocol()*,  
 11687 *pthread\_mutexattr\_getpshared()*, *pthread\_mutexattr\_getrobust()*, *pthread\_mutexattr\_gettype()*,  
 11688 *pthread\_once()*, *pthread\_rwlock\_clockrdlock()*, *pthread\_rwlock\_clockwrlock()*,  
 11689 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_trywrlock()*,  
 11690 *pthread\_rwlock\_unlock()*, *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_getpshared()*,  
 11691 *pthread\_self()*, *pthread\_setcancelstate()*, *pthread\_setschedprio()*, *pthread\_spin\_destroy()*,  
 11692 *pthread\_spin\_lock()*, *pthread\_spin\_unlock()*

11693 **CHANGE HISTORY**

11694 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

11695 **Issue 6**

11696 The RTT margin markers are broken out into their POSIX options.

11697 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
 11698 *pthread\_cond\_wait()* function.

11699 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the  
 11700 *pthread\_setschedparam()* function so that its second argument is of type **int**.

11701 The *pthread\_getcpuclockid()* and *pthread\_mutex\_timedlock()* functions are added for alignment  
 11702 with IEEE Std 1003.1d-1999.

11703 The following functions are added for alignment with IEEE Std 1003.1j-2000:

11704 *pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*,  
 11705 *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_init()*,  
 11706 *pthread\_barrierattr\_setpshared()*, *pthread\_condattr\_getclock()*, *pthread\_condattr\_setclock()*,  
 11707 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_spin\_destroy()*,  
 11708 *pthread\_spin\_init()*, *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, and *pthread\_spin\_unlock()*.

11709 PTHREAD\_RWLOCK\_INITIALIZER is removed for alignment with IEEE Std 1003.1j-2000.

11710 Functions previously marked as part of the Read-Write Locks option are now moved to the  
 11711 Threads option.

11712 The **restrict** keyword is added to the prototypes for *pthread\_attr\_getguardsize()*,  
 11713 *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedparam()*, *pthread\_attr\_getschedpolicy()*,  
 11714 *pthread\_attr\_getscope()*, *pthread\_attr\_getstackaddr()*, *pthread\_attr\_getstacksize()*,  
 11715 *pthread\_attr\_setschedparam()*, *pthread\_barrier\_init()*, *pthread\_barrierattr\_getpshared()*,  
 11716 *pthread\_cond\_init()*, *pthread\_cond\_signal()*, *pthread\_cond\_timedwait()*, *pthread\_cond\_wait()*,  
 11717 *pthread\_condattr\_getclock()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*,  
 11718 *pthread\_getschedparam()*, *pthread\_mutex\_getprioceiling()*, *pthread\_mutex\_init()*,  
 11719 *pthread\_mutex\_setprioceiling()*, *pthread\_mutexattr\_getprioceiling()*, *pthread\_mutexattr\_getprotocol()*,  
 11720 *pthread\_mutexattr\_getpshared()*, *pthread\_mutexattr\_gettype()*, *pthread\_rwlock\_init()*,  
 11721 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlockattr\_getpshared()*, and  
 11722 *pthread\_sigmask()*.

11723 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and  
 11724 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI option.

11725 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for  
 11726 the *pthread\_kill()* and *pthread\_sigmask()* functions. These are required to be in the <signal.h>  
 11727 header. They are allowed here through the name space rules.

11728 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread\_setschedprio()* function.

11729 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors  
 11730 that were in contradiction with the System Interfaces volume of POSIX.1-2024.

11731 **Issue 7**

11732 SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread\_mutex\_timedlock()*  
 11733 function prototype.

11734 SD5-XBD-ERN-62 is applied.

11735 Austin Group Interpretation 1003.1-2001 #048 is applied, reinstating the  
 11736 PTHREAD\_RWLOCK\_INITIALIZER symbol.

11737 The <pthread.h> header is moved from the Threads option to the Base.

11738 The following extended mutex types are moved from the XSI option to the Base:

```
11739     PTHREAD_MUTEX_NORMAL
11740     PTHREAD_MUTEX_ERRORCHECK
11741     PTHREAD_MUTEX_RECURSIVE
11742     PTHREAD_MUTEX_DEFAULT
```

11743 The PTHREAD\_MUTEX\_ROBUST and PTHREAD\_MUTEX\_STALLED symbols and the  
 11744 *pthread\_mutex\_consistent()*, *pthread\_mutexattr\_getrobust()*, and *pthread\_mutexattr\_setrobust()*  
 11745 functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

11746 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options  
 11747 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex

- 11748 or Robust Mutex Priority Inheritance, respectively.
- 11749 This reference page is clarified with respect to macros and symbolic constants.
- 11750 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0069 [624] is applied.
- 11751 **Issue 8**
- 11752 Austin Group Defect 599 is applied, adding the PTHREAD\_NULL constant.
- 11753 Austin Group Defect 851 is applied, marking *pthread\_atfork()* as obsolescent.
- 11754 Austin Group Defect 1216 is applied, adding *pthread\_cond\_clockwait()*, *pthread\_mutex\_clocklock()*,  
11755 *pthread\_rwlock\_clockrdlock()*, and *pthread\_rwlock\_clockwrlock()*.
- 11756 Austin Group Defect 1302 is applied, adding *\_Noreturn* to *pthread\_exit()*.
- 11757 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

11758 **NAME**

11759           pwd.h — password structure

11760 **SYNOPSIS**

11761           #include &lt;pwd.h&gt;

11762 **DESCRIPTION**11763           The **<pwd.h>** header shall define the **struct passwd**, structure, which shall include at least the  
11764           following members:

11765	char	*pw_name	User's login name.
11766	uid_t	pw_uid	Numerical user ID.
11767	gid_t	pw_gid	Numerical group ID.
11768	char	*pw_dir	Initial working directory.
11769	char	*pw_shell	Program to use as shell.

11770           The **<pwd.h>** header shall define the **gid\_t**, **uid\_t**, and **size\_t** types as described in  
11771           **<sys/types.h>**.11772           The following shall be declared as functions and may also be defined as macros. Function  
11773           prototypes shall be provided.

```

11774 XSI void          endpwent(void);
11775     struct passwd *getpwent(void);
11776     struct passwd *getpwnam(const char *);
11777     int           getpwnam_r(const char *, struct passwd *, char *,
11778                             size_t, struct passwd **);
11779     struct passwd *getpwuid(uid_t);
11780     int           getpwuid_r(uid_t, struct passwd *, char *,
11781                             size_t, struct passwd **);
11782 XSI void          setpwent(void);

```

11783 **APPLICATION USAGE**

11784           None.

11785 **RATIONALE**

11786           None.

11787 **FUTURE DIRECTIONS**

11788           None.

11789 **SEE ALSO**11790           **<sys/types.h>**11791           XSH *endpwent()*, *getpwnam()*, *getpwuid()*11792 **CHANGE HISTORY**

11793           First released in Issue 1.

11794 **Issue 5**

11795           The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

11796 **Issue 6**11797           The following new requirements on POSIX implementations derive from alignment with the  
11798           Single UNIX Specification:

- 11799
- The **gid\_t** and **uid\_t** types are mandated.

11800                   • The *getpwnam\_r()* and *getpwuid\_r()* functions are marked as part of the Thread-Safe  
11801                    Functions option.

11802 **Issue 7**

11803                   SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size\_t** type.

11804 **NAME**

11805           regex.h — regular expression matching types

11806 **SYNOPSIS**

11807       #include &lt;regex.h&gt;

11808 **DESCRIPTION**11809       The **<regex.h>** header shall define the structures and symbolic constants used by the *regcomp()*,  
11810 *regexexec()*, *regerror()*, and *regfree()* functions.11811       The **<regex.h>** header shall define the **regex\_t** structure type, which shall include at least the  
11812 following member:

11813       size\_t     re\_nsub     Number of parenthesized subexpressions.

11814       The **<regex.h>** header shall define the **size\_t** type as described in **<sys/types.h>**.11815       The **<regex.h>** header shall define the **regoff\_t** type as a signed integer type that can hold the  
11816 largest value that can be stored in either a **ptrdiff\_t** type or a **ssize\_t** type.11817       The **<regex.h>** header shall define the **regmatch\_t** structure type, which shall include at least the  
11818 following members:11819       regoff\_t    rm\_so     Byte offset from start of string  
11820                             to start of substring.11821       regoff\_t    rm\_eo     Byte offset from start of string of the  
11822                             first character after the end of substring.11823       The **<regex.h>** header shall define the following symbolic constants for the *cflags* parameter to  
11824 the *regcomp()* function:

11825       REG\_EXTENDED     Use Extended Regular Expressions.

11826       REG\_ICASE         Perform pattern matching in a case-insensitive manner.

11827       REG\_MINIMAL     Change default matching behavior to leftmost shortest possible match.  
11828                         Only applicable to REG\_EXTENDED regular expressions.11829       REG\_NOSUB        Report only success or fail in *regexexec()*.

11830       REG\_NEWLINE     Change the handling of &lt;newline&gt;.

11831       The **<regex.h>** header shall define the following symbolic constants for the *eflags* parameter to  
11832 the *regexexec()* function:11833       REG\_NOTBOL       The <circumflex> character ('^'), when taken as a special character, does  
11834                         not match the beginning of *string*.11835       REG\_NOTEOL       The <dollar-sign> ('\$'), when taken as a special character, does not  
11836                         match the end of *string*.11837       The **<regex.h>** header shall define the following symbolic constants as error return values:11838       REG\_NOMATCH     *regexexec()* failed to match.

11839       REG\_BADPAT       Invalid regular expression.

11840       REG\_ECOLLATE     Invalid collating element referenced.

11841       REG\_ETYPE        Invalid character class type referenced.

11842       REG\_EESCAPE     Trailing &lt;backslash&gt; character in pattern.



11843	REG_ESUBREG	Number in <i>\digit</i> invalid or in error.
11844	REG_EBRACK	"[]" imbalance.
11845	REG_EPAREN	"\(\)" or "()" imbalance.
11846	REG_EBRACE	"\{\}" imbalance.
11847	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than
11848		two numbers, first larger than second.
11849	REG_ERANGE	Invalid endpoint in range expression.
11850	REG_ESPACE	Out of memory.
11851	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
11852		The following shall be declared as functions and may also be defined as macros. Function
11853		prototypes shall be provided.
11854		<code>int regcomp(regex_t *restrict, const char *restrict, int);</code>
11855		<code>size_t regerror(int, const regex_t *restrict, char *restrict, size_t);</code>
11856		<code>int regexec(const regex_t *restrict, const char *restrict, size_t,</code>
11857		<code>regmatch_t [restrict], int);</code>
11858		<code>void regfree(regex_t *);</code>
11859		The implementation may define additional macros or constants using names beginning with
11860		REG_.

#### 11861 APPLICATION USAGE

11862 None.

#### 11863 RATIONALE

11864 None.

#### 11865 FUTURE DIRECTIONS

11866 None.

#### 11867 SEE ALSO

11868 [<sys/types.h>](#)

11869 XSH *regcomp()*

#### 11870 CHANGE HISTORY

11871 First released in Issue 4.

11872 Originally derived from the ISO POSIX-2 standard.

#### 11873 Issue 6

11874 The REG\_ENOSYS constant is marked obsolescent.

11875 The **restrict** keyword is added to the prototypes for *regcomp()*, *regerror()*, and *regexec()*.

11876 A statement is added that the **size\_t** type is defined as described in [<sys/types.h>](#).

#### 11877 Issue 7

11878 SD5-XBD-ERN-60 is applied.

11879 The obsolescent REG\_ENOSYS constant is removed.

11880 This reference page is clarified with respect to macros and symbolic constants.

11881 **Issue 8**

11882 Austin Group Defect 793 is applied, adding REG\_MINIMAL.

11883 Austin Group Defect 1031 is applied, changing the description of REG\_ICASE.

11884 **NAME**  
 11885 sched.h — execution scheduling

11886 **SYNOPSIS**  
 11887 #include <sched.h>

11888 **DESCRIPTION**

11889 PS The <sched.h> header shall define the **pid\_t** type as described in <sys/types.h>.

11890 SS|TSP The <sched.h> header shall define the **time\_t** type as described in <sys/types.h>.

11891 The <sched.h> header shall define the **timespec** structure as described in <time.h>.

11892 The <sched.h> header shall define the **sched\_param** structure, which shall include the  
 11893 scheduling parameters required for implementation of each supported scheduling policy. This  
 11894 structure shall include at least the following member:

11895 int sched\_priority Process or thread execution scheduling priority.

11896 SS|TSP The **sched\_param** structure defined in <sched.h> shall include the following members in  
 11897 addition to those specified above:

11898	int	sched_ss_low_priority	Low scheduling priority for
11899			sporadic server.
11900	struct timespec	sched_ss_repl_period	Replenishment period for
11901			sporadic server.
11902	struct timespec	sched_ss_init_budget	Initial budget for sporadic server.
11903	int	sched_ss_max_repl	Maximum pending replenishments for
11904			sporadic server.

11905 Each process or thread is controlled by an associated scheduling policy and priority. Associated  
 11906 with each policy is a priority range. Each policy definition specifies the minimum priority range  
 11907 for that policy. The priority ranges for each policy may overlap the priority ranges of other  
 11908 policies.

11909 Four scheduling policies are defined; others may be defined by the implementation. The four  
 11910 standard policies are indicated by the values of the following symbolic constants:

11911 PS|TPS **SCHED\_FIFO** First in-first out (FIFO) scheduling policy.

11912 PS|TPS **SCHED\_RR** Round robin scheduling policy.

11913 SS|TSP **SCHED\_SPORADIC** Sporadic server scheduling policy.

11914 PS|TPS **SCHED\_OTHER** Another scheduling policy.

11915 The values of these constants are distinct.

11916 The following shall be declared as functions and may also be defined as macros. Function  
 11917 prototypes shall be provided.

```

11918 PS|TPS int sched_get_priority_max(int);
11919 int sched_get_priority_min(int);
11920 PS int sched_getparam(pid_t, struct sched_param *);
11921 int sched_getscheduler(pid_t);
11922 PS|TPS int sched_rr_get_interval(pid_t, struct timespec *);
11923 PS int sched_setparam(pid_t, const struct sched_param *);
11924 int sched_setscheduler(pid_t, int, const struct sched_param *);
11925 int sched_yield(void);
    
```

- 11926 Inclusion of the **<sched.h>** header may make visible all symbols from the **<time.h>** header.
- 11927 **APPLICATION USAGE**
- 11928 None.
- 11929 **RATIONALE**
- 11930 None.
- 11931 **FUTURE DIRECTIONS**
- 11932 None.
- 11933 **SEE ALSO**
- 11934 **<sys/types.h>**, **<time.h>**
- 11935 XSH *sched\_get\_priority\_max()*, *sched\_getparam()*, *sched\_getscheduler()*, *sched\_rr\_get\_interval()*,  
11936 *sched\_setparam()*, *sched\_setscheduler()*, *sched\_yield()*
- 11937 **CHANGE HISTORY**
- 11938 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 11939 **Issue 6**
- 11940 The **<sched.h>** header is marked as part of the Process Scheduling option.
- 11941 Sporadic server members are added to the **sched\_param** structure, and the SCHED\_SPORADIC  
11942 scheduling policy is added for alignment with IEEE Std 1003.1d-1999.
- 11943 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched\_param** structure whose  
11944 members *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* should be type **struct timespec** and not  
11945 **timespec**.
- 11946 Symbols from **<time.h>** may be made visible when **<sched.h>** is included.
- 11947 IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied,  
11948 aligning the function prototype shading and margin codes with the System Interfaces volume of  
11949 IEEE Std 1003.1-2001.
- 11950 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/23 is applied, updating the  
11951 DESCRIPTION to differentiate between thread and process execution.
- 11952 **Issue 7**
- 11953 SD5-XBD-ERN-13 is applied.
- 11954 Austin Group Interpretation 1003.1-2001 #064 is applied, correcting the options markings.
- 11955 The **<sched.h>** headers is moved from the Threads option to the Base.
- 11956 Declarations for the **pid\_t** and **time\_t** types and the **timespec** structure are added.

11957 **NAME**

11958 search.h — search tables

11959 **SYNOPSIS**

11960 XSI `#include <search.h>`

11961 **DESCRIPTION**

11962 The <search.h> header shall define the **ENTRY** type for structure **entry** which shall include the  
11963 following members:

11964 char \*key  
11965 void \*data

11966 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as  
11967 follows:

11968 enum { FIND, ENTER } ACTION;  
11969 enum { preorder, postorder, endorder, leaf } VISIT;

11970 The <search.h> header shall define the **size\_t** type as described in <sys/types.h>.

11971 The <search.h> header shall define via **typedef** the **posix\_tnode** type as an alias for **void**.

11972 The following shall be declared as functions and may also be defined as macros. Function  
11973 prototypes shall be provided.

```
11974 int      hcreate(size_t);
11975 void     hdestroy(void);
11976 ENTRY   *hsearch(ENTRY, ACTION);
11977 void     insque(void *, void *);
11978 void     *lfind(const void *, const void *, size_t *,
11979               size_t, int (*)(const void *, const void *));
11980 void     *lsearch(const void *, void *, size_t *,
11981                 size_t, int (*)(const void *, const void *));
11982 void     remque(void *);
11983 void     *tdelete(const void *restrict, posix_tnode **restrict,
11984                  int (*)(const void *, const void *));
11985 posix_tnode *tfind(const void *, posix_tnode *const *,
11986                  int (*)(const void *, const void *));
11987 posix_tnode *tsearch(const void *, posix_tnode **,
11988                     int (*)(const void *, const void *));
11989 void     twalk(const posix_tnode *,
11990               void (*)(const posix_tnode *, VISIT, int));
```

11991 **APPLICATION USAGE**

11992 None.

11993 **RATIONALE**

11994 Earlier versions of this standard explicitly used **void** for both node and key references where this  
11995 version now uses **posix\_tnode** for nodes and keeps **void** in the text referring only to keys. In  
11996 order to preserve backwards compatibility, this version defines **posix\_tnode** as an alias for **void**.  
11997 The change was made to make the function prototypes more easily understandable.

11998 **FUTURE DIRECTIONS**

11999 None.

12000 **SEE ALSO**

12001        [<sys/types.h>](#)

12002        XSH *hcreate()*, *insque()*, *lsearch()*, *tdelete()*

12003 **CHANGE HISTORY**

12004        First released in Issue 1. Derived from Issue 1 of the SVID.

12005 **Issue 6**

12006        The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete()* and  
12007        *tsearch()*.

12008        The **restrict** keyword is added to the prototype for *tdelete()*.

12009 **Issue 8**

12010        Austin Group Defect 1011 is applied, adding the **posix\_tnode** type and changing some  
12011        prototypes to use it instead of **void**.

12012 **NAME**

12013 semaphore.h — semaphores

12014 **SYNOPSIS**

12015 #include &lt;semaphore.h&gt;

12016 **DESCRIPTION**

12017 The <semaphore.h> header shall define the **sem\_t** type, used in performing semaphore  
 12018 operations. The semaphore may be implemented using a file descriptor, in which case  
 12019 applications are able to open up at least a total of {OPEN\_MAX} files and semaphores.

12020 The <semaphore.h> header shall define the **timespec** structure as described in <time.h>.

12021 The <semaphore.h> header shall define the symbolic constant SEM\_FAILED which shall have  
 12022 type **sem\_t** \*.

12023 The <semaphore.h> header shall define O\_CREAT and O\_EXCL as described in <fcntl.h>.

12024 The following shall be declared as functions and may also be defined as macros. Function  
 12025 prototypes shall be provided.

```

12026 int      sem_clockwait(sem_t *restrict, clockid_t,
12027                      const struct timespec *restrict);
12028 int      sem_close(sem_t *);
12029 int      sem_destroy(sem_t *);
12030 int      sem_getvalue(sem_t *restrict, int *restrict);
12031 int      sem_init(sem_t *, int, unsigned);
12032 sem_t *sem_open(const char *, int, ...);
12033 int      sem_post(sem_t *);
12034 int      sem_timedwait(sem_t *restrict, const struct timespec *restrict);
12035 int      sem_trywait(sem_t *);
12036 int      sem_unlink(const char *);
12037 int      sem_wait(sem_t *);

```

12038 Inclusion of the <semaphore.h> header may make visible symbols defined in the <fcntl.h> and  
 12039 <time.h> headers.

12040 **APPLICATION USAGE**

12041 None.

12042 **RATIONALE**

12043 None.

12044 **FUTURE DIRECTIONS**

12045 None.

12046 **SEE ALSO**

12047 &lt;fcntl.h&gt;, &lt;sys/types.h&gt;, &lt;time.h&gt;

12048 XSH *sem\_clockwait()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*,  
 12049 *sem\_post()*, *sem\_trywait()*, *sem\_unlink()*

12050 **CHANGE HISTORY**

12051 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

12052 **Issue 6**

12053 The &lt;semaphore.h&gt; header is marked as part of the Semaphores option.

12054 The Open Group Corrigendum U021/3 is applied, adding a description of SEM\_FAILED.

12055 The *sem\_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.

- 12056 The **restrict** keyword is added to the prototypes for *sem\_getvalue()* and *sem\_timedwait()*.
- 12057 **Issue 7**
- 12058 SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the **<time.h>**
- 12059 header.
- 12060 The **<semaphore.h>** header is moved from the Semaphores option to the Base.
- 12061 This reference page is clarified with respect to macros and symbolic constants.
- 12062 **Issue 8**
- 12063 Austin Group Defect 592 is applied, requiring **<semaphore.h>** to define the **timespec** structure.
- 12064 Austin Group Defect 593 is applied, adding **O\_CREAT** and **O\_EXCL**.
- 12065 Austin Group Defect 1216 is applied, adding *sem\_clockwait()*.



12066 **NAME**  
 12067 setjmp.h — stack environment declarations

12068 **SYNOPSIS**  
 12069 #include <setjmp.h>

12070 **DESCRIPTION**  
 12071 CX Some of the functionality described on this reference page extends the ISO C standard.  
 12072 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 12073 enable the visibility of these symbols in this header.

12074 CX The <setjmp.h> header shall define the array types **jmp\_buf** and **sigjmp\_buf**.  
 12075 The following shall be declared as functions and may also be defined as macros. Function  
 12076 prototypes shall be provided.

12077 \_Noreturn void longjmp(jmp\_buf, int);  
 12078 CX \_Noreturn void siglongjmp(sigjmp\_buf, int);

12079 The following may be declared as functions, or defined as macros, or both. If functions are  
 12080 declared, function prototypes shall be provided.

12081 int setjmp(jmp\_buf);  
 12082 CX int sigsetjmp(sigjmp\_buf, int);

12083 **APPLICATION USAGE**  
 12084 None.

12085 **RATIONALE**  
 12086 None.

12087 **FUTURE DIRECTIONS**  
 12088 None.

12089 **SEE ALSO**  
 12090 XSH Section 2.2 (on page 496), *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*

12091 **CHANGE HISTORY**  
 12092 First released in Issue 1.

12093 **Issue 6**  
 12094 Extensions beyond the ISO C standard are marked.

12095 **Issue 7**  
 12096 SD5-XBD-ERN-6 is applied.

12097 **Issue 8**  
 12098 Austin Group Defect 1302 is applied, adding *\_Noreturn* to *longjmp()* and *siglongjmp()*.  
 12099 Austin Group Defect 1330 is applied, removing obsolescent interfaces.



12139 The **signal** union shall be defined as:

12140 `int sival_int` Integer signal value.

12141 `void *sival_ptr` Pointer signal value.

12142 The <**signal.h**> header shall declare the SIGRTMIN and SIGRTMAX macros, which shall expand to positive integer expressions with type **int**, but which need not be constant expressions. These macros specify a range of signal numbers that are reserved for application use and for which the realtime signal behavior specified in this volume of POSIX.1-2024 is supported. The signal numbers in this range do not overlap any of the signals specified in the following table.

12143

12144

12145

12146

12147 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG\_MAX} signal numbers. The value of SIGRTMAX shall be less than the value returned by `sysconf(_SC_NSIG)`.

12148

12149 It is implementation-defined whether realtime signal behavior is supported for other signals.

12150 The <**signal.h**> header shall define the following symbolic constant. The value shall be suitable for use in **#if** preprocessing directives:

12151

12152 SIG2STR\_MAX Maximum size of a signal name returned by `sig2str()`, including the terminating null byte.

12153

12154 The <**signal.h**> header shall define the following macros that are used to refer to the signals that occur in the system. Signals defined here begin with the letters SIG followed by an uppercase letter. The macros shall expand to positive integer constant expressions with type **int** and distinct values less than the value of {NSIG\_MAX} defined in <**limits.h**>. The value 0 is reserved for use as the null signal (see `kill()`). Additional implementation-defined signals may occur in the system.

12155

12156

12157 CX

12158

12159

12160 The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, and SIGTERM to be defined. An implementation need not generate any of these six signals, except as a result of explicit use of interfaces that generate signals, such as `raise()`, `kill()`, the General Terminal Interface (see Section 11.1.9, on page 203), and the `kill` utility, unless otherwise stated (see, for example, XSH Section 2.8.3.3, on page 530).

12161

12162 CX

12163

12164

12165 The following signals shall be supported on all implementations (default actions are explained below the table):

12166

	Signal	Default Action	Description
12167	SIGABRT	A	Process abort signal.
12168	SIGALRM	T	Alarm clock.
12169	SIGBUS	A	Access to an undefined portion of a memory object.
12170	SIGCHLD	I	Child process terminated, stopped,
12171			or continued.
12172 XSI	SIGCONT	C	Continue executing, if stopped.
12173	SIGFPE	A	Erroneous arithmetic operation.
12174	SIGHUP	T	Hangup.
12175	SIGILL	A	Illegal instruction.
12176	SIGINT	T	Terminal interrupt signal.
12177	SIGKILL	T	Kill (cannot be caught or ignored).
12178	SIGPIPE	T	Write on a pipe with no one to read it.
12179	SIGQUIT	A	Terminal quit signal.
12180	SIGSEGV	A	Invalid memory reference.
12181	SIGSTOP	S	Stop executing (cannot be caught or ignored).
12182	SIGTERM	T	Termination signal.
12183	SIGTSTP	S	Terminal stop signal.
12184	SIGTTIN	S	Background process attempting read.
12185	SIGTTOU	S	Background process attempting write.
12186	SIGUSR1	T	User-defined signal 1.
12187	SIGUSR2	T	User-defined signal 2.
12188	SIGWINCH	I	Terminal window size changed.
12189	SIGSYS	A	Bad system call.
12190 XSI	SIGTRAP	A	Trace/breakpoint trap.
12191	SIGURG	I	High bandwidth data is available at a socket.
12192	SIGVTALRM	T	Virtual timer expired.
12193 XSI	SIGXCPU	A	CPU time limit exceeded.
12194	SIGXFSZ	A	File size limit exceeded.
12195			

12196 The default actions are as follows:

- 12197 T Abnormal termination of the process.
- 12198 A Abnormal termination of the process with additional actions.
- 12199 I Ignore the signal.
- 12200 S Stop the process.
- 12201 C Continue the process, if it is stopped; otherwise, ignore the signal.

12202 The effects on the process in each case are described in XSH Section 2.4.3 (on page 516).

12203 CX The <signal.h> header shall declare the **sigaction** structure, which shall include at least the following members:

```

12205 void (*sa_handler)(int) Pointer to a signal-catching function
12206 or one of the SIG_IGN or SIG_DFL.
12207 sigset_t sa_mask Set of signals to be blocked during execution
12208 of the signal handling function.
12209 int sa_flags Special flags.
12210 void (*sa_sigaction)(int, siginfo_t *, void *)
12211 Pointer to a signal-catching function.
```

12212	CX	The storage occupied by <i>sa_handler</i> and <i>sa_sigaction</i> may overlap, and a conforming application shall not use both simultaneously.	
12213			
12214		The <signal.h> header shall define the following macros which shall expand to integer constant expressions that need not be usable in #if preprocessing directives:	
12215			
12216	CX	<b>SIG_BLOCK</b>	The resulting set is the union of the current set and the signal set pointed to by the argument <i>set</i> .
12217			
12218	CX	<b>SIG_UNBLOCK</b>	The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument <i>set</i> .
12219			
12220	CX	<b>SIG_SETMASK</b>	The resulting set is the signal set pointed to by the argument <i>set</i> .
12221		The <signal.h> header shall also define the following symbolic constants:	
12222	CX	<b>SA_NOCLDSTOP</b>	Do not generate SIGCHLD when children stop
12223	XSI		or stopped children continue.
12224	XSI	<b>SA_ONSTACK</b>	Causes signal delivery to occur on an alternate stack.
12225	CX	<b>SA_RESETHAND</b>	Causes signal dispositions to be set to SIG_DFL on entry to signal handlers.
12226			
12227	CX	<b>SA_RESTART</b>	Causes certain functions to become restartable.
12228	CX	<b>SA_SIGINFO</b>	Causes extra information to be passed to signal handlers at the time of receipt of a signal.
12229			
12230	XSI	<b>SA_NOCLDWAIT</b>	Causes implementations not to create zombie processes or status information on child termination. See <i>sigaction()</i> .
12231			
12232	CX	<b>SA_NODEFER</b>	Causes signal not to be automatically blocked on entry to signal handler.
12233	XSI	<b>SS_ONSTACK</b>	Process is executing on an alternate signal stack.
12234	XSI	<b>SS_DISABLE</b>	Alternate signal stack is disabled.
12235	XSI	<b>MINSIGSTKSZ</b>	Minimum stack size for a signal handler.
12236	XSI	<b>SIGSTKSZ</b>	Default size in bytes for the alternate signal stack.
12237	CX	The <signal.h> header shall define the <b>mcontext_t</b> type through <b>typedef</b> .	
12238	CX	The <signal.h> header shall define the <b>ucontext_t</b> type as a structure that shall include at least the following members:	
12239			
12240		<code>ucontext_t *uc_link</code>	Pointer to the context that is resumed when this context returns.
12241			
12242		<code>sigset_t uc_sigmask</code>	The set of signals that are blocked when this context is active.
12243			
12244		<code>stack_t uc_stack</code>	The stack used by this context.
12245		<code>mcontext_t uc_mcontext</code>	A machine-specific representation of the saved context.
12246			
12247		The <signal.h> header shall define the <b>stack_t</b> type as a structure, which shall include at least the following members:	
12248			
12249		<code>void *ss_sp</code>	Stack base or pointer.
12250		<code>size_t ss_size</code>	Stack size.
12251		<code>int ss_flags</code>	Flags.

12252 CX The <signal.h> header shall define the **siginfo\_t** type as a structure, which shall include at least  
12253 the following members:

12254	CX	int	si_signo	Signal number.
12255		int	si_code	Signal code.
12256	XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with 12257 this signal, as described in <errno.h>.
12258	CX	pid_t	si_pid	Sending process ID.
12259		uid_t	si_uid	Real user ID of sending process.
12260		void	*si_addr	Address that caused fault.
12261		int	si_status	Exit value or signal.
12262		union sigval	si_value	Signal value.

12263 CX The <signal.h> header shall define the symbolic constants in the **Code** column of the following  
12264 table for use as values of *si\_code* that are signal-specific or non-signal-specific reasons why the  
12265 signal was generated.

	Signal	Code	Reason
12266	SIGILL	ILL_ILLOPC	Illegal opcode.
12267 CX		ILL_ILLOPN	Illegal operand.
12268		ILL_ILLADR	Illegal addressing mode.
12269		ILL_ILLTRP	Illegal trap.
12270		ILL_PRVOPC	Privileged opcode.
12271		ILL_PRVREG	Privileged register.
12272		ILL_COPROC	Coprocessor error.
12273		ILL_BADSTK	Internal stack error.
12274	SIGFPE	FPE_INTDIV	Integer divide by zero.
12275		FPE_INTOVF	Integer overflow.
12276		FPE_FLTDIV	Floating-point divide by zero.
12277		FPE_FLTOVF	Floating-point overflow.
12278		FPE_FLTUND	Floating-point underflow.
12279		FPE_FLTRES	Floating-point inexact result.
12280		FPE_FLTINV	Invalid floating-point operation.
12281		FPE_FLTSUB	Subscript out of range.
12282	SIGSEGV	SEGV_MAPERR	Address not mapped to object.
12283		SEGV_ACCERR	Invalid permissions for mapped object.
12284	SIGBUS	BUS_ADRALN	Invalid address alignment.
12285		BUS_ADRERR	Nonexistent physical address.
12286		BUS_OBJERR	Object-specific hardware error.
12287	SIGTRAP	TRAP_BRKPT	Process breakpoint.
12288 XSI		TRAP_TRACE	Process trace trap.
12289	SIGCHLD	CLD_EXITED	Child has exited.
12290 CX		CLD_KILLED	Child has terminated abnormally with no additional actions.
12291		CLD_DUMPED	Child has terminated abnormally and additional actions may have been taken.
12292		CLD_TRAPPED	Traced child has trapped.
12293		CLD_STOPPED	Child has stopped.
12294		CLD_CONTINUED	Stopped child has continued.
12295	Any	SI_USER	Signal sent by <i>kill()</i> .
12296		SI_QUEUE	Signal sent by <i>sigqueue()</i> .
12297		SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
12298		SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
12299		SI_MESGQ	Signal generated by arrival of a message on an empty message queue
12300			
12301			
12302			
12303			
12304 CX	Implementations may support additional <i>si_code</i> values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.		
12305			
12306			
12307			
12308			

12309 CX In addition, the following signal-specific information shall be available:

Signal	Member	Value
SIGILL SIGFPE	<b>void * <i>si_addr</i></b>	Address of faulting instruction.
SIGSEGV SIGBUS	<b>void * <i>si_addr</i></b>	Address of faulting memory reference.
SIGCHLD	<b>pid_t <i>si_pid</i></b> <b>int <i>si_status</i></b>	Child process ID. If <i>si_code</i> is equal to CLD_EXITED, then <i>si_status</i> holds the exit value of the process; otherwise, it is equal to the signal that caused the process to change state. The exit value in <i>si_status</i> shall be equal to the full exit value (that is, the value passed to <i>_exit()</i> , <i>_Exit()</i> , or <i>exit()</i> , or returned from <i>main()</i> ); it shall not be limited to the least significant eight bits of the value.
	<b>uid_t <i>si_uid</i></b>	Real user ID of the process that sent the signal.

12323 For some implementations, the value of *si\_addr* may be inaccurate.

12324 The following shall be declared as functions and may also be defined as macros. Function  
12325 prototypes shall be provided.

```

12326 CX int kill(pid_t, int);
12327 XSI int killpg(pid_t, int);
12328 CX void psiginfo(const siginfo_t *, const char *);
12329 void psignal(int, const char *);
12330 int pthread_kill(pthread_t, int);
12331 int pthread_sigmask(int, const sigset_t *restrict,
12332 sigset_t *restrict);
12333 int raise(int);
12334 CX int sig2str(int, char *);
12335 int sigaction(int, const struct sigaction *restrict,
12336 struct sigaction *restrict);
12337 int sigaddset(sigset_t *, int);
12338 XSI int sigaltstack(const stack_t *restrict, stack_t *restrict);
12339 CX int sigdelset(sigset_t *, int);
12340 int sigemptyset(sigset_t *);
12341 int sigfillset(sigset_t *);
12342 int sigismember(const sigset_t *, int);
12343 void (*signal(int, void (*)(int)))(int);
12344 CX int sigpending(sigset_t *);
12345 int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
12346 int sigqueue(pid_t, int, union sigval);
12347 int sigsuspend(const sigset_t *);
12348 int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
12349 const struct timespec *restrict);
12350 int sigwait(const sigset_t *restrict, int *restrict);
12351 int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
12352 int str2sig(const char *restrict, int *restrict);

```

12353 CX Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.



12354 **APPLICATION USAGE**

12355 On systems not supporting the XSI option, the *si\_pid* and *si\_uid* members of **siginfo\_t** are only  
 12356 required to be valid when *si\_code* is SI\_USER or SI\_QUEUE. On XSI-conforming systems, they  
 12357 are also valid for all *si\_code* values less than or equal to 0; however, it is unspecified whether  
 12358 SI\_USER and SI\_QUEUE have values less than or equal to zero, and therefore XSI applications  
 12359 should check whether *si\_code* has the value SI\_USER or SI\_QUEUE or is less than or equal to 0 to  
 12360 tell whether *si\_pid* and *si\_uid* are valid.

12361 **RATIONALE**

12362 None.

12363 **FUTURE DIRECTIONS**

12364 None.

12365 **SEE ALSO**

12366 <errno.h>, <limits.h>, <sys/types.h>, <time.h>

12367 XSH Section 2.2 (on page 496), *alarm()*, *kill()*, *killpg()*, *psiginfo()*, *pthread\_kill()*, *pthread\_sigmask()*,  
 12368 *raise()*, *sig2str()*, *sigaction()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,  
 12369 *sigismember()*, *signal()*, *sigpending()*, *sigqueue()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*,  
 12370 *timer\_create()*, *wait()*, *waitid()*

12371 XCU *kill*

12372 **CHANGE HISTORY**

12373 First released in Issue 1.

12374 **Issue 5**

12375 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 12376 Threads Extension.

12377 The default action for SIGURG is changed from i to iii. The function prototype for *sigmask()* is  
 12378 removed.

12379 **Issue 6**

12380 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for  
 12381 abnormal termination is clarified.

12382 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()*  
 12383 function.

12384 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev\_notify\_function*  
 12385 function member of the **sigevent** structure.

12386 The following new requirements on POSIX implementations derive from alignment with the  
 12387 Single UNIX Specification:

- 12388 • The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now  
 12389 mandated. This is also a FIPS requirement.
- 12390 • The **pid\_t** definition is mandated.

12391 The RT markings are changed to RTS to denote that the semantics are part of the Realtime  
 12392 Signals Extension option.

12393 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,  
 12394 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

12395 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from  
 12396 <time.h> may be made visible when <signal.h> is included.

- 12397 Extensions beyond the ISO C standard are marked.
- 12398 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive  
12399 text for members of the **sigaction** structure.
- 12400 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of  
12401 the *sa\_sigaction* member of the **sigaction** structure.
- 12402 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/24 is applied, reworking the ordering of  
12403 the **siginfo\_t** type structure in the DESCRIPTION. This is an editorial change and no normative  
12404 change is intended.
- 12405 **Issue 7**
- 12406 SD5-XBD-ERN-5 is applied.
- 12407 SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed  
12408 at the same time as the LEGACY *sigstack()* function.
- 12409 SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size\_t** type.
- 12410 Austin Group Interpretation 1003.1-2001 #034 is applied.
- 12411 The **ucontext\_t** and **mcontext\_t** structures are added here from the obsolescent **<ucontext.h>**  
12412 header.
- 12413 The *psiginfo()* and *psignal()* functions are added from The Open Group Technical Standard, 2006,  
12414 Extended API Set Part 1.
- 12415 The SIGPOLL and SIGPROF signals and text relating to the XSI STREAMS option are marked  
12416 obsolescent.
- 12417 The SA\_RESETHAND, SA\_RESTART, SA\_SIGINFO, SA\_NOCLDWAIT, and SA\_NODEFER  
12418 constants are moved from the XSI option to the Base.
- 12419 Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 12420 This reference page is clarified with respect to macros and symbolic constants, and declarations  
12421 for the **pthread\_attr\_t**, **pthread\_t**, and **uid\_t** types and the **timespec** structure are added.
- 12422 SIGRTMIN and SIGRTMAX are required to be positive integer expressions.
- 12423 The APPLICATION USAGE section is updated to describe the *si\_pid* and *si\_uid* members of  
12424 **siginfo\_t**.
- 12425 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0062 [208], XBD/TC1-2008/0063 [80],  
12426 and XBD/TC1-2008/0064 [157] are applied.
- 12427 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0070 [536], XBD/TC2-2008/0071 [690],  
12428 XBD/TC2-2008/0072 [594], XBD/TC2-2008/0073 [844], and XBD/TC2-2008/0074 [536] are  
12429 applied.
- 12430 **Issue 8**
- 12431 Austin Group Defect 741 is applied, restricting the value of SIGRTMAX to less than the value  
12432 returned by *sysconf(\_SC\_NSIG)* and the value of macros that are used to refer to the signals to  
12433 less than {NSIG\_MAX}.
- 12434 Austin Group Defect 1138 is applied, adding *sig2str()* and *str2sig()*.
- 12435 Austin Group Defect 1141 is applied, changing the descriptions of CLD\_KILLED and  
12436 CLD\_DUMPED.
- 12437 Austin Group Defect 1151 is applied, adding SIGWINCH.

12438 Austin Group Defect 1215 is applied, removing XSI shading from text relating to abnormal  
12439 process termination with additional actions.

12440 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

12441 Austin Group Defect 1775 is applied, changing the description of the *si\_addr* member of  
12442 **siginfo\_t**.

12443 **NAME**

12444 spawn.h — spawn (ADVANCED REALTIME)

12445 **SYNOPSIS**12446 SPN `#include <spawn.h>`12447 **DESCRIPTION**12448 The **<spawn.h>** header shall define the **posix\_spawnattr\_t** and **posix\_spawn\_file\_actions\_t**  
12449 types used in performing spawn operations.12450 The **<spawn.h>** header shall define the **mode\_t** and **pid\_t** types as described in **<sys/types.h>**.12451 The **<spawn.h>** header shall define the **sigset\_t** type as described in **<signal.h>**.12452 The tag **sched\_param** shall be declared as naming an incomplete structure type, the contents of  
12453 which are described in the **<sched.h>** header.12454 The **<spawn.h>** header shall define the following symbolic constants for use as the flags that  
12455 may be set in a **posix\_spawnattr\_t** object using the *posix\_spawnattr\_setflags()* function:

12456 POSIX\_SPAWN\_RESETIDS

12457 POSIX\_SPAWN\_SETPGROUP

12458 PS POSIX\_SPAWN\_SETSCHEDPARAM

12459 POSIX\_SPAWN\_SETSCHEDULER

12460 POSIX\_SPAWN\_SETSID

12461 POSIX\_SPAWN\_SETSIGDEF

12462 POSIX\_SPAWN\_SETSIGMASK

12463 The following shall be declared as functions and may also be defined as macros. Function  
12464 prototypes shall be provided.

```

12465 int    posix_spawn(pid_t *restrict, const char *restrict,
12466                  const posix_spawn_file_actions_t *restrict,
12467                  const posix_spawnattr_t *restrict, char *const [restrict],
12468                  char *const [restrict]);
12469 int    posix_spawn_file_actions_addchdir(posix_spawn_file_actions_t
12470                  *restrict, const char *restrict);
12471 int    posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
12472                  int);
12473 int    posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
12474                  int, int);
12475 int    posix_spawn_file_actions_addfchdir(posix_spawn_file_actions_t *,
12476                  int);
12477 int    posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
12478                  *restrict, int, const char *restrict, int, mode_t);
12479 int    posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
12480 int    posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
12481 int    posix_spawnattr_destroy(posix_spawnattr_t *);
12482 int    posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
12483                  short *restrict);
12484 int    posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
12485                  pid_t *restrict);
12486 PS int    posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
12487                  struct sched_param *restrict);
12488 int    posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
12489                  int *restrict);

```

```

12490     int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
12491                                         sigset_t *restrict);
12492     int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
12493                                       sigset_t *restrict);
12494     int    posix_spawnattr_init(posix_spawnattr_t *);
12495     int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
12496     int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
12497 PS     int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
12498                                             const struct sched_param *restrict);
12499     int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
12500     int    posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
12501                                         sigset_t *restrict);
12502     int    posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
12503                                       sigset_t *restrict);
12504     int    posix_spawn(pid_t *restrict, const char *restrict,
12505                       const posix_spawn_file_actions_t *restrict,
12506                       const posix_spawnattr_t *restrict,
12507                       char *const [restrict], char *const [restrict]);

```

12508 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h> and  
 12509 <signal.h> headers.

#### 12510 APPLICATION USAGE

12511 None.

#### 12512 RATIONALE

12513 None.

#### 12514 FUTURE DIRECTIONS

12515 None.

#### 12516 SEE ALSO

12517 [<sched.h>](#), [<semaphore.h>](#), [<signal.h>](#), [<sys/types.h>](#)

12518 XSH [posix\\_spawn\(\)](#), [posix\\_spawn\\_file\\_actions\\_addclose\(\)](#), [posix\\_spawn\\_file\\_actions\\_adddup2\(\)](#),  
 12519 [posix\\_spawn\\_file\\_actions\\_destroy\(\)](#), [posix\\_spawnattr\\_destroy\(\)](#), [posix\\_spawnattr\\_getflags\(\)](#),  
 12520 [posix\\_spawnattr\\_getpgroup\(\)](#), [posix\\_spawnattr\\_getschedparam\(\)](#), [posix\\_spawnattr\\_getschedpolicy\(\)](#),  
 12521 [posix\\_spawnattr\\_getsigdefault\(\)](#), [posix\\_spawnattr\\_getsigmask\(\)](#)

#### 12522 CHANGE HISTORY

12523 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

12524 The **restrict** keyword is added to the prototypes for [posix\\_spawn\(\)](#),  
 12525 [posix\\_spawn\\_file\\_actions\\_addopen\(\)](#), [posix\\_spawnattr\\_getsigdefault\(\)](#), [posix\\_spawnattr\\_getflags\(\)](#),  
 12526 [posix\\_spawnattr\\_getpgroup\(\)](#), [posix\\_spawnattr\\_getschedparam\(\)](#), [posix\\_spawnattr\\_getschedpolicy\(\)](#),  
 12527 [posix\\_spawnattr\\_getsigmask\(\)](#), [posix\\_spawnattr\\_setsigdefault\(\)](#), [posix\\_spawnattr\\_setschedparam\(\)](#),  
 12528 [posix\\_spawnattr\\_setsigmask\(\)](#), and [posix\\_spawn\(\)](#).

#### 12529 Issue 7

12530 This reference page is clarified with respect to macros and symbolic constants, and declarations  
 12531 for the **mode\_t**, **pid\_t**, and **sigset\_t** types are added.

#### 12532 Issue 8

12533 Austin Group Defect 1044 is applied, adding POSIX\_SPAWN\_SETSID.

12534 Austin Group Defect 1208 is applied, adding [posix\\_spawn\\_file\\_actions\\_addchdir\(\)](#) and  
 12535 [posix\\_spawn\\_file\\_actions\\_addfchdir\(\)](#).

12536  
12537

Austin Group Defect 1328 is applied, adding the **restrict** keyword to the third parameter of *posix\_spawn()* and *posix\_spawnp()*.

12538 **NAME**

12539           stdalign.h — alignment macros

12540 **SYNOPSIS**

12541           #include &lt;stdalign.h&gt;

12542 **DESCRIPTION**

12543 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
12544       conflict between the requirements described here and the ISO C standard is unintentional. This  
12545       volume of POSIX.1-2024 defers to the ISO C standard.

12546       The &lt;stdalign.h&gt; header shall define the following macros:

12547       alignas       Expands to **\_Alignas**12548       alignof       Expands to **\_Alignof**

12549       \_\_alignas\_is\_defined

12550               Expands to the integer constant 1

12551       \_\_alignof\_is\_defined

12552               Expands to the integer constant 1

12553       The `__alignas_is_defined` and `__alignof_is_defined` macros shall be suitable for use in `#if`  
12554       preprocessing directives.

12555 **APPLICATION USAGE**

12556       None.

12557 **RATIONALE**

12558       None.

12559 **FUTURE DIRECTIONS**

12560       None.

12561 **SEE ALSO**

12562       None.

12563 **CHANGE HISTORY**

12564       First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

12565 **NAME**

12566           stdatomic.h — atomics

12567 **SYNOPSIS**

12568           #include &lt;stdatomic.h&gt;

12569 **DESCRIPTION**

12570 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
12571           conflict between the requirements described here and the ISO C standard is unintentional. This  
12572           volume of POSIX.1-2024 defers to the ISO C standard.

12573           Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide this header  
12574           nor support any of its facilities.

12575           The **<stdatomic.h>** header shall define the **atomic\_flag** type as a structure type. This type  
12576           provides the classic test-and-set functionality. It shall have two states, set and clear. Operations  
12577           on an object of type **atomic\_flag** shall be lock free.

12578           The **<stdatomic.h>** header shall define each of the atomic integer types in the following table as  
12579           a type that has the same representation and alignment requirements as the corresponding direct  
12580           type.

12581           **Note:**       The same representation and alignment requirements are meant to imply interchangeability as  
12582           arguments to functions, return values from functions, and members of unions.



	Atomic type name	Direct type
12583		
12584	<code>atomic_bool</code>	<code>_Atomic_Bool</code>
12585	<code>atomic_char</code>	<code>_Atomic char</code>
12586	<code>atomic_schar</code>	<code>_Atomic signed char</code>
12587	<code>atomic_uchar</code>	<code>_Atomic unsigned char</code>
12588	<code>atomic_short</code>	<code>_Atomic short</code>
12589	<code>atomic_ushort</code>	<code>_Atomic unsigned short</code>
12590	<code>atomic_int</code>	<code>_Atomic int</code>
12591	<code>atomic_uint</code>	<code>_Atomic unsigned int</code>
12592	<code>atomic_long</code>	<code>_Atomic long</code>
12593	<code>atomic_ulong</code>	<code>_Atomic unsigned long</code>
12594	<code>atomic_llong</code>	<code>_Atomic long long</code>
12595	<code>atomic_ullong</code>	<code>_Atomic unsigned long long</code>
12596	<code>atomic_char16_t</code>	<code>_Atomic char16_t</code>
12597	<code>atomic_char32_t</code>	<code>_Atomic char32_t</code>
12598	<code>atomic_wchar_t</code>	<code>_Atomic wchar_t</code>
12599	<code>atomic_int_least8_t</code>	<code>_Atomic int_least8_t</code>
12600	<code>atomic_uint_least8_t</code>	<code>_Atomic uint_least8_t</code>
12601	<code>atomic_int_least16_t</code>	<code>_Atomic int_least16_t</code>
12602	<code>atomic_uint_least16_t</code>	<code>_Atomic uint_least16_t</code>
12603	<code>atomic_int_least32_t</code>	<code>_Atomic int_least32_t</code>
12604	<code>atomic_uint_least32_t</code>	<code>_Atomic uint_least32_t</code>
12605	<code>atomic_int_least64_t</code>	<code>_Atomic int_least64_t</code>
12606	<code>atomic_uint_least64_t</code>	<code>_Atomic uint_least64_t</code>
12607	<code>atomic_int_fast8_t</code>	<code>_Atomic int_fast8_t</code>
12608	<code>atomic_uint_fast8_t</code>	<code>_Atomic uint_fast8_t</code>
12609	<code>atomic_int_fast16_t</code>	<code>_Atomic int_fast16_t</code>
12610	<code>atomic_uint_fast16_t</code>	<code>_Atomic uint_fast16_t</code>
12611	<code>atomic_int_fast32_t</code>	<code>_Atomic int_fast32_t</code>
12612	<code>atomic_uint_fast32_t</code>	<code>_Atomic uint_fast32_t</code>
12613	<code>atomic_int_fast64_t</code>	<code>_Atomic int_fast64_t</code>
12614	<code>atomic_uint_fast64_t</code>	<code>_Atomic uint_fast64_t</code>
12615	<code>atomic_intptr_t</code>	<code>_Atomic intptr_t</code>
12616	<code>atomic_uintptr_t</code>	<code>_Atomic uintptr_t</code>
12617	<code>atomic_size_t</code>	<code>_Atomic size_t</code>
12618	<code>atomic_ptrdiff_t</code>	<code>_Atomic ptrdiff_t</code>
12619	<code>atomic_intmax_t</code>	<code>_Atomic intmax_t</code>
12620	<code>atomic_uintmax_t</code>	<code>_Atomic uintmax_t</code>

12621 The <stdatomic.h> header shall define the `memory_order` type as an enumerated type whose  
 12622 enumerators shall include at least the following:

- 12623 `memory_order_relaxed`
- 12624 `memory_order_consume`
- 12625 `memory_order_acquire`
- 12626 `memory_order_release`
- 12627 `memory_order_acq_rel`
- 12628 `memory_order_seq_cst`

12629 The <stdatomic.h> header shall define the following atomic lock-free macros:

- 12630 `ATOMIC_BOOL_LOCK_FREE`
- 12631 `ATOMIC_CHAR_LOCK_FREE`
- 12632 `ATOMIC_CHAR16_T_LOCK_FREE`

```

12633     ATOMIC_CHAR32_T_LOCK_FREE
12634     ATOMIC_WCHAR_T_LOCK_FREE
12635     ATOMIC_SHORT_LOCK_FREE
12636     ATOMIC_INT_LOCK_FREE
12637     ATOMIC_LONG_LOCK_FREE
12638     ATOMIC_LLONG_LOCK_FREE
12639     ATOMIC_POINTER_LOCK_FREE

```

12640 which shall expand to constant expressions suitable for use in **#if** preprocessing directives and  
 12641 which shall indicate the lock-free property of the corresponding atomic types (both signed and  
 12642 unsigned). A value of 0 shall indicate that the type is never lock-free; a value of 1 shall indicate  
 12643 that the type is sometimes lock-free; a value of 2 shall indicate that the type is always lock-free.

12644 The <stdatomic.h> header shall define the macro `ATOMIC_FLAG_INIT` which shall expand to  
 12645 an initializer for an object of type `atomic_flag`. This macro shall initialize an `atomic_flag` to the  
 12646 clear state. An `atomic_flag` that is not explicitly initialized with `ATOMIC_FLAG_INIT` is initially  
 12647 in an indeterminate state.

12648 OB The <stdatomic.h> header shall define the macro `ATOMIC_VAR_INIT(value)` which shall  
 12649 expand to a token sequence suitable for initializing an atomic object of a type that is  
 12650 initialization-compatible with the non-atomic type of its *value* argument. An atomic object with  
 12651 automatic storage duration that is not explicitly initialized is initially in an indeterminate state.

12652 The <stdatomic.h> header shall define the macro `kill_dependency()` which shall behave as  
 12653 described in *kill\_dependency()*.

12654 The <stdatomic.h> header shall declare the following generic functions, where *A* refers to an  
 12655 atomic type, *C* refers to its corresponding non-atomic type, and *M* is *C* for atomic integer types  
 12656 or `ptrdiff_t` for atomic pointer types.

```

12657     _Bool    atomic_compare_exchange_strong(volatile A *, C *, C);
12658     _Bool    atomic_compare_exchange_strong_explicit(volatile A *,
12659             C *, C, memory_order, memory_order);
12660     _Bool    atomic_compare_exchange_weak(volatile A *, C *, C);
12661     _Bool    atomic_compare_exchange_weak_explicit(volatile A *, C *,
12662             C, memory_order, memory_order);
12663     C        atomic_exchange(volatile A *, C);
12664     C        atomic_exchange_explicit(volatile A *, C, memory_order);
12665     C        atomic_fetch_add(volatile A *, M);
12666     C        atomic_fetch_add_explicit(volatile A *, M,
12667             memory_order);
12668     C        atomic_fetch_and(volatile A *, M);
12669     C        atomic_fetch_and_explicit(volatile A *, M,
12670             memory_order);
12671     C        atomic_fetch_or(volatile A *, M);
12672     C        atomic_fetch_or_explicit(volatile A *, M, memory_order);
12673     C        atomic_fetch_sub(volatile A *, M);
12674     C        atomic_fetch_sub_explicit(volatile A *, M,
12675             memory_order);
12676     C        atomic_fetch_xor(volatile A *, M);
12677     C        atomic_fetch_xor_explicit(volatile A *, M,
12678             memory_order);
12679     void     atomic_init(volatile A *, C);
12680     _Bool    atomic_is_lock_free(const volatile A *);
12681     C        atomic_load(const volatile A *);

```

```

12682     C      atomic_load_explicit(const volatile A *, memory_order);
12683     void    atomic_store(volatile A *, C);
12684     void    atomic_store_explicit(volatile A *, C, memory_order);

```

12685 It is unspecified whether any generic function declared in <stdatomic.h> is a macro or an  
 12686 identifier declared with external linkage. If a macro definition is suppressed in order to access an  
 12687 actual function, or a program defines an external identifier with the name of a generic function,  
 12688 the behavior is undefined.

12689 The following shall be declared as functions and may also be defined as macros. Function  
 12690 prototypes shall be provided.

```

12691     void    atomic_flag_clear(volatile atomic_flag *);
12692     void    atomic_flag_clear_explicit(volatile atomic_flag *,
12693                                     memory_order);
12694     _Bool   atomic_flag_test_and_set(volatile atomic_flag *);
12695     _Bool   atomic_flag_test_and_set_explicit(
12696                                     volatile atomic_flag *, memory_order);
12697     void    atomic_signal_fence(memory_order);
12698     void    atomic_thread_fence(memory_order);

```

#### 12699 APPLICATION USAGE

12700 None.

#### 12701 RATIONALE

12702 Since operations on the **atomic\_flag** type are lock free, the operations should also be address-  
 12703 free. No other type requires lock-free operations, so the **atomic\_flag** type is the minimum  
 12704 hardware-implemented type needed to conform to this standard. The remaining types can be  
 12705 emulated with **atomic\_flag**, though with less than ideal properties.

12706 The representation of atomic integer types need not have the same size as their corresponding  
 12707 regular types. They should have the same size whenever possible, as it eases effort required to  
 12708 port existing code.

#### 12709 FUTURE DIRECTIONS

12710 The ISO C standard states that the macro `ATOMIC_VAR_INIT` is an obsolescent feature. This  
 12711 macro may be removed in a future version of this standard.

#### 12712 SEE ALSO

12713 [Section 4.15.1](#)

12714 XSH `atomic_compare_exchange_strong()`, `atomic_exchange()`, `atomic_fetch_add()`, `atomic_flag_clear()`,  
 12715 `atomic_flag_test_and_set()`, `atomic_init()`, `atomic_is_lock_free()`, `atomic_load()`, `atomic_signal_fence()`,  
 12716 `atomic_store()`, `kill_dependency()`.

#### 12717 CHANGE HISTORY

12718 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

12719 **NAME**12720         **stdarg.h** — handle variable argument list12721 **SYNOPSIS**

```
12722         #include <stdarg.h>
12723         void va_start(va_list ap, argN);
12724         void va_copy(va_list dest, va_list src);
12725         type va_arg(va_list ap, type);
12726         void va_end(va_list ap);
```

12727 **DESCRIPTION**

12728 CX         The functionality described on this reference page is aligned with the ISO C standard. Any  
12729         conflict between the requirements described here and the ISO C standard is unintentional. This  
12730         volume of POSIX.1-2024 defers to the ISO C standard.

12731         The **<stdarg.h>** header shall contain a set of macros which allows portable functions that accept  
12732         variable argument lists to be written. Functions that have variable argument lists (such as  
12733         *printf()*) but do not use these macros are inherently non-portable, as different systems use  
12734         different argument-passing conventions.

12735         The **<stdarg.h>** header shall define the **va\_list** type for variables used to traverse the list.

12736         The *va\_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to  
12737         *va\_arg()*.

12738         The *va\_copy()* macro initializes *dest* as a copy of *src*, as if the *va\_start()* macro had been applied  
12739         to *dest* followed by the same sequence of uses of the *va\_arg()* macro as had previously been used  
12740         to reach the present state of *src*. Neither the *va\_copy()* nor *va\_start()* macro shall be invoked to  
12741         reinitialize *dest* without an intervening invocation of the *va\_end()* macro for the same *dest*.

12742         The object *ap* may be passed as an argument to another function; if that function invokes the  
12743         *va\_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall  
12744         be passed to the *va\_end()* macro prior to any further reference to *ap*. The parameter *argN* is the  
12745         identifier of the rightmost parameter in the variable parameter list in the function definition (the  
12746         one just before the *...*). If the parameter *argN* is declared with the **register** storage class, with a  
12747         function type or array type, or with a type that is not compatible with the type that results after  
12748         application of the default argument promotions, the behavior is undefined.

12749         The *va\_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation  
12750         of *va\_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*  
12751         parameter shall be a type name specified such that the type of a pointer to an object that has the  
12752         specified type can be obtained simply by postfixing a '\*' to type. If there is no actual next  
12753         argument, or if *type* is not compatible with the type of the actual next argument (as promoted  
12754         according to the default argument promotions), the behavior is undefined, except for the  
12755         following cases:

- 12756                 • One type is a signed integer type, the other type is the corresponding unsigned integer  
12757                 type, and the value is representable in both types.
- 12758                 • One type is a pointer to **void** and the other is a pointer to a character type.
- 12759 XSI            • Both types are pointers.

12760         Different types can be mixed, but it is up to the routine to know what type of argument is  
12761         expected.

12762         The *va\_end()* macro is used to clean up; it invalidates *ap* for use (unless *va\_start()* or *va\_copy()* is  
12763         invoked again).

12764 Each invocation of the *va\_start()* and *va\_copy()* macros shall be matched by a corresponding  
 12765 invocation of the *va\_end()* macro in the same function.

12766 Multiple traversals, each bracketed by *va\_start()* ... *va\_end()*, are possible.

#### 12767 EXAMPLES

12768 This example is a possible implementation of *execl()*:

```
12769 #include <stdarg.h>
12770 #define MAXARGS 31
12771 /*
12772  * execl is called by
12773  * execl(file, arg1, arg2, ..., (char *) (0));
12774  */
12775 int execl(const char *file, const char *args, ...)
12776 {
12777     va_list ap;
12778     char *array[MAXARGS + 1];
12779     int argno = 0;
12780
12781     va_start(ap, args);
12782     while (args != 0 && argno < MAXARGS)
12783     {
12784         array[argno++] = args;
12785         args = va_arg(ap, const char *);
12786     }
12787     array[argno] = (char *) 0;
12788     va_end(ap);
12789     return execv(file, array);
12790 }
```

#### 12790 APPLICATION USAGE

12791 It is up to the calling routine to communicate to the called routine how many arguments there  
 12792 are, since it is not always possible for the called routine to determine this in any other way. For  
 12793 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell  
 12794 how many arguments are there by the *format* argument.

#### 12795 RATIONALE

12796 None.

#### 12797 FUTURE DIRECTIONS

12798 None.

#### 12799 SEE ALSO

12800 XSH *exec*, *fprintf()*

#### 12801 CHANGE HISTORY

12802 First released in Issue 4. Derived from the ANSI C standard.

#### 12803 Issue 6

12804 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12805 **NAME**

12806           stdbool.h — boolean type and values

12807 **SYNOPSIS**

12808           #include &lt;stdbool.h&gt;

12809 **DESCRIPTION**

12810 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
12811       conflict between the requirements described here and the ISO C standard is unintentional. This  
12812       volume of POSIX.1-2024 defers to the ISO C standard.

12813       The **<stdbool.h>** header shall define the following macros:12814       bool     Expands to **\_Bool**.

12815       true     Expands to the integer constant 1.

12816       false    Expands to the integer constant 0.

12817       \_\_bool\_true\_false\_are\_defined

12818           Expands to the integer constant 1.

12819       The macros true, false and \_\_bool\_true\_false\_are\_defined shall be suitable for use in **#if**  
12820       preprocessing directives.

12821 OB       An application can undefine and then possibly redefine the macros bool, true, and false.

12822 **APPLICATION USAGE**

12823       None.

12824 **RATIONALE**

12825       None.

12826 **FUTURE DIRECTIONS**

12827       The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature  
12828       and may be removed in a future version.

12829 **SEE ALSO**

12830       None.

12831 **CHANGE HISTORY**

12832       First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

12833 **Issue 8**

12834       Austin Group Defect 1257 is applied, correcting a mismatch with the ISO C standard by adding  
12835       the requirement for the integer-valued macros to be suitable for use in **#if** preprocessing  
12836       directives.

12837       Austin Group Defect 1302 is applied, adding OB shading to the statement that an application  
12838       can “undefine and then possibly redefine the macros bool, true, and false”.

12839 **NAME**  
 12840         stddef.h — standard type definitions

12841 **SYNOPSIS**  
 12842         #include <stddef.h>

12843 **DESCRIPTION**  
 12844 CX         The functionality described on this reference page is aligned with the ISO C standard. Any  
 12845             conflict between the requirements described here and the ISO C standard is unintentional. This  
 12846             volume of POSIX.1-2024 defers to the ISO C standard.

12847         The <stddef.h> header shall define the following macros:

12848 CX         **NULL**         Null pointer constant. The macro shall expand to an integer constant expression  
 12849             with the value 0 cast to type **void \***. Additionally, any pointer object whose  
 12850             representation has all bits set to zero, perhaps by *memset()* to 0 or by *calloc()*, shall  
 12851             be treated as a null pointer.

12852             *offsetof(type, member-designator)*  
 12853             Integer constant expression of type **size\_t**, the value of which is the offset in bytes  
 12854             to the structure member (*member-designator*), from the beginning of its structure  
 12855             (*type*).

12856         The <stddef.h> header shall define the following types:

12857         **max\_align\_t** Object type whose alignment is the greatest fundamental alignment.

12858         **ptrdiff\_t**         Signed integer type of the result of subtracting two pointers.

12859         **wchar\_t**         Integer type whose range of values can represent distinct codes for all members of  
 12860             the largest extended character set specified among the supported locales; the null  
 12861             character shall have the code value zero. Each member of the basic character set  
 12862             shall have a code value equal to its value when used as the lone character in an  
 12863             integer character constant if an implementation does not define  
 12864             \_\_STDC\_MB\_MIGHT\_NEQ\_WC\_\_.

12865         **size\_t**             Unsigned integer type of the result of the *sizeof* operator.

12866         The implementation shall support one or more programming environments in which the widths  
 12867         of **ptrdiff\_t**, **size\_t**, and **wchar\_t** are no greater than the width of type **long**. The names of these  
 12868         programming environments can be obtained using the *confstr()* function or the *getconf* utility.

12869 **APPLICATION USAGE**  
 12870         None.

12871 **RATIONALE**  
 12872         The ISO C standard does not require the **NULL** macro to include the cast to type **void \*** and  
 12873         specifies that the **NULL** macro be implementation-defined. POSIX.1-2024 requires the cast and  
 12874         therefore need not be implementation-defined.

12875         Likewise, the ISO C standard does not require a pointer object whose representation has all bits  
 12876         set to zero to be treated as a null pointer. While there has been historical hardware where non-  
 12877         zero patterns were more efficient for use as the canonical null pointer, no known POSIX system  
 12878         has tried to target such hardware. However, though unlikely in modern hardware, a compiler is  
 12879         still allowed to treat more than one bit pattern as a representation of the null pointer (all such  
 12880         patterns will compare equal to one another, and unequal to any pointer to any other object).  
 12881         Thus, applications should not assume that a pointer object with non-zero representation is not a  
 12882         null pointer.

12883 **FUTURE DIRECTIONS**

12884       None.

12885 **SEE ALSO**

12886       [<sys/types.h>](#), [<wchar.h>](#)

12887       XSH *confstr()*

12888       XCU *getconf*

12889 **CHANGE HISTORY**

12890       First released in Issue 4. Derived from the ANSI C standard.

12891 **Issue 7**

12892       This reference page is clarified with respect to macros and symbolic constants.

12893       SD5-XBD-ERN-53 is applied, updating the definition of **wchar\_t** to align with  
12894       ISO/IEC 9899:1999 standard, Technical Corrigendum 3.

12895 **Issue 8**

12896       Austin Group Defect 940 is applied, adding a requirement that any pointer object whose  
12897       representation has all bits set to zero is interpreted as a null pointer.

12898       Austin Group Defect 1302 is applied, adding **max\_align\_t**.



12899 **NAME**

12900           stdint.h — integer types

12901 **SYNOPSIS**

12902           #include <stdint.h>

12903 **DESCRIPTION**

12904 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 12905       Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 12906       enable the visibility of these symbols in this header.

12907       The <stdint.h> header shall declare sets of integer types having specified widths, and shall  
 12908       define corresponding sets of macros. It shall also define macros that specify limits of integer  
 12909       types corresponding to types defined in other standard headers.

12910       **Note:**     The “width” of an integer type is the number of bits used to store its value in a pure binary  
 12911       system; the actual type may use more bits than that (for example, a 28-bit type could be stored  
 12912       in 32 bits of actual storage). An  $N$ -bit signed type in two’s complement representation has  
 12913       values in the range  $-2^{N-1}$  to  $2^{N-1}-1$ , while an  $N$ -bit unsigned type has values in the range 0 to  
 12914        $2^N-1$ . While the ISO C standard also permits signed integers in sign-magnitude or one’s  
 12915       complement form, this standard requires an implementation to use two’s complement  
 12916       representation for the standard integer types.

12917       Types are defined in the following categories:

- 12918           • Integer types having certain exact widths
- 12919           • Integer types having at least certain specified widths
- 12920           • Fastest integer types having at least certain specified widths
- 12921           • Integer types wide enough to hold pointers to objects
- 12922           • Integer types having greatest width

12923       (Some of these types may denote the same type.)

12924       Corresponding macros specify limits of the declared types and construct suitable constants.

12925       For each type described herein that the implementation provides, the <stdint.h> header shall  
 12926       declare that **typedef** name and define the associated macros. Conversely, for each type described  
 12927       herein that the implementation does not provide, the <stdint.h> header shall not declare that  
 12928       **typedef** name, nor shall it define the associated macros. An implementation shall provide those  
 12929       types described as required, but need not provide any of the others (described as optional).

12930 **Integer Types**

12931       When **typedef** names differing only in the absence or presence of the initial  $u$  are defined, they  
 12932       shall denote corresponding signed and unsigned types as described in the ISO C standard,  
 12933       Section 6.2.5; an implementation providing one of these corresponding types shall also provide  
 12934       the other.

12935       In the following descriptions, the symbol  $N$  represents an unsigned decimal integer with no  
 12936       leading zeros (for example, 8 or 24, but not 04 or 048).

- 12937           • Exact-width integer types

12938           The **typedef** name **intN\_t** designates a signed integer type with width  $N$ , no padding bits,  
 12939           and a two’s-complement representation. Thus, **int8\_t** denotes such a signed integer type  
 12940           with a width of exactly 8 bits.

12941           The **typedef** name **uintN\_t** designates an unsigned integer type with width  $N$  and no

12942 padding bits. Thus, **uint24\_t** denotes such an unsigned integer type with a width of exactly  
12943 24 bits.

12944 CX The following types are required:  
12945 **int8\_t**  
12946 **int16\_t**  
12947 **int32\_t**  
12948 **uint8\_t**  
12949 **uint16\_t**  
12950 **uint32\_t**

12951 If an implementation provides integer types with width 64 that meet these requirements,  
12952 then the following types are required:

12953 **int64\_t**  
12954 **uint64\_t**

12955 CX In particular, this is the case if any of the following are true:  
12956 — The implementation supports the `_POSIX_V8_ILP32_OFFBIG` programming  
12957 environment and the application is being built in the `_POSIX_V8_ILP32_OFFBIG`  
12958 programming environment (see the Shell and Utilities volume of POSIX.1-2024, *c17*,  
12959 Programming Environments).  
12960 — The implementation supports the `_POSIX_V8_LP64_OFF64` programming  
12961 environment and the application is being built in the `_POSIX_V8_LP64_OFF64`  
12962 programming environment.  
12963 — The implementation supports the `_POSIX_V8_LPBIG_OFFBIG` programming  
12964 environment and the application is being built in the `_POSIX_V8_LPBIG_OFFBIG`  
12965 programming environment.

12966 If the representation of any of the standard types **short**, **int**, **long** or **long long** is not the  
12967 same as one of the above required types, an **intN\_t** type with that representation shall be  
12968 defined along with its **uintN\_t** counterpart.

12969 All other types of this form are optional.

12970 • Minimum-width integer types

12971 The **typedef** name **int\_leastN\_t** designates a signed integer type with a width of at least *N*,  
12972 such that no signed integer type with lesser size has at least the specified width. Thus,  
12973 **int\_least32\_t** denotes a signed integer type with a width of at least 32 bits.

12974 The **typedef** name **uint\_leastN\_t** designates an unsigned integer type with a width of at  
12975 least *N*, such that no unsigned integer type with lesser size has at least the specified width.  
12976 Thus, **uint\_least16\_t** denotes an unsigned integer type with a width of at least 16 bits.

12977 The following types are required:

12978 **int\_least8\_t**  
 12979 **int\_least16\_t**  
 12980 **int\_least32\_t**  
 12981 **int\_least64\_t**  
 12982 **uint\_least8\_t**  
 12983 **uint\_least16\_t**  
 12984 **uint\_least32\_t**  
 12985 **uint\_least64\_t**

12986 All other types of this form are optional.

- 12987 • Fastest minimum-width integer types

12988 Each of the following types designates an integer type that is usually fastest to operate  
 12989 with among all integer types that have at least the specified width.

12990 The designated type is not guaranteed to be fastest for all purposes; if the implementation  
 12991 has no clear grounds for choosing one type over another, it may simply pick some integer  
 12992 type satisfying the signedness and width requirements.

12993 The **typedef** name **int\_fastN\_t** designates the fastest signed integer type with a width of at  
 12994 least *N*. The **typedef** name **uint\_fastN\_t** designates the fastest unsigned integer type with  
 12995 a width of at least *N*.

12996 The following types are required:

12997 **int\_fast8\_t**  
 12998 **int\_fast16\_t**  
 12999 **int\_fast32\_t**  
 13000 **int\_fast64\_t**  
 13001 **uint\_fast8\_t**  
 13002 **uint\_fast16\_t**  
 13003 **uint\_fast32\_t**  
 13004 **uint\_fast64\_t**

13005 All other types of this form are optional.

- 13006 • Integer types capable of holding object pointers

13007 The following type designates a signed integer type with the property that any valid  
 13008 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and  
 13009 the result shall compare equal to the original pointer:

13010 **intptr\_t**

13011 The following type designates an unsigned integer type with the property that any valid  
 13012 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and  
 13013 the result shall compare equal to the original pointer:

13014 **uintptr\_t**

13015 XSI On XSI-conformant systems, the **intptr\_t** and **uintptr\_t** types are required; otherwise, they  
 13016 are optional.

- 13017 • Greatest-width integer types

13018 CX The following type designates a signed integer type using two's complement  
 13019 representation capable of representing any value of any signed integer type:

13020 **intmax\_t**

13021 The following type designates an unsigned integer type capable of representing any value  
13022 of any unsigned integer type:

13023 **uintmax\_t**

13024 These types are required.

13025 **Note:** Applications can test for optional types by using the corresponding limit macro from [Limits of](#)  
13026 [Specified-Width Integer Types](#).

13027 **Limits of Specified-Width Integer Types**

13028 The following macros specify the minimum and maximum limits of the types declared in the  
13029 <stdint.h> header. Each macro name corresponds to a similar type name in [Integer Types](#) (on  
13030 page 369).

13031 Each instance of any defined macro shall be replaced by a constant expression suitable for use in  
13032 #if preprocessing directives, and this expression shall have the same type as would an  
13033 expression that is an object of the corresponding type converted according to the integer  
13034 promotions. Its implementation-defined value shall be equal to or greater in magnitude  
13035 (absolute value) than the corresponding value given below, with the same sign, except where  
13036 stated to be exactly the given value.

13037 • Limits of exact-width integer types

13038 — Minimum values of exact-width signed integer types:

13039 {INTN\_MIN} Exactly  $-(2^{N-1})$

13040 — Maximum values of exact-width signed integer types:

13041 {INTN\_MAX} Exactly  $2^{N-1} - 1$

13042 — Maximum values of exact-width unsigned integer types:

13043 {UINTN\_MAX} Exactly  $2^N - 1$

13044 • Limits of minimum-width integer types

13045 — Minimum values of minimum-width signed integer types:

13046 CX {INT\_LEASTN\_MIN}  $-(2^{N-1})$

13047 — Maximum values of minimum-width signed integer types:

13048 {INT\_LEASTN\_MAX}  $2^{N-1} - 1$

13049 — Maximum values of minimum-width unsigned integer types:

13050 {UINT\_LEASTN\_MAX}  $2^N - 1$

13051 • Limits of fastest minimum-width integer types

13052 — Minimum values of fastest minimum-width signed integer types:

13053 CX {INT\_FASTN\_MIN}  $-(2^{N-1})$

13054 — Maximum values of fastest minimum-width signed integer types:

- 13055                    {INT\_FASTN\_MAX}         $2^{N-1} - 1$
- 13056                    — Maximum values of fastest minimum-width unsigned integer types:
- 13057                    {UINT\_FASTN\_MAX}         $2^N - 1$
- 13058                    • Limits of integer types capable of holding object pointers
- 13059                    — Minimum value of pointer-holding signed integer type:
- 13060 CX                    {INTPTR\_MIN}             $-(2^{15})$
- 13061                    — Maximum value of pointer-holding signed integer type:
- 13062                    {INTPTR\_MAX}             $2^{15} - 1$
- 13063                    — Maximum value of pointer-holding unsigned integer type:
- 13064                    {UINTPTR\_MAX}            $2^{16} - 1$
- 13065                    • Limits of greatest-width integer types
- 13066                    — Minimum value of greatest-width signed integer type:
- 13067 CX                    {INTMAX\_MIN}             $-(2^{63})$
- 13068                    — Maximum value of greatest-width signed integer type:
- 13069                    {INTMAX\_MAX}             $2^{63} - 1$
- 13070                    — Maximum value of greatest-width unsigned integer type:
- 13071                    {UINTMAX\_MAX}            $2^{64} - 1$

**Limits of Other Integer Types**

The following macros specify the minimum and maximum limits of integer types corresponding to types defined in other standard headers.

Each instance of these macros shall be replaced by a constant expression suitable for use in #if preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign.

- 13080                    • Limits of **ptrdiff\_t**:
- 13081 CX                    {PTRDIFF\_MIN}            $-65\,536$
- 13082                    {PTRDIFF\_MAX}            $+65\,535$
- 13083                    • Limits of **sig\_atomic\_t**:
- 13084                    {SIG\_ATOMIC\_MIN}        See below.
- 13085                    {SIG\_ATOMIC\_MAX}        See below.
- 13086                    • Limit of **size\_t**:
- 13087                    {SIZE\_MAX}                $65\,535$
- 13088                    • Limits of **wchar\_t**:
- 13089                    {WCHAR\_MIN}             See below.

13090 {WCHAR\_MAX} See below.

13091 • Limits of **wint\_t**:

13092 {WINT\_MIN} See below.

13093 {WINT\_MAX} See below.

13094 If **sig\_atomic\_t** (see the **<signal.h>** header) is defined as a signed integer type, the value of  
 13095 {SIG\_ATOMIC\_MIN} shall be no greater than  $-127$  and the value of {SIG\_ATOMIC\_MAX} shall  
 13096 be no less than  $127$ ; otherwise, **sig\_atomic\_t** shall be defined as an unsigned integer type, and  
 13097 the value of {SIG\_ATOMIC\_MIN} shall be  $0$  and the value of {SIG\_ATOMIC\_MAX} shall be no  
 13098 less than  $255$ .

13099 If **wchar\_t** (see the **<stddef.h>** header) is defined as a signed integer type, the value of  
 13100 {WCHAR\_MIN} shall be no greater than  $-127$  and the value of {WCHAR\_MAX} shall be no less  
 13101 than  $127$ ; otherwise, **wchar\_t** shall be defined as an unsigned integer type, and the value of  
 13102 {WCHAR\_MIN} shall be  $0$  and the value of {WCHAR\_MAX} shall be no less than  $255$ .

13103 If **wint\_t** (see the **<wchar.h>** header) is defined as a signed integer type, the value of  
 13104 {WINT\_MIN} shall be no greater than  $-32767$  and the value of {WINT\_MAX} shall be no less  
 13105 than  $32767$ ; otherwise, **wint\_t** shall be defined as an unsigned integer type, and the value of  
 13106 {WINT\_MIN} shall be  $0$  and the value of {WINT\_MAX} shall be no less than  $65535$ .

### 13107 **Macros for Integer Constant Expressions**

13108 The following macros expand to integer constant expressions suitable for initializing objects that  
 13109 have integer types corresponding to types defined in the **<stdint.h>** header. Each macro name  
 13110 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*  
 13111 *integer types*.

13112 Each invocation of one of these macros shall expand to an integer constant expression suitable  
 13113 for use in **#if** preprocessing directives. The type of the expression shall have the same type as  
 13114 would an expression that is an object of the corresponding type converted according to the  
 13115 integer promotions. The value of the expression shall be that of the argument.

13116 The argument in any instance of these macros shall be an unsuffixed integer constant with a  
 13117 value that does not exceed the limits for the corresponding type.

13118 • Macros for minimum-width integer constant expressions

13119 The macro **INTN\_C(value)** shall expand to an integer constant expression corresponding to  
 13120 the type **int\_leastN\_t**. The macro **UINTN\_C(value)** shall expand to an integer constant  
 13121 expression corresponding to the type **uint\_leastN\_t**. For example, if **uint\_least64\_t** is a  
 13122 name for the type **unsigned long long**, then **UINT64\_C(0x123)** might expand to the integer  
 13123 constant **0x123ULL**.

13124 • Macros for greatest-width integer constant expressions

13125 The following macro expands to an integer constant expression having the value specified  
 13126 by its argument and the type **intmax\_t**:

13127 **INTMAX\_C(value)**

13128 The following macro expands to an integer constant expression having the value specified  
 13129 by its argument and the type **uintmax\_t**:

13130 **UINTMAX\_C(value)**

13131 **APPLICATION USAGE**

13132 None.

13133 **RATIONALE**

13134 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in  
13135 freestanding environments, which might not support the formatted I/O functions. In some  
13136 environments, if the formatted conversion support is not wanted, using this header instead of  
13137 the <inttypes.h> header avoids defining such a large number of macros.

13138 As a consequence of adding **int8\_t**, the following are true:

- 13139 • A byte is exactly 8 bits.
- 13140 • {CHAR\_BIT} has the value 8, {SCHAR\_MAX} has the value 127, {SCHAR\_MIN} has the  
13141 value -128, and {UCHAR\_MAX} has the value 255.

13142 Since the POSIX.1 standard explicitly requires 8-bit **char** with two's complement arithmetic, it is  
13143 easier for application writers if the same two's complement guarantees are extended to all of the  
13144 other standard integer types. Furthermore, in programming environments with a 32-bit **long**,  
13145 some POSIX.1 interfaces, such as *mrnd48()*, cannot be implemented if **long** does not use a two's  
13146 complement representation.

13147 **FUTURE DIRECTIONS**

13148 **typedef** names beginning with **int** or **uint** and ending with **\_t** may be added to the types defined  
13149 in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **\_MAX**,  
13150 **\_MIN**, or **\_C** may be added to the macros defined in the <stdint.h> header.

13151 **SEE ALSO**

13152 &lt;inttypes.h&gt;, &lt;signal.h&gt;, &lt;stddef.h&gt;, &lt;wchar.h&gt;

13153 XSH Section 2.2 (on page 496)

13154 **CHANGE HISTORY**

13155 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

13156 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is applied.

13157 **Issue 7**

13158 ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #40 is applied.

13159 SD5-XBD-ERN-67 is applied.

13160 **Issue 8**13161 Austin Group Defect 1108 is applied, changing the maximum allowed value for all signed  
13162 integer minimum limits.

13163 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.

13164 Austin Group Defect 1330 is applied, changing ``\_V7\_'' to ``\_V8\_''.

13165 **NAME**

13166           stdio.h — standard buffered input/output

13167 **SYNOPSIS**

13168           #include &lt;stdio.h&gt;

13169 **DESCRIPTION**

13170 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 13171       Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 496) to  
 13172       enable the visibility of these symbols in this header.

13173       The **<stdio.h>** header shall define the following data types through **typedef**:

13174       **FILE**                A type containing information about a file. The **FILE** type may be an  
 13175       incomplete type.

13176       **fpos\_t**             A complete object type, other than an array type, capable of recording all  
 13177       the information needed to specify uniquely every position within a file.

13178       **off\_t**              As described in **<sys/types.h>**.

13179       **size\_t**             As described in **<stddef.h>**.

13180 CX       **ssize\_t**         As described in **<sys/types.h>**.

13181 CX       **va\_list**         As described in **<stdarg.h>**.

13182       The **<stdio.h>** header shall define the following macros which shall expand to integer constant  
 13183       expressions:

13184 CX       **BUFSIZ**         Size of **<stdio.h>** buffers. This shall expand to a positive value.

13185 CX       **L\_ctermid**       Maximum size of character array to hold *ctermid()* output.

13186 OB       **L\_tmpnam**        Maximum size of character array to hold *tmpnam()* output.

13187       The **<stdio.h>** header shall define the following macros which shall expand to integer constant  
 13188       expressions with distinct values:

13189       **\_IOFBF**            Input/output fully buffered.

13190       **\_IOLBF**            Input/output line buffered.

13191       **\_IONBF**            Input/output unbuffered.

13192       The **<stdio.h>** header shall define the following macros which shall expand to integer constant  
 13193       expressions with distinct values:

13194       **SEEK\_CUR**         Seek relative to current position.

13195       **SEEK\_END**         Seek relative to end-of-file.

13196       **SEEK\_SET**         Seek relative to start-of-file.

13197       The **<stdio.h>** header shall define the following macros which shall expand to integer constant  
 13198       expressions denoting implementation limits:

13199       **{FILENAME\_MAX}**    Maximum size in bytes of the longest pathname that the implementation  
 13200       guarantees can be opened.

13201       **{FOPEN\_MAX}**       Number of streams which the implementation guarantees can be open  
 13202       simultaneously. The value is at least eight.



13203 OB {TMP\_MAX} Minimum number of unique filenames generated by *tmpnam()*.  
 13204 Maximum number of times an application can call *tmpnam()* reliably. The  
 13205 value of {TMP\_MAX} is at least 25.

13206 OB XSI On XSI-conformant systems, the value of {TMP\_MAX} is at least 10 000.

13207 The <stdio.h> header shall define the following macro which shall expand to an integer constant  
 13208 expression with type **int** and a negative value:

13209 EOF End-of-file return value.

13210 The <stdio.h> header shall define NULL as described in <stddef.h>.

13211 The <stdio.h> header shall define the following macros which shall expand to expressions of  
 13212 type “pointer to FILE” that point to the FILE objects associated, respectively, with the standard  
 13213 error, input, and output streams:

13214 *stderr* Standard error output stream.  
 13215 *stdin* Standard input stream.  
 13216 *stdout* Standard output stream.

13217 The following shall be declared as functions and may also be defined as macros. Function  
 13218 prototypes shall be provided.

13219 void clearerr(FILE \*);  
 13220 CX char \*ctermid(char \*);  
 13221 int dprintf(int, const char \*restrict, ...)  
 13222 int fclose(FILE \*);  
 13223 CX FILE \*fdopen(int, const char \*);  
 13224 int feof(FILE \*);  
 13225 int ferror(FILE \*);  
 13226 int fflush(FILE \*);  
 13227 int fgetc(FILE \*);  
 13228 int fgetpos(FILE \*restrict, fpos\_t \*restrict);  
 13229 char \*fgets(char \*restrict, int, FILE \*restrict);  
 13230 CX int fileno(FILE \*);  
 13231 void flockfile(FILE \*);  
 13232 FILE \*fmemopen(void \*restrict, size\_t, const char \*restrict);  
 13233 FILE \*fopen(const char \*restrict, const char \*restrict);  
 13234 int fprintf(FILE \*restrict, const char \*restrict, ...);  
 13235 int fputc(int, FILE \*);  
 13236 int fputs(const char \*restrict, FILE \*restrict);  
 13237 size\_t fread(void \*restrict, size\_t, size\_t, FILE \*restrict);  
 13238 FILE \*freopen(const char \*restrict, const char \*restrict,  
 13239 FILE \*restrict);  
 13240 int fscanf(FILE \*restrict, const char \*restrict, ...);  
 13241 int fseek(FILE \*, long, int);  
 13242 CX int fseeko(FILE \*, off\_t, int);  
 13243 int fsetpos(FILE \*, const fpos\_t \*);  
 13244 long ftell(FILE \*);  
 13245 CX off\_t ftello(FILE \*);  
 13246 int ftrylockfile(FILE \*);  
 13247 void funlockfile(FILE \*);  
 13248 size\_t fwrite(const void \*restrict, size\_t, size\_t, FILE \*restrict);  
 13249 int getc(FILE \*);

```

13250     int      getchar(void);
13251 CX   int     getc_unlocked(FILE *);
13252     int      getchar_unlocked(void);
13253     ssize_t  getdelim(char **restrict, size_t *restrict, int,
13254                   FILE *restrict);
13255     ssize_t  getline(char **restrict, size_t *restrict, FILE *restrict);
13256     FILE *   *open_memstream(char **, size_t *);
13257     int      pclose(FILE *);
13258     void     perror(const char *);
13259 CX   FILE *   *popen(const char *, const char *);
13260     int      printf(const char *restrict, ...);
13261     int      putchar(int, FILE *);
13262     int      putchar(int);
13263 CX   int      putchar_unlocked(int, FILE *);
13264     int      putchar_unlocked(int);
13265     int      puts(const char *);
13266     int      remove(const char *);
13267     int      rename(const char *, const char *);
13268 CX   int      renameat(int, const char *, int, const char *);
13269     void     rewind(FILE *);
13270     int      scanf(const char *restrict, ...);
13271     void     setbuf(FILE *restrict, char *restrict);
13272     int      setvbuf(FILE *restrict, char *restrict, int, size_t);
13273     int      snprintf(char *restrict, size_t, const char *restrict, ...);
13274     int      sprintf(char *restrict, const char *restrict, ...);
13275     int      sscanf(const char *restrict, const char *restrict, ...);
13276     FILE *   *tmpfile(void);
13277 OB   char *   *tmpnam(char *);
13278     int      ungetc(int, FILE *);
13279 CX   int      vdprintf(int, const char *restrict, va_list);
13280     int      vfprintf(FILE *restrict, const char *restrict, va_list);
13281     int      vfscanf(FILE *restrict, const char *restrict, va_list);
13282     int      vprintf(const char *restrict, va_list);
13283     int      vscanf(const char *restrict, va_list);
13284     int      vsnprintf(char *restrict, size_t, const char *restrict,
13285                   va_list);
13286     int      vsprintf(char *restrict, const char *restrict, va_list);
13287     int      vsscanf(const char *restrict, const char *restrict, va_list);
13288 CX   Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.

```

**13289 APPLICATION USAGE**

13290 Since standard I/O streams may use an underlying file descriptor to access the file associated  
13291 with a stream, application developers need to be aware that {FOPEN\_MAX} streams may not be  
13292 available if file descriptors are being used to access files that are not associated with streams.

13293 Since the latest revision of the ISO C standard allows **FILE** to be an incomplete type (and POSIX  
13294 also allows it), portable applications can no longer allocate or copy an object of type **FILE**; only  
13295 pointers to objects of type **FILE** can be allocated.

13296 **RATIONALE**

13297 None.

13298 **FUTURE DIRECTIONS**

13299 None.

13300 **SEE ALSO**

13301 <stdarg.h>, <stddef.h>, <sys/types.h>

13302 XSH Section 2.2 (on page 496), *clearerr()*, *ctermid()*, *fclose()*, *fdopen()*, *feof()*, *ferror()*, *fflush()*,  
 13303 *fgetc()*, *fgetpos()*, *fgets()*, *fileno()*, *flockfile()*, *fmemopen()*, *fopen()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*,  
 13304 *freopen()*, *fscanf()*, *fseek()*, *fsetpos()*, *ftell()*, *fwrite()*, *getc()*, *getchar()*, *getc\_unlocked()*, *getdelim()*,  
 13305 *getopt()*, *open\_memstream()*, *pclose()*, *perror()*, *popen()*, *putc()*, *putchar()*, *puts()*, *remove()*,  
 13306 *rename()*, *rewind()*, *setbuf()*, *setvbuf()*, *stdin*, *system()*, *tmpfile()*, *tmpnam()*, *ungetc()*, *vfprintf()*,  
 13307 *vfscanf()*

13308 **CHANGE HISTORY**

13309 First released in Issue 1. Derived from Issue 1 of the SVID.

13310 **Issue 5**

13311 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

13312 Large File System extensions are added.

13313 The constant *L\_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as  
 13314 extensions and LEGACY.

13315 The *cuserid()* and *getopt()* functions are marked LEGACY.

13316 **Issue 6**

13317 The constant *L\_cuserid* and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed  
 13318 as they were previously marked LEGACY.

13319 The *cuserid()*, *getopt()*, and *getw()* functions are removed as they were previously marked  
 13320 LEGACY.

13321 Several functions are marked as part of the Thread-Safe Functions option.

13322 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the  
 13323 description of the **fpos\_t** type is now explicitly updated to exclude array types.

13324 Extensions beyond the ISO C standard are marked.

13325 **Issue 7**

13326 Austin Group Interpretation 1003.1-2001 #172 is applied, adding rationale about a conflict for the  
 13327 definition of {TMP\_MAX} with the ISO C standard.

13328 SD5-XBD-ERN-99 is applied, adding APPLICATION USAGE.

13329 The *dprintf()*, *fmemopen()*, *getdelim()*, *getline()*, *open\_memstream()*, and *vdprintf()* functions are  
 13330 added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

13331 The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 13332 Set Part 2.

13333 The *gets()*, *tmpnam()*, and *tempnam()* functions and the *L\_tmpnam* macro are marked  
 13334 obsolescent.

13335 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
 13336 for the **off\_t** type is added.

13337 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0065 [291,427] is applied.

13338 **Issue 8**

13339 Austin Group Defect 1054 is applied, allowing **FILE** to be an incomplete type.

13340 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.

13341 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

13342 **NAME**  
 13343        stdlib.h — standard library definitions

13344 **SYNOPSIS**  
 13345        #include <stdlib.h>

13346 **DESCRIPTION**  
 13347 CX       Some of the functionality described on this reference page extends the ISO C standard.  
 13348        Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 13349        enable the visibility of these symbols in this header.

13350        The <stdlib.h> header shall define the following macros which shall expand to integer constant  
 13351        expressions:

13352 CX       EXIT\_FAILURE   Unsuccessful termination for *exit()*; the value shall be between 1 and 125  
 13353                            inclusive.

13354        EXIT\_SUCCESS   Successful termination for *exit()*; the value shall be 0.

13355        {RAND\_MAX}    Maximum value returned by *rand()*; at least 32 767.

13356        The <stdlib.h> header shall define the following macro which shall expand to a positive integer  
 13357        expression with type **size\_t**:

13358        {MB\_CUR\_MAX} Maximum number of bytes in a character specified by the current locale  
 13359                            (category *LC\_CTYPE*).

13360 CX       In the POSIX locale the value of {MB\_CUR\_MAX} shall be 1.

13361        The <stdlib.h> header shall define NULL as described in <stddef.h>.

13362        The <stdlib.h> header shall define the following data types through **typedef**:

13363        **div\_t**            Structure type returned by the *div()* function.

13364        **ldiv\_t**          Structure type returned by the *ldiv()* function.

13365        **lldiv\_t**         Structure type returned by the *lldiv()* function.

13366        **size\_t**          As described in <stddef.h>.

13367        **wchar\_t**         As described in <stddef.h>.

13368 CX       In addition, the <stdlib.h> header shall define the following symbolic constants and macros as  
 13369        described in <sys/wait.h>:

13370        WCOREDUMP  
 13371        WEXITSTATUS  
 13372        WIFEXITED  
 13373        WIFSIGNALED  
 13374        WIFSTOPPED  
 13375        WNOHANG  
 13376        WSTOPSIG  
 13377        WTERMSIG  
 13378        WUNTRACED

13379 CX       The <stdlib.h> header shall define the following symbolic constants as described in <fcntl.h>:

13380        O\_APPEND  
 13381        O\_CLOEXEC  
 13382        O\_CLOFORK

```

13383 SIO  O_DSYNC
13384 XSI  O_NOCTTY
13385     O_RDWR
13386 SIO  O_RSYNC
13387 CX   O_SYNC

```

13388 The following shall be declared as functions and may also be defined as macros. Function  
13389 prototypes shall be provided.

```

13390     _Noreturn void  _Exit(int);
13391 XSI  long          a64l(const char *);
13392     _Noreturn void  abort(void);
13393     int             abs(int);
13394     void            *aligned_alloc(size_t, size_t);
13395     int             at_quick_exit(void (*)(void));
13396     int             atexit(void (*)(void));
13397     double          atof(const char *);
13398     int             atoi(const char *);
13399     long            atol(const char *);
13400     long long       atoll(const char *);
13401     void            *bsearch(const void *, const void *, size_t, size_t,
13402                             int (*)(const void *, const void *));
13403     void            *calloc(size_t, size_t);
13404     div_t           div(int, int);
13405 XSI  double        drand48(void);
13406     double          erand48(unsigned short [3]);
13407     _Noreturn void  exit(int);
13408     void            free(void *);
13409     char            *getenv(const char *);
13410 CX   int           getsubopt(char **restrict, char *const *restrict,
13411                             char **restrict);
13412 XSI  int           grantpt(int);
13413     char            *initstate(unsigned, char *, size_t);
13414     long            jrand48(unsigned short [3]);
13415     char            *l64a(long);
13416     long            labs(long);
13417 XSI  void          lcong48(unsigned short [7]);
13418     ldiv_t          ldiv(long, long);
13419     long long       llabs(long long);
13420     lldiv_t         lldiv(long long, long long);
13421 XSI  long          lrand48(void);
13422     void            *malloc(size_t);
13423     int             mblen(const char *, size_t);
13424     size_t          mbstowcs(wchar_t *restrict, const char *restrict,
13425                             size_t);
13426     int             mbtowc(wchar_t *restrict, const char *restrict, size_t);
13427 CX   char            *mkdtemp(char *);
13428     int             mkostemp(char *, int);
13429     int             mkstemp(char *);
13430 XSI  long          mrand48(void);
13431     long            nrand48(unsigned short [3]);

```

```

13432 ADV int posix_memalign(void **, size_t, size_t);
13433 XSI int posix_openpt(int);
13434 char *ptsname(int);
13435 int ptsname_r(int, char *, size_t);
13436 int putenv(char *);
13437 void qsort(void *, size_t, size_t, int (*)(const void *,
13438 const void *));
13439 _Noreturn void quick_exit(int);
13440 CX void qsort_r(void *, size_t, size_t, int (*)(const void *,
13441 const void *, void *), void *);
13442 int rand(void);
13443 XSI long random(void);
13444 void *realloc(void *, size_t);
13445 CX void *reallocarray(void *, size_t, size_t);
13446 char *realpath(const char *restrict, char *restrict);
13447 char *secure_getenv(const char *);
13448 XSI unsigned short *seed48(unsigned short [3]);
13449 CX int setenv(const char *, const char *, int);
13450 OB XSI void setkey(const char *);
13451 XSI char *setstate(char *);
13452 void srand(unsigned);
13453 XSI void srand48(long);
13454 void *srandom(unsigned);
13455 double strtod(const char *restrict, char **restrict);
13456 float strtof(const char *restrict, char **restrict);
13457 long strtol(const char *restrict, char **restrict, int);
13458 long double strtold(const char *restrict, char **restrict);
13459 long long strtoll(const char *restrict, char **restrict, int);
13460 unsigned long strtoul(const char *restrict, char **restrict, int);
13461 unsigned long long strtoull(const char *restrict, char **restrict, int);
13462 int system(const char *);
13464 XSI int unlockpt(int);
13465 CX int unsetenv(const char *);
13466 size_t wcstombs(char *restrict, const wchar_t *restrict,
13467 size_t);
13468 int wctomb(char *, wchar_t);

13469 CX Inclusion of the <stdlib.h> header may also make visible all symbols from <fcntl.h>, <limits.h>,
13470 <math.h>, <stddef.h>, and <sys/wait.h>.

```

**APPLICATION USAGE**

None.

**RATIONALE**

The ISO C standard requires that `exit(EXIT_FAILURE)` returns “unsuccessful termination status” to the host environment. In a POSIX host environment this means that the lower 8 bits of `EXIT_FAILURE` must have at least one bit set. The standard developers decided to further restrict the allowed values for the following reasons:

- Exit statuses of 126 and greater are ambiguous in certain circumstances because they have special meanings in the shell (see XCU Section 2.8.2 (on page 2499) and the EXIT STATUS section of *sh*).

- 13481 • The *xargs* utility quits when a command execution exits with status 255 (see XCU *xargs*).
- 13482 • Calling *exit()* with a value greater than 255 or less than 0 is something that only programs
- 13483 which are specifically designed to have their exit status obtained by *waitid()* should do
- 13484 (since it does not truncate the exit status to 8 bits). “Pure ISO C” programs that call
- 13485 *exit* (`EXIT_FAILURE`) do not meet this design criterion.

13486 The requirement that the value of `EXIT_SUCCESS` is 0 is not shaded CX because this matches the  
 13487 requirement in the ISO C standard that *exit* (`EXIT_SUCCESS`) returns “successful termination  
 13488 status” to the host environment (when the host environment is a POSIX implementation).

13489 **FUTURE DIRECTIONS**

13490 None.

13491 **SEE ALSO**

13492 <limits.h>, <math.h>, <stddef.h>, <sys/types.h>, <sys/wait.h>

13493 XSH Section 2.2 (on page 496), *\_Exit()*, *a64l()*, *abort()*, *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*,  
 13494 *bsearch()*, *calloc()*, *div()*, *drand48()*, *exit()*, *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *labs()*,  
 13495 *ldiv()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkdtemp()*, *posix\_memalign()*, *posix\_openpt()*,  
 13496 *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *setenv()*, *setkey()*, *strtod()*, *strtol()*,  
 13497 *strtol()*, *system()*, *unlockpt()*, *unsetenv()*, *waitid()*, *wcstombs()*, *wctomb()*

13498 **CHANGE HISTORY**

13499 First released in Issue 3.

13500 **Issue 5**

13501 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

13502 The *ttyslot()* and *valloc()* functions are marked LEGACY.

13503 The type of the third argument to *initstate()* is changed from **int** to **size\_t**. The type of the return  
 13504 value from *setstate()* is changed from **char** to **char \***, and the type of the first argument is  
 13505 changed from **char \*** to **const char \***.

13506 **Issue 6**

13507 The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be  
 13508 consistent with the reference page.

13509 The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be  
 13510 consistent with the reference page.

13511 The *rand\_r()* function is marked as part of the Thread-Safe Functions option.

13512 Function prototypes for *setenv()* and *unsetenv()* are added.

13513 The *posix\_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

13514 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

13515 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.

13516 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY.

13517 Extensions beyond the ISO C standard are marked.

13518 **Issue 7**

13519 SD5-XBD-ERN-79 and SD5-XBD-ERN-105 are applied.

13520 The LEGACY functions are removed.

13521 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API



- 13522 Set Part 1.
- 13523 The *rand\_r()* function is marked obsolescent.
- 13524 This reference page is clarified with respect to macros and symbolic constants.
- 13525 The type of the first argument to *setstate()* is changed from **const char \*** to **char \***.
- 13526 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0066 [197] is applied.
- 13527 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0075 [663] is applied.
- 13528 **Issue 8**
- 13529 Austin Group Defect 411 is applied, adding *mkostemp()*.
- 13530 Austin Group Defect 444 is applied, adding the **restrict** keyword to the *getsubopt()* prototype.
- 13531 Austin Group Defect 508 is applied, adding the *ptsname\_r()* function.
- 13532 Austin Group Defects 593 and 1350 are applied, adding some *O\_\** symbolic constants and  
13533 allowing **<stdlib.h>** to make visible all symbols from **<fcntl.h>**.
- 13534 Austin Group Defect 900 is applied, adding the *qsort\_r()* function.
- 13535 Austin Group Defect 922 is applied, adding the *secure\_getenv()* function.
- 13536 Austin Group Defect 1141 is applied, adding WCOREDUMP.
- 13537 Austin Group Defect 1192 is applied, marking the *setkey()* function as obsolescent.
- 13538 Austin Group Defect 1218 is applied, adding *reallocarray()*.
- 13539 Austin Group Defect 1229 is applied, changing the descriptions of EXIT\_FAILURE and  
13540 EXIT\_SUCCESS, and the RATIONALE section.
- 13541 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.
- 13542 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 13543 Austin Group Defect 1629 is applied, changing the RATIONALE section.
- 13544 Austin Group Defect 1663 is applied, removing XSI shading from *realpath()*.

13545 **NAME**

13546           stdnoreturn.h — noreturn macro

13547 **SYNOPSIS**

13548           #include &lt;stdnoreturn.h&gt;

13549 **DESCRIPTION**

13550 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
13551           conflict between the requirements described here and the ISO C standard is unintentional. This  
13552           volume of POSIX.1-2024 defers to the ISO C standard.

13553           The **<stdnoreturn.h>** header shall define the macro `noreturn` which shall expand to `_Noreturn`.13554 **APPLICATION USAGE**

13555           None.

13556 **RATIONALE**

13557           None.

13558 **FUTURE DIRECTIONS**

13559           None.

13560 **SEE ALSO**

13561           None.

13562 **CHANGE HISTORY**

13563           First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

13564 **NAME**

13565 string.h — string operations

13566 **SYNOPSIS**

13567 #include <string.h>

13568 **DESCRIPTION**

13569 CX Some of the functionality described on this reference page extends the ISO C standard.  
 13570 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 13571 enable the visibility of these symbols in this header.

13572 The <string.h> header shall define NULL and `size_t` as described in <stddef.h>.

13573 CX The <string.h> header shall define the `locale_t` type as described in <locale.h>.

13574 The following shall be declared as functions and may also be defined as macros. Function  
 13575 prototypes shall be provided for use with ISO C standard compilers.

```

13576 XSI void *memcpy(void *restrict, const void *restrict, int, size_t);
13577 void *memchr(const void *, int, size_t);
13578 int memcmp(const void *, const void *, size_t);
13579 void *memcpy(void *restrict, const void *restrict, size_t);
13580 CX void *memmem(const void *, size_t, const void *, size_t);
13581 void *memmove(void *, const void *, size_t);
13582 void *memset(void *, int, size_t);
13583 CX char *strcpy(char *restrict, const char *restrict);
13584 char *stpncpy(char *restrict, const char *restrict, size_t);
13585 char *strcat(char *restrict, const char *restrict);
13586 char *strchr(const char *, int);
13587 int strcmp(const char *, const char *);
13588 int strcoll(const char *, const char *);
13589 CX int strcoll_l(const char *, const char *, locale_t);
13590 char *strcpy(char *restrict, const char *restrict);
13591 size_t strcspn(const char *, const char *);
13592 CX char *strdup(const char *);
13593 char *strerror(int);
13594 CX char *strerror_l(int, locale_t);
13595 int strerror_r(int, char *, size_t);
13596 size_t strlcat(char *restrict, const char *restrict, size_t);
13597 size_t strlcpy(char *restrict, const char *restrict, size_t);
13598 size_t strlen(const char *);
13599 char *strncat(char *restrict, const char *restrict, size_t);
13600 int strncmp(const char *, const char *, size_t);
13601 char *strncpy(char *restrict, const char *restrict, size_t);
13602 CX char *strndup(const char *, size_t);
13603 size_t strnlen(const char *, size_t);
13604 char *strpbrk(const char *, const char *);
13605 char *strrchr(const char *, int);
13606 CX char *strsignal(int);
13607 size_t strspn(const char *, const char *);
13608 char *strstr(const char *, const char *);
13609 char *strtok(char *restrict, const char *restrict);
13610 CX char *strtok_r(char *restrict, const char *restrict, char **restrict);
13611 size_t strxfrm(char *restrict, const char *restrict, size_t);
    
```

```
13612 CX      size_t  strxfrm_l(char *restrict, const char *restrict,  
13613          size_t, locale_t);
```

13614 CX Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

#### 13615 APPLICATION USAGE

13616 None.

#### 13617 RATIONALE

13618 None.

#### 13619 FUTURE DIRECTIONS

13620 None.

#### 13621 SEE ALSO

13622 [<locale.h>](#), [<stddef.h>](#), [<sys/types.h>](#)

13623 XSH Section 2.2 (on page 496), *memcpy()*, *memchr()*, *memcmp()*, *memcpy()*, *memmem()*,  
13624 *memmove()*, *memset()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*, *strcspn()*, *strdup()*, *strerror()*,  
13625 *strlcat()*, *strlen()*, *strncat()*, *strncmp()*, *strncpy()*, *strpbrk()*, *strrchr()*, *strsignal()*, *strspn()*, *strstr()*,  
13626 *strtok()*, *strxfrm()*

#### 13627 CHANGE HISTORY

13628 First released in Issue 1. Derived from Issue 1 of the SVID.

##### 13629 Issue 5

13630 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

##### 13631 Issue 6

13632 The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.

13633 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

13634 The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

##### 13635 Issue 7

13636 SD5-XBD-ERN-15 is applied, correcting the prototype for the *strerror\_r()* function.

13637 The *stpncpy()*, *stpncpy()*, *strndup()*, *strlen()*, and *strsignal()* functions are added from The Open  
13638 Group Technical Standard, 2006, Extended API Set Part 1.

13639 The *strcoll\_l()*, *strerror\_l()*, and *strxfrm\_l()* functions are added from The Open Group Technical  
13640 Standard, 2006, Extended API Set Part 4.

13641 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
13642 for the **locale\_t** type is added.

##### 13643 Issue 8

13644 Austin Group Defect 986 is applied, adding *strlcat()* and *strlcpy()*.

13645 Austin Group Defect 1061 is applied, adding *memmem()*.

13646 **NAME**

13647 strings.h — string operations

13648 **SYNOPSIS**

13649 #include <strings.h>

13650 **DESCRIPTION**

13651 The following shall be declared as functions and may also be defined as macros. Function  
13652 prototypes shall be provided for use with ISO C standard compilers.

```
13653 XSI int ffs(int);
13654 int ffs1(long);
13655 int ffs11(long long);
13656 int strcasecmp(const char *, const char *);
13657 int strcasecmp_l(const char *, const char *, locale_t);
13658 int strncasecmp(const char *, const char *, size_t);
13659 int strncasecmp_l(const char *, const char *, size_t, locale_t);
```

13660 The <strings.h> header shall define the **locale\_t** type as described in <locale.h>.

13661 The <strings.h> header shall define the **size\_t** type as described in <sys/types.h>.

13662 **APPLICATION USAGE**

13663 None.

13664 **RATIONALE**

13665 None.

13666 **FUTURE DIRECTIONS**

13667 None.

13668 **SEE ALSO**

13669 <locale.h>, <sys/types.h>

13670 XSH *ffs()*, *strcasecmp()*

13671 **CHANGE HISTORY**

13672 First released in Issue 4, Version 2.

13673 **Issue 6**

13674 The Open Group Corrigendum U021/2 is applied, correcting the prototype for *index()* to be  
13675 consistent with the reference page.

13676 The *bcmp()*, *bcopy()*, *bzero()*, *index()*, and *rindex()* functions are marked LEGACY.

13677 **Issue 7**

13678 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size\_t** type.

13679 The LEGACY functions are removed.

13680 The <strings.h> header is moved from the XSI option to the Base.

13681 The *strcasecmp\_l()* and *strncasecmp\_l()* functions are added from The Open Group Technical  
13682 Standard, 2006, Extended API Set Part 4.

13683 A declaration for the **locale\_t** type is added.

13684 **Issue 8**

13685 Austin Group Defect 617 is applied, adding *ffsl()* and *ffsll()*.

13686 **NAME**13687 `sys/ipc.h` — XSI interprocess communication access structure13688 **SYNOPSIS**13689 XSI `#include <sys/ipc.h>`13690 **DESCRIPTION**

13691 The **<sys/ipc.h>** header is used by three mechanisms for XSI interprocess communication (IPC):  
13692 messages, semaphores, and shared memory. All use a common structure type, **ipc\_perm**, to pass  
13693 information used in determining permission to perform an IPC operation.

13694 The **<sys/ipc.h>** header shall define the **ipc\_perm** structure, which shall include the following  
13695 members:

13696	<code>uid_t</code>	<code>uid</code>	Owner's user ID.
13697	<code>gid_t</code>	<code>gid</code>	Owner's group ID.
13698	<code>uid_t</code>	<code>cuid</code>	Creator's user ID.
13699	<code>gid_t</code>	<code>cgid</code>	Creator's group ID.
13700	<code>mode_t</code>	<code>mode</code>	Read/write permission.

13701 The **<sys/ipc.h>** header shall define the **uid\_t**, **gid\_t**, **mode\_t**, and **key\_t** types as described in  
13702 **<sys/types.h>**.

13703 The **<sys/ipc.h>** header shall define the following symbolic constants.

13704 Mode bits:

13705	<code>IPC_CREAT</code>	Create entry if key does not exist.
13706	<code>IPC_EXCL</code>	Fail if key exists.
13707	<code>IPC_NOWAIT</code>	Error if request would need to wait.

13708 Keys:

13709	<code>IPC_PRIVATE</code>	Private key.
-------	--------------------------	--------------

13710 Control commands:

13711	<code>IPC_RMID</code>	Remove identifier.
13712	<code>IPC_SET</code>	Set options.
13713	<code>IPC_STAT</code>	Get options.

13714 The following shall be declared as a function and may also be defined as a macro. A function  
13715 prototype shall be provided.

13716 `key_t ftok(const char *, int);`

13717 **APPLICATION USAGE**

13718 None.

13719 **RATIONALE**

13720 None.

13721 **FUTURE DIRECTIONS**

13722 None.

13723 **SEE ALSO**13724        [<sys/types.h>](#)13725        XSH *ftok()*13726 **CHANGE HISTORY**

13727        First released in Issue 2. Derived from System V Release 2.0.

13728 **Issue 7**

13729        This reference page is clarified with respect to macros and symbolic constants.

13730 **Issue 8**

13731        The description of IPC\_NOWAIT is updated to eliminate the use of ``must''.

13732 **NAME**

13733 sys/mman.h — memory management declarations

13734 **SYNOPSIS**

13735 #include <sys/mman.h>

13736 **DESCRIPTION**

13737 The <sys/mman.h> header shall define the following symbolic constants for use as protection  
13738 options:

13739 PROT\_EXEC Page can be executed.

13740 PROT\_NONE Page cannot be accessed.

13741 PROT\_READ Page can be read.

13742 PROT\_WRITE Page can be written.

13743 The <sys/mman.h> header shall define the following symbolic constants for use as flag options:

13744 MAP\_ANON Synonym for MAP\_ANONYMOUS. MAP\_ANON shall have the same  
13745 value as MAP\_ANONYMOUS.

13746 MAP\_ANONYMOUS Map anonymous memory.

13747 MAP\_FIXED Interpret *addr* exactly.

13748 MAP\_PRIVATE Changes are private.

13749 MAP\_SHARED Share changes.

13750 XSI|SIO The <sys/mman.h> header shall define the following symbolic constants for the *msync()*  
13751 function:

13752 MS\_ASYNC Perform asynchronous writes.

13753 MS\_INVALIDATE Invalidate mappings.

13754 MS\_SYNC Perform synchronous writes.

13755 ML The <sys/mman.h> header shall define the following symbolic constants for the *mlockall()*  
13756 function:

13757 MCL\_CURRENT Lock currently mapped pages.

13758 MCL\_FUTURE Lock pages that become mapped.

13759 The <sys/mman.h> header shall define the symbolic constant MAP\_FAILED which shall have  
13760 type **void \*** and shall be used to indicate a failure from the *mmap()* function .

13761 ADV If the Advisory Information option is supported, the <sys/mman.h> header shall define  
13762 symbolic constants for the *advice* argument to the *posix\_madvise()* function as follows:

13763 POSIX\_MADV\_DONTNEED

13764 The application expects that it will not access the specified range in the near future.

13765 POSIX\_MADV\_NORMAL

13766 The application has no advice to give on its behavior with respect to the specified range. It  
13767 is the default characteristic if no advice is given for a range of memory.



13768 POSIX\_MADV\_RANDOM  
 13769 The application expects to access the specified range in a random order.

13770 POSIX\_MADV\_SEQUENTIAL  
 13771 The application expects to access the specified range sequentially from lower addresses to  
 13772 higher addresses.

13773 POSIX\_MADV\_WILLNEED  
 13774 The application expects to access the specified range in the near future.

13775 TYM The <sys/mman.h> header shall define the following symbolic constants for use as flags for the  
 13776 *posix\_typed\_mem\_open()* function:

13777 POSIX\_TYPED\_MEM\_ALLOCATE  
 13778 Allocate on *mmap()*.

13779 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG  
 13780 Allocate contiguously on *mmap()*.

13781 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE  
 13782 Map on *mmap()*, without affecting allocatability.

13783 The <sys/mman.h> header shall define the **mode\_t**, **off\_t**, and **size\_t** types as described in  
 13784 <sys/types.h>.

13785 TYM The <sys/mman.h> header shall define the **posix\_typed\_mem\_info** structure, which shall  
 13786 include at least the following member:

13787 `size_t posix_tmi_length` Maximum length which may be allocated  
 13788 from a typed memory object.

13789 The <sys/mman.h> header shall define the following symbolic constants as described in  
 13790 <fcntl.h>:

13791 SHM|TYM O\_RDONLY  
 13792 O\_RDWR  
 13793 TYM O\_WRONLY  
 13794 O\_CLOEXEC  
 13795 O\_CLOFORK  
 13796 SHM O\_CREAT  
 13797 O\_EXCL  
 13798 O\_TRUNC

13799 The following shall be declared as functions and may also be defined as macros. Function  
 13800 prototypes shall be provided.

13801 MLR `int mlock(const void *, size_t);`  
 13802 ML `int mlockall(int);`  
 13803 `void *mmap(void *, size_t, int, int, int, off_t);`  
 13804 `int mprotect(void *, size_t, int);`  
 13805 XSI|SIO `int msync(void *, size_t, int);`  
 13806 MLR `int munlock(const void *, size_t);`  
 13807 ML `int munlockall(void);`  
 13808 `int munmap(void *, size_t);`

```

13809 ADV      int      posix_madvise(void *, size_t, int);
13810 TYM      int      posix_mem_offset(const void *restrict, size_t, off_t *restrict,
13811           size_t *restrict, int *restrict);
13812           int      posix_typed_mem_get_info(int, struct posix_typed_mem_info *);
13813           int      posix_typed_mem_open(const char *, int, int);
13814 SHM      int      shm_open(const char *, int, mode_t);
13815           int      shm_unlink(const char *);

```

13816 Inclusion of the <sys/mman.h> header may make visible all symbols from the <fcntl.h> header.

### 13817 APPLICATION USAGE

13818 None.

### 13819 RATIONALE

13820 None.

### 13821 FUTURE DIRECTIONS

13822 None.

### 13823 SEE ALSO

13824 [<sys/types.h>](#)

13825 XSH [mlock\(\)](#), [mlockall\(\)](#), [mmap\(\)](#), [mprotect\(\)](#), [msync\(\)](#), [munmap\(\)](#), [posix\\_madvise\(\)](#),  
13826 [posix\\_mem\\_offset\(\)](#), [posix\\_typed\\_mem\\_get\\_info\(\)](#), [posix\\_typed\\_mem\\_open\(\)](#), [shm\\_open\(\)](#),  
13827 [shm\\_unlink\(\)](#)

### 13828 CHANGE HISTORY

13829 First released in Issue 4, Version 2.

#### 13830 Issue 5

13831 Updated for alignment with the POSIX Realtime Extension.

#### 13832 Issue 6

13833 The <sys/mman.h> header is marked as dependent on support for either the Memory Mapped  
13834 Files, Process Memory Locking, or Shared Memory Objects options.

13835 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 13836 • The TYM margin code is added to the list of margin codes for the <sys/mman.h> header  
13837 line, as well as for other lines.
- 13838 • The POSIX\_TYPED\_MEM\_ALLOCATE, POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG,  
13839 and POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flags are added.
- 13840 • The **posix\_tmi\_length** structure is added.
- 13841 • The *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, and *posix\_typed\_mem\_open()* functions  
13842 are added.

13843 The **restrict** keyword is added to the prototype for *posix\_mem\_offset()*.

13844 IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix\_madvise()*.

13845 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and  
13846 shading errors for the *mlock()* and *munlock()* functions.

13847 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code  
13848 for the *mmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).

13849 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code

- 13850 for the *munmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).
- 13851 **Issue 7**
- 13852 SD5-XBD-ERN-5 is applied, rewriting the DESCRIPTION.
- 13853 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
- 13854 the Base.
- 13855 This reference page is clarified with respect to macros and symbolic constants.
- 13856 **Issue 8**
- 13857 Austin Group Defect 593 is applied, adding O\_RDONLY, O\_RDWR, O\_WRONLY, O\_CLOEXEC,
- 13858 O\_CLOFORK, O\_CREAT, O\_EXCL, and O\_TRUNC, and allowing <sys/mman.h> to make
- 13859 visible all symbols from <fcntl.h>.
- 13860 Austin Group Defect 850 is applied, adding MAP\_ANON and MAP\_ANONYMOUS.

13861 **NAME**

13862 sys/msg.h — XSI message queue structures

13863 **SYNOPSIS**13864 XSI `#include <sys/msg.h>`13865 **DESCRIPTION**13866 The <sys/msg.h> header shall define the following data types through **typedef**:13867 **msgqnum\_t** Used for the number of messages in the message queue.13868 **msglen\_t** Used for the number of bytes allowed in a message queue.13869 These types shall be unsigned integer types that are able to store values at least as large as a type  
13870 **unsigned short**.13871 The <sys/msg.h> header shall define the following symbolic constant as a message operation  
13872 flag:13873 **MSG\_NOERROR** No error if big message.13874 The <sys/msg.h> header shall define the **msqid\_ds** structure, which shall include the following  
13875 members:

13876	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
13877	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
13878	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
13879	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <i>msgsnd()</i> .
13880	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <i>msgrcv()</i> .
13881	<code>time_t</code>	<code>msg_stime</code>	Time of last <i>msgsnd()</i> .
13882	<code>time_t</code>	<code>msg_rtime</code>	Time of last <i>msgrcv()</i> .
13883	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

13884 The <sys/msg.h> header shall define the **pid\_t**, **size\_t**, **ssize\_t**, and **time\_t** types as described in  
13885 <sys/types.h>.13886 The following shall be declared as functions and may also be defined as macros. Function  
13887 prototypes shall be provided.

```

13888 int      msgctl(int, int, struct msqid_ds *);
13889 int      msgget(key_t, int);
13890 ssize_t  msgrcv(int, void *, size_t, long, int);
13891 int      msgsnd(int, const void *, size_t, int);

```

13892 In addition, the &lt;sys/msg.h&gt; header shall include the &lt;sys/ipc.h&gt; header.

13893 **APPLICATION USAGE**

13894 None.

13895 **RATIONALE**

13896 None.

13897 **FUTURE DIRECTIONS**

13898 None.

13899 **SEE ALSO**13900 [<sys/ipc.h>](#), [<sys/types.h>](#)13901 XSH *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

13902 **CHANGE HISTORY**

13903 First released in Issue 2. Derived from System V Release 2.0.

13904 **Issue 7**

13905 Austin Group Interpretation 1003.1-2001 #179 is applied.

13906 This reference page is clarified with respect to macros and symbolic constants.

13907 **NAME**  
13908 sys/resource.h — definitions for resource operations

13909 **SYNOPSIS**  
13910 #include <sys/resource.h>

13911 **DESCRIPTION**

13912 XSI The <sys/resource.h> header shall define the following symbolic constants as possible values of  
13913 the *which* argument of *getpriority()* and *setpriority()*:  
13914 PRIO\_PROCESS Identifies the *who* argument as a process ID.  
13915 PRIO\_PGRP Identifies the *who* argument as a process group ID.  
13916 PRIO\_USER Identifies the *who* argument as a user ID.

13917 The <sys/resource.h> header shall define the following type through **typedef**:

13918 **rlim\_t** Unsigned integer type used for limit values.

13919 The <sys/resource.h> header shall define the following symbolic constants, which shall have  
13920 values suitable for use in **#if** preprocessing directives:

13921 RLIM\_INFINITY A value of **rlim\_t** indicating no limit.  
13922 RLIM\_SAVED\_MAX A value of type **rlim\_t** indicating an unrepresentable saved hard  
13923 limit.  
13924 RLIM\_SAVED\_CUR A value of type **rlim\_t** indicating an unrepresentable saved soft limit.

13925 On implementations where all resource limits are representable in an object of type **rlim\_t**,  
13926 RLIM\_SAVED\_MAX and RLIM\_SAVED\_CUR need not be distinct from RLIM\_INFINITY.

13927 XSI The <sys/resource.h> header shall define the following symbolic constants as possible values of  
13928 the *who* parameter of *getrusage()*:  
13929 RUSAGE\_SELF Returns information about the current process.  
13930 RUSAGE\_CHILDREN Returns information about children of the current process.

13931 The <sys/resource.h> header shall define the **rlimit** structure, which shall include at least the  
13932 following members:

13933 rlim\_t rlim\_cur The current (soft) limit.  
13934 rlim\_t rlim\_max The hard limit.

13935 XSI The <sys/resource.h> header shall define the **rusage** structure, which shall include at least the  
13936 following members:

13937 struct timeval ru\_utime User time used.  
13938 struct timeval ru\_stime System time used.

13939 The <sys/resource.h> header shall define the **timeval** structure as described in <sys/time.h>.

13940 The <sys/resource.h> header shall define the following symbolic constants as possible values for  
13941 the *resource* argument of *getrlimit()* and *setrlimit()*:

13942 RLIMIT\_CORE Limit on size of core image.  
13943 XSI RLIMIT\_CPU Limit on CPU time per process.

13944 RLIMIT\_DATA Limit on data segment size.

13945 RLIMIT\_FSIZE Limit on file size.

13946 RLIMIT\_NOFILE Limit on number of open files.

13947 RLIMIT\_STACK Limit on stack size.

13948 RLIMIT\_AS Limit on address space size.

13949 The following shall be declared as functions and may also be defined as macros. Function  
 13950 prototypes shall be provided.

13951 XSI `int getpriority(int, id_t);`

13952 `int getrlimit(int, struct rlimit *);`

13953 XSI `int getrusage(int, struct rusage *);`

13954 `int setpriority(int, id_t, int);`

13955 `int setrlimit(int, const struct rlimit *);`

13956 XSI The <sys/resource.h> header shall define the `id_t` type through `typedef`, as described in  
 13957 <sys/types.h>.

13958 Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.

13959 **APPLICATION USAGE**

13960 None.

13961 **RATIONALE**

13962 None.

13963 **FUTURE DIRECTIONS**

13964 None.

13965 **SEE ALSO**

13966 [<sys/time.h>](#), [<sys/types.h>](#)

13967 XSH `getpriority()`, `getrlimit()`, `getrusage()`

13968 **CHANGE HISTORY**

13969 First released in Issue 4, Version 2.

13970 **Issue 5**

13971 Large File System extensions are added.

13972 **Issue 7**

13973 This reference page is clarified with respect to macros and symbolic constants.

13974 **Issue 8**

13975 Austin Group Defects 51 and 1669 are applied, moving the `getrlimit()` and `setrlimit()` functions,  
 13976 excluding the RLIMIT\_CPU limit, from the XSI option to the Base.

13977 Austin Group Defect 1141 is applied, changing the description of RLIMIT\_CORE.

**13978 NAME**

13979 sys/select.h — select types

**13980 SYNOPSIS**

13981 #include <sys/select.h>

**13982 DESCRIPTION**

13983 The **<sys/select.h>** header shall define the **timeval** structure, which shall include at least the  
13984 following members:

13985 time\_t tv\_sec Seconds.

13986 suseconds\_t tv\_usec Microseconds.

13987 The **<sys/select.h>** header shall define the **time\_t** and **suseconds\_t** types as described in  
13988 **<sys/types.h>**.

13989 The **<sys/select.h>** header shall define the **sigset\_t** type as described in **<signal.h>**.

13990 The **<sys/select.h>** header shall define the **timespec** structure as described in **<time.h>**.

13991 The **<sys/select.h>** header shall define the **fd\_set** type as a structure.

13992 The **<sys/select.h>** header shall define the following symbolic constant, which shall have a value  
13993 suitable for use in **#if** preprocessing directives:

13994 FD\_SETSIZE Maximum number of file descriptors in an **fd\_set** structure.

13995 The following shall be declared as functions, defined as macros, or both. If functions are  
13996 declared, function prototypes shall be provided.

13997 void FD\_CLR(int, fd\_set \*);

13998 int FD\_ISSET(int, const fd\_set \*);

13999 void FD\_SET(int, fd\_set \*);

14000 void FD\_ZERO(fd\_set \*);

14001 If implemented as macros, these may evaluate their arguments more than once, so applications  
14002 should ensure that the arguments they supply are never expressions with side-effects.

14003 The following shall be declared as functions and may also be defined as macros. Function  
14004 prototypes shall be provided.

14005 int pselect(int, fd\_set \*restrict, fd\_set \*restrict, fd\_set \*restrict,  
14006 const struct timespec \*restrict, const sigset\_t \*restrict);

14007 int select(int, fd\_set \*restrict, fd\_set \*restrict, fd\_set \*restrict,  
14008 struct timeval \*restrict);

14009 Inclusion of the **<sys/select.h>** header may make visible all symbols from the headers  
14010 **<signal.h>** and **<time.h>**.

**14011 APPLICATION USAGE**

14012 None.

**14013 RATIONALE**

14014 None.

**14015 FUTURE DIRECTIONS**

14016 None.



14017 **SEE ALSO**

14018       &lt;signal.h&gt;, &lt;sys/time.h&gt;, &lt;sys/types.h&gt;, &lt;time.h&gt;

14019       XSH *pselect()*14020 **CHANGE HISTORY**

14021       First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

14022       The requirement for the **fd\_set** structure to have a member *fds\_bits* has been removed as per The  
14023       Open Group Base Resolution bwg2001-005.14024 **Issue 7**

14025       SD5-XBD-ERN-6 is applied, reordering the DESCRIPTION.

14026       This reference page is clarified with respect to macros and symbolic constants.

14027 **Issue 8**14028       Austin Group Defect 220 is applied, adding `const` to the second parameter of *FD\_ISSET()*.

14029 **NAME**

14030 sys/sem.h — XSI semaphore facility

14031 **SYNOPSIS**14032 XSI `#include <sys/sem.h>`14033 **DESCRIPTION**14034 The **<sys/sem.h>** header shall define the following symbolic constant for use as a semaphore  
14035 operation flag:

14036 SEM\_UNDO Set up adjust on exit entry.

14037 The **<sys/sem.h>** header shall define the following symbolic constants for use as commands for  
14038 the *semctl()* function:14039 GETNCNT Get *semncnt*.14040 GETPID Get *sempid*.14041 GETVAL Get *semval*.14042 GETALL Get all cases of *semval*.14043 GETZCNT Get *semzcnt*.14044 SETVAL Set *semval*.14045 SETALL Set all cases of *semval*.14046 The **<sys/sem.h>** header shall define the **semid\_ds** structure, which shall include the following  
14047 members:

14048 struct ipc\_perm sem\_perm Operation permission structure.

14049 unsigned short sem\_nsems Number of semaphores in set.

14050 time\_t sem\_otime Last *semop()* time.14051 time\_t sem\_ctime Last time changed by *semctl()*.14052 The **<sys/sem.h>** header shall define the **pid\_t**, **size\_t**, and **time\_t** types as described in  
14053 **<sys/types.h>**.14054 A semaphore shall be represented by an anonymous structure, which shall include the following  
14055 members:

14056 unsigned short semval Semaphore value.

14057 pid\_t sempid Process ID of last operation.

14058 unsigned short semncnt Number of processes waiting for *semval*  
14059 to become greater than current value.14060 unsigned short semzcnt Number of processes waiting for *semval*  
14061 to become 0.14062 The **<sys/sem.h>** header shall define the **sembuf** structure, which shall include the following  
14063 members:

14064 unsigned short sem\_num Semaphore number.

14065 short sem\_op Semaphore operation.

14066 short sem\_flg Operation flags.

14067 The following shall be declared as functions and may also be defined as macros. Function  
14068 prototypes shall be provided.

14069 int semctl(int, int, int, ...);

```
14070     int    semget(key_t, int, int);
14071     int    semop(int, struct sembuf *, size_t);
```

14072 In addition, the <sys/sem.h> header shall include the <sys/ipc.h> header.

#### 14073 **APPLICATION USAGE**

14074 None.

#### 14075 **RATIONALE**

14076 None.

#### 14077 **FUTURE DIRECTIONS**

14078 None.

#### 14079 **SEE ALSO**

14080 [<sys/ipc.h>](#), [<sys/types.h>](#)

14081 XSH [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#)

#### 14082 **CHANGE HISTORY**

14083 First released in Issue 2. Derived from System V Release 2.0.

#### 14084 **Issue 7**

14085 Austin Group Interpretation 1003.1-2001 #179 is applied.

14086 This reference page is clarified with respect to macros and symbolic constants.

14087 **NAME**

14088 sys/shm.h — XSI shared memory facility

14089 **SYNOPSIS**

14090 XSI #include <sys/shm.h>

14091 **DESCRIPTION**

14092 The <sys/shm.h> header shall define the following symbolic constants:

14093 SHM\_RDONLY Attach read-only (else read-write).

14094 SHM\_RND Round attach address to SHMLBA.

14095 SHMLBA Segment low boundary address multiple.

14096 The <sys/shm.h> header shall define the symbolic constant SHM\_FAILED which shall evaluate  
14097 to the same value as ((void\*)(intptr\_t)-1).

14098 The <sys/shm.h> header shall define the type **intptr\_t** as described in <stdint.h>.

14099 The <sys/shm.h> header shall define the following data type through **typedef**:

14100 **shmatt\_t** Unsigned integer used for the number of current attaches that shall be able to  
14101 store values at least as large as a type **unsigned short**.

14102 The <sys/shm.h> header shall define the **shmids** structure, which shall include the following  
14103 members:

14104	struct ipc_perm	shm_perm	Operation permission structure.
14105	size_t	shm_segsz	Size of segment in bytes.
14106	pid_t	shm_lpid	Process ID of last shared memory operation.
14107	pid_t	shm_cpid	Process ID of creator.
14108	shmatt_t	shm_nattch	Number of current attaches.
14109	time_t	shm_atime	Time of last <i>shmat</i> ().
14110	time_t	shm_dtime	Time of last <i>shmdt</i> ().
14111	time_t	shm_ctime	Time of last change by <i>shmctl</i> ().

14112 The <sys/shm.h> header shall define the **pid\_t**, **size\_t**, and **time\_t** types as described in  
14113 <sys/types.h>.

14114 The following shall be declared as functions and may also be defined as macros. Function  
14115 prototypes shall be provided.

```
14116 void *shmat(int, const void *, int);  
14117 int shmctl(int, int, struct shmids *);  
14118 int shmdt(const void *);  
14119 int shmget(key_t, size_t, int);
```

14120 In addition, the <sys/shm.h> header shall include the <sys/ipc.h> header.

14121 **APPLICATION USAGE**

14122 None.

14123 **RATIONALE**

14124 None.

14125 **FUTURE DIRECTIONS**

14126 None.

14127 **SEE ALSO**14128 [<sys/ipc.h>](#), [<sys/types.h>](#)14129 XSH [shmat\(\)](#), [shmctl\(\)](#), [shmdt\(\)](#), [shmget\(\)](#)14130 **CHANGE HISTORY**

14131 First released in Issue 2. Derived from System V Release 2.0.

14132 **Issue 5**14133 The type of *shm\_segsz* is changed from **int** to **size\_t**.14134 **Issue 7**

14135 Austin Group Interpretation 1003.1-2001 #179 is applied.

14136 This reference page is clarified with respect to macros and symbolic constants.

14137 **Issue 8**14138 Austin Group Defect 1239 is applied, adding SHM\_FAILED and requiring [<sys/shm.h>](#) to  
14139 define **intptr\_t**.

14140 **NAME**

14141 sys/socket.h — main sockets header

14142 **SYNOPSIS**

14143 #include &lt;sys/socket.h&gt;

14144 **DESCRIPTION**14145 The **<sys/socket.h>** header shall define the **socklen\_t** type, which is an integer type of width of  
14146 at least 32 bits; see APPLICATION USAGE.14147 The **<sys/socket.h>** header shall define the **sa\_family\_t** unsigned integer type.14148 The **<sys/socket.h>** header shall define the **sockaddr** structure, which shall include at least the  
14149 following members:14150 sa\_family\_t sa\_family Address family.  
14151 char sa\_data[] Socket address (variable-length data).14152 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,  
14153 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.14154 The **<sys/socket.h>** header shall define the **sockaddr\_storage** structure, which shall be:

- 14155 • Large enough to accommodate all supported protocol-specific address structures
- 
- 14156 • Aligned at an appropriate boundary so that pointers to it can be cast as pointers to
- 
- 14157 protocol-specific address structures and used to access the fields of those structures
- 
- 14158 without alignment problems

14159 The **sockaddr\_storage** structure shall include at least the following members:

14160 sa\_family\_t ss\_family

14161 When a pointer to a **sockaddr\_storage** structure is converted to a pointer to a **sockaddr**  
14162 structure, or vice versa, the *ss\_family* member of the **sockaddr\_storage** structure shall map onto  
14163 the *sa\_family* member of the **sockaddr** structure. When a pointer to a **sockaddr\_storage** structure  
14164 is converted to a pointer to a protocol-specific address structure, or vice versa, the *ss\_family*  
14165 member shall map onto a member of that structure that is of type **sa\_family\_t** that identifies the  
14166 protocol's address family. When a pointer to a **sockaddr** structure is converted to a pointer to a  
14167 protocol-specific address structure, or vice versa, the *sa\_family* member shall map onto a member  
14168 of that structure that is of type **sa\_family\_t** that identifies the protocol's address family.  
14169 Additionally, the structures shall be defined in such a way that the compiler treats an access to  
14170 the stored value of the **sa\_family\_t** member of any of these structures, via an lvalue expression  
14171 whose type involves any other one of these structures, as permissible, despite the more  
14172 restrictive expression rules on stored value access as stated in the ISO C standard. Similarly,  
14173 when a pointer to a **sockaddr\_storage** or **sockaddr** structure is converted to a pointer to a  
14174 protocol-specific address structure, the compiler shall treat an access (using this converted  
14175 pointer) to the stored value of any member of the protocol-specific structure as permissible. The  
14176 application shall ensure that the protocol-specific address structure corresponds to the family  
14177 indicated by the member with type **sa\_family\_t** of that structure and the pointed-to object has  
14178 sufficient memory for addressing all members of the protocol-specific structure.14179 The **<sys/socket.h>** header shall define the **msghdr** structure, which shall include at least the  
14180 following members:14181 void \*msg\_name Optional address.  
14182 socklen\_t msg\_namelen Size of address.  
14183 struct iovec \*msg\_iov Scatter/gather array.  
14184 int msg\_iovlen Members in *msg\_iov*.

14185 void \*msg\_control Ancillary data; see below.  
 14186 socklen\_t msg\_controllen Ancillary data buffer *len*.  
 14187 int msg\_flags Flags on received message.

14188 The **msghdr** structure is used to reduce the number of directly supplied parameters to the  
 14189 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the  
 14190 *recvmsg()* function and *value* only for the *sendmsg()* function.

14191 The <sys/socket.h> header shall define the **iovec** structure as described in <sys/uio.h>.

14192 The <sys/socket.h> header shall define the **cmsghdr** structure, which shall include at least the  
 14193 following members:

14194 socklen\_t cmsg\_len Data byte count, including the **cmsghdr**.  
 14195 int cmsg\_level Originating protocol.  
 14196 int cmsg\_type Protocol-specific type.

14197 The **cmsghdr** structure is used for storage of ancillary data object information.

14198 Ancillary data consists of a sequence of pairs, each consisting of a **cmsghdr** structure followed  
 14199 by a data array. The data array contains the ancillary data message, and the **cmsghdr** structure  
 14200 contains descriptive information that allows an application to correctly parse the data.

14201 The values for *cmsg\_level* shall be legal values for the *level* argument to the *getsockopt()* and  
 14202 *setsockopt()* functions. The system documentation shall specify the *cmsg\_type* definitions for the  
 14203 supported protocols.

14204 Ancillary data is also possible at the socket level. The <sys/socket.h> header shall define the  
 14205 following symbolic constant for use as the *cmsg\_type* value when *cmsg\_level* is SOL\_SOCKET:

14206 SCM\_RIGHTS Indicates that the data array contains the access rights to be sent or  
 14207 received.

14208 The <sys/socket.h> header shall define the following macros to gain access to the data arrays in  
 14209 the ancillary data associated with a message header:

14210 CMSG\_DATA(*cmsg*)

14211 If the argument is a pointer to a **cmsghdr** structure, this macro shall return an unsigned  
 14212 character pointer to the data array associated with the **cmsghdr** structure.

14213 CMSG\_NXTHDR(*mhdr,cmsg*)

14214 If the first argument is a pointer to a **msghdr** structure and the second argument is a pointer  
 14215 to a **cmsghdr** structure in the ancillary data pointed to by the *msg\_control* field of that  
 14216 **msghdr** structure, this macro shall return a pointer to the next **cmsghdr** structure, or a null  
 14217 pointer if the second argument points to the last **cmsghdr** and data array pair in the  
 14218 ancillary data. If the ancillary data contains another **cmsghdr** structure after this one but the  
 14219 *cmsg\_len* value in that structure is such that the data array following that structure would  
 14220 extend beyond the end of the ancillary data, it is unspecified whether this macro returns a  
 14221 pointer to that **cmsghdr** structure or returns a null pointer.

14222 If the first argument is a pointer to a **msghdr** structure and the second argument is a null  
 14223 pointer, this macro shall be equivalent to CMSG\_FIRSTHDR(*mhdr*).

14224 CMSG\_FIRSTHDR(*mhdr*)

14225 If the argument is a pointer to a **msghdr** structure, this macro shall return a pointer to the  
 14226 first **cmsghdr** structure in the ancillary data associated with this **msghdr** structure, or a null  
 14227 pointer if either there is no ancillary data associated with the **msghdr** structure  
 14228 (*msg\_controllen* is zero) or there is insufficient room in the ancillary data for a complete  
 14229 **cmsghdr** structure (*msg\_controllen* is non-zero but less than `sizeof(struct cmsghdr)`).

14230 CMSG\_SPACE(*length*)  
 14231 If the argument has a type such that its value can be assigned to an object of type **socklen\_t**,  
 14232 this macro shall return the space required by an ancillary data object of the specified length  
 14233 and its **cmsghdr** structure, including any padding needed to satisfy alignment  
 14234 requirements. This macro can be used, for example, to allocate space dynamically for the  
 14235 ancillary data. This macro should not be used to initialize the *msg\_len* member of a  
 14236 **cmsghdr** structure. If the argument is an integer constant expression, this macro shall  
 14237 expand to an integer constant expression.

14238 CMSG\_LEN(*length*)  
 14239 If the argument has a type such that its value can be assigned to an object of type **socklen\_t**,  
 14240 this macro shall return the value to store in the *msg\_len* member of the **cmsghdr** structure  
 14241 for an ancillary data object of the specified length, taking into account any padding needed  
 14242 to satisfy alignment requirements. If the argument is an integer constant expression, this  
 14243 macro shall expand to an integer constant expression.

14244 The <sys/socket.h> header shall define the **linger** structure, which shall include at least the  
 14245 following members:

14246 int l\_onoff Indicates whether linger option is enabled.  
 14247 int l\_linger Linger time, in seconds.

14248 The <sys/socket.h> header shall define the following socket types (see XSH Section 2.10.6, on  
 14249 page 550) as symbolic constants with distinct values:

14250 SOCK\_DGRAM Datagram socket.

14251 RS SOCK\_RAW Raw Protocol Interface.

14252 SOCK\_SEQPACKET Sequenced-packet socket.

14253 SOCK\_STREAM Byte-stream socket.

14254 Implementations may provide additional socket types.

14255 The header shall define the following socket creation flags, for use in *socket()*, *socketpair()*, and  
 14256 *accept4()*. These flags shall be symbolic constants with values that are bitwise distinct from each  
 14257 other and from all SOCK\_\* constants representing socket types:

14258 SOCK\_NONBLOCK Create a socket file descriptor with the O\_NONBLOCK flag atomically set  
 14259 on the new open file description.

14260 SOCK\_CLOEXEC Create a socket file descriptor with the FD\_CLOEXEC flag atomically set  
 14261 on that file descriptor.

14262 SOCK\_CLOFORK Create a socket file descriptor with the FD\_CLOFORK flag atomically set  
 14263 on that file descriptor.

14264 Implementations may provide additional socket creation flags.

14265 The <sys/socket.h> header shall define the following symbolic constant for use as the *level*  
 14266 argument of *setsockopt()* and *getsockopt()*.

14267 SOL\_SOCKET Options to be accessed at socket level, not protocol level.

14268 The <sys/socket.h> header shall define the following symbolic constants with distinct values for  
 14269 use as the *option\_name* argument in *getsockopt()* or *setsockopt()* calls (see XSH Section 2.10.16, on  
 14270 page 554):



14271	SO_ACCEPTCONN	Socket is accepting connections.
14272	SO_BROADCAST	Transmission of broadcast messages is supported.
14273	SO_DEBUG	Debugging information is being recorded.
14274	SO_DOMAIN	Socket domain.
14275	SO_DONTROUTE	Bypass normal routing.
14276	SO_ERROR	Socket error status.
14277	SO_KEEPALIVE	Connections are kept alive with periodic messages.
14278	SO_LINGER	Socket lingers on close.
14279	SO_OOINLINE	Out-of-band data is transmitted in line.
14280	SO_PROTOCOL	Socket protocol.
14281	SO_RCVBUF	Receive buffer size.
14282	SO_RCVLOWAT	Receive ``low water mark``.
14283	SO_RCVTIMEO	Receive timeout.
14284	SO_REUSEADDR	Reuse of local addresses is supported.
14285	SO_SNDBUF	Send buffer size.
14286	SO_SNDLOWAT	Send ``low water mark``.
14287	SO_SNDTIMEO	Send timeout.
14288	SO_TYPE	Socket type.
14289	The <sys/socket.h> header shall define the following symbolic constant for use as the maximum	
14290	<i>backlog</i> queue length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
14291	SOMAXCONN	The maximum <i>backlog</i> queue length.
14292	The <sys/socket.h> header shall define the following symbolic constants with distinct values for	
14293	use as the valid values for the <i>msg_flags</i> field in the <b>msghdr</b> structure, or the <i>flags</i> parameter in	
14294	<i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
14295	MSG_CMSG_CLOEXEC	
14296		Atomically set the FD_CLOEXEC flag on any file descriptors created via
14297		SCM_RIGHTS during <i>recvmsg()</i> .
14298	MSG_CMSG_CLOFORK	
14299		Atomically set the FD_CLOFORK flag on any file descriptors created via
14300		SCM_RIGHTS during <i>recvmsg()</i> .
14301	MSG_CTRUNC	Control data truncated.
14302	MSG_DONTROUTE	Send without using routing tables.
14303	MSG_EOR	Terminates a record (if supported by the protocol).
14304	MSG_OOB	Out-of-band data.
14305	MSG_NOSIGNAL	No SIGPIPE generated when an attempt to send is made on a stream-
14306		oriented socket that is no longer connected.

14307 MSG\_PEEK Leave received data in queue.

14308 MSG\_TRUNC Normal data truncated.

14309 MSG\_WAITALL Attempt to fill the read buffer.

14310 The **<sys/socket.h>** header shall define the following symbolic constants with distinct values:

14311 AF\_INET Internet domain sockets for use with IPv4 addresses.

14312 IP6 AF\_INET6 Internet domain sockets for use with IPv6 addresses.

14313 AF\_UNIX UNIX domain sockets.

14314 AF\_UNSPEC Unspecified.

14315 The value of AF\_UNSPEC shall be 0.

14316 The **<sys/socket.h>** header shall define the following symbolic constants with distinct values:

14317 SHUT\_RD Disables further receive operations.

14318 SHUT\_RDWR Disables further send and receive operations.

14319 SHUT\_WR Disables further send operations.

14320 The **<sys/socket.h>** header shall define the **size\_t** and **ssize\_t** types as described in

14321 **<sys/types.h>**.

14322 The following shall be declared as functions and may also be defined as macros. Function

14323 prototypes shall be provided.

```

14324 int accept(int, struct sockaddr *restrict, socklen_t *restrict);
14325 int accept4(int, struct sockaddr *restrict, socklen_t *restrict,
14326 int);
14327 int bind(int, const struct sockaddr *, socklen_t);
14328 int connect(int, const struct sockaddr *, socklen_t);
14329 int getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
14330 int getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
14331 int getsockopt(int, int, int, void *restrict, socklen_t *restrict);
14332 int listen(int, int);
14333 ssize_t recv(int, void *, size_t, int);
14334 ssize_t recvfrom(int, void *restrict, size_t, int,
14335 struct sockaddr *restrict, socklen_t *restrict);
14336 ssize_t recvmsg(int, struct msghdr *, int);
14337 ssize_t send(int, const void *, size_t, int);
14338 ssize_t sendmsg(int, const struct msghdr *, int);
14339 ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
14340 socklen_t);
14341 int setsockopt(int, int, int, const void *, socklen_t);
14342 int shutdown(int, int);
14343 int socketatmark(int);
14344 int socket(int, int, int);
14345 int socketpair(int, int, int, int [2]);

```

14346 Inclusion of **<sys/socket.h>** may also make visible all symbols from **<sys/uio.h>**.

## 14347 APPLICATION USAGE

14348 To forestall portability problems, it is recommended that applications not use values larger than  
14349  $2^{31} - 1$  for the `socklen_t` type.

14350 The `sockaddr_storage` structure solves the problem of declaring storage for automatic variables  
14351 which is both large enough and aligned enough for storing the socket address data structure of  
14352 any family. For example, code with a file descriptor and without the context of the address  
14353 family can pass a pointer to a variable of this type, where a pointer to a socket address structure  
14354 is expected in calls such as `getpeername()`, and determine the address family by accessing the  
14355 received content after the call.

14356 The example below illustrates a data structure which aligns on a 64-bit boundary. An  
14357 implementation-defined field `_ss_align` following `_ss_pad1` is used to force a 64-bit alignment  
14358 which covers proper alignment good enough for needs of at least `sockaddr_in6` (IPv6) and  
14359 `sockaddr_in` (IPv4) address data structures. The size of padding field `_ss_pad1` depends on the  
14360 chosen alignment boundary. The size of padding field `_ss_pad2` depends on the value of overall  
14361 size chosen for the total size of the structure. This size and alignment are represented in the  
14362 above example by implementation-defined (not required) constants `_SS_MAXSIZE` (chosen  
14363 value 128) and `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived  
14364 value 6) and `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The  
14365 implementation-defined definitions and structure field names above start with an <underscore>  
14366 to denote implementation private name space. Portable code is not expected to access or  
14367 reference those fields or constants. Note that this example only deals with size and alignment;  
14368 see RATIONALE for additional issues related to these structures.

```
14369 /*
14370  * Desired design of maximum size and alignment.
14371  */
14372 #define _SS_MAXSIZE 128
14373     /* Implementation-defined maximum size. */
14374 #define _SS_ALIGNSIZE (sizeof(int64_t))
14375     /* Implementation-defined desired alignment. */
14376 /*
14377  * Definitions used for sockaddr_storage structure paddings design.
14378  */
14379 #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
14380 #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t) + \
14381     _SS_PAD1SIZE + _SS_ALIGNSIZE))
14382 struct sockaddr_storage {
14383     sa_family_t ss_family; /* Address family. */
14384 /*
14385  * Following fields are implementation-defined.
14386  */
14387     char _ss_pad1[_SS_PAD1SIZE];
14388     /* 6-byte pad; this is to make implementation-defined
14389     pad up to alignment field that follows explicit in
14390     the data structure. */
14391     int64_t _ss_align; /* Field to force desired structure
14392     storage alignment. */
14393     char _ss_pad2[_SS_PAD2SIZE];
14394     /* 112-byte pad to achieve desired size,
14395     _SS_MAXSIZE value minus size of ss_family
14396     _ss_pad1, _ss_align fields is 112. */
```

14397 } ;

14398 Portable applications need to account for the alternative behaviors of the CMSG\_NXTHDR  
14399 macro as follows:

- 14400 • When constructing ancillary data in a **msghdr** structure, ensure that all locations within  
14401 the ancillary data that might be returned by CMSG\_NXTHDR contain a *cmsg\_len* value of  
14402 zero (typically this is achieved by using *memset()* to initialize the entire *msg\_control* buffer  
14403 to null bytes before populating the first **cmsghdr** structure).
- 14404 • When extracting ancillary data from a received **msghdr** structure, check that the data array  
14405 following the last **cmsghdr** structure does not extend beyond the end of the ancillary data.

#### 14406 RATIONALE

14407 Note that defining the **sockaddr\_storage** and **sockaddr** structures using only mechanisms  
14408 defined in early editions of the ISO C standard may produce aliasing diagnostics when  
14409 applications use casting between pointers to the various socket address structures. Because of  
14410 the large body of existing code utilizing sockets in a way that could trigger undefined behavior  
14411 due to strict aliasing rules, this standard mandates that these structures can alias each other for  
14412 accessing the **sa\_family\_t** member of the structures (or other members for protocol-specific  
14413 structure references), so as to preserve well-defined semantics. An implementation's header files  
14414 may need to use anonymous unions, or even an implementation-specific extension, to comply  
14415 with the requirements of this standard.

#### 14416 FUTURE DIRECTIONS

14417 None.

#### 14418 SEE ALSO

14419 [<sys/types.h>](#), [<sys/uio.h>](#)

14420 XSH *accept()*, *bind()*, *connect()*, *getpeername()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*,  
14421 *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *setsockopt()*, *shutdown()*, *socketmark()*, *socket()*,  
14422 *socketpair()*

#### 14423 CHANGE HISTORY

14424 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

14425 The **restrict** keyword is added to the prototypes for *accept()*, *getpeername()*, *getsockname()*,  
14426 *getsockopt()*, and *recvfrom()*.

#### 14427 Issue 7

14428 SD5-XBD-ERN-56 is applied, adding a reference to [<sys/types.h>](#) for the **ssize\_t** type.

14429 SD5-XBD-ERN-62 is applied.

14430 The MSG\_NOSIGNAL symbolic constant is added from The Open Group Technical Standard,  
14431 2006, Extended API Set Part 2.

14432 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
14433 for the **size\_t** type is added.

14434 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0067 [355] is applied.

14435 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0077 [934] is applied.

#### 14436 Issue 8

14437 Austin Group Defects 411 and 1318 are applied, adding SOCK\_NONBLOCK, SOCK\_CLOEXEC,  
14438 SOCK\_CLOFORK, MSG\_CMSG\_CLOEXEC, MSG\_CMSG\_CLOFORK, and *accept4()*.

14439 Austin Group Defect 840 is applied, adding SO\_DOMAIN and SO\_PROTOCOL.

14440 Austin Group Defect 978 is applied, adding CMSG\_SPACE and CMSG\_LEN, and clarifying the  
14441 behavior of CMSG\_NXTHDR when the second argument is a null pointer.

14442 Austin Group Defect 1056 is applied, clarifying the conditions under which CMSG\_NXTHDR  
14443 and CMSG\_FIRSTHDR return a null pointer, and adding a new paragraph to APPLICATION  
14444 USAGE.

14445 Austin Group Defect 1641 is applied, clarifying the requirements for conversions between  
14446 pointers to **sockaddr\_storage** and **storage** structures and between pointers to those structures  
14447 and pointers to protocol-specific address structures.

14448 **NAME**

14449 sys/stat.h — data returned by the stat() function

14450 **SYNOPSIS**

14451 #include <sys/stat.h>

14452 **DESCRIPTION**

14453 The <sys/stat.h> header shall define the structure of the data returned by the *fstat()*, *lstat()*, and  
14454 *stat()* functions.

14455 The <sys/stat.h> header shall define the **stat** structure, which shall include at least the following  
14456 members:

14457	dev_t st_dev	Device ID of device containing file.
14458	ino_t st_ino	File serial number.
14459	mode_t st_mode	Mode of file (see below).
14460	nlink_t st_nlink	Number of hard links to the file.
14461	uid_t st_uid	User ID of file.
14462	gid_t st_gid	Group ID of file.
14463 XSI	dev_t st_rdev	Device ID (if file is character or block special).
14464	off_t st_size	For regular files, the file size in bytes.
14465		For symbolic links, the length in bytes of the
14466		pathname contained in the symbolic link.
14467 SHM		For a shared memory object, the length in bytes.
14468 TYM		For a typed memory object, the length in bytes.
14469		For other file types, the use of this field is
14470		unspecified.
14471	struct timespec st_atim	Last data access timestamp.
14472	struct timespec st_mtim	Last data modification timestamp.
14473	struct timespec st_ctim	Last file status change timestamp.
14474 XSI	blksize_t st_blksize	A file system-specific preferred I/O block size
14475		for this object. In some file system types, this
14476		may vary from file to file.
14477	blkcnt_t st_blocks	Number of blocks allocated for this object.

14478 A *file identity* is uniquely determined by the combination of *st\_dev* and *st\_ino*. At any given time  
14479 in a system, distinct files shall have distinct file identities; hard links to the same file shall have  
14480 the same file identity. Over time, these file identities can be reused for different files. For  
14481 example, the *st\_ino* value can be reused after the last link to a file is unlinked and the space  
14482 occupied by the file has been freed, and the *st\_dev* value associated with a file system can be  
14483 reused if that file system is detached (“unmounted”) and another is attached (“mounted”).

14484 The *st\_nlink* value shall be the number of hard links to the file within the file system in which the  
14485 file resides.

14486 **Note:** The number of links to the file that can be found by traversing the file hierarchy can differ from  
14487 *st\_nlink*. For example, it can be less than *st\_nlink* if a link to the file cannot be reached because it  
14488 is below a directory that has been overlaid with a mount point for a different file system, and it  
14489 can be greater than *st\_nlink* on implementations that allow a file system (or part of one) to be  
14490 duplicated at additional mount points.

14491 XSI The <sys/stat.h> header shall define the **blkcnt\_t**, **blksize\_t**, **dev\_t**, **ino\_t**, **mode\_t**, **nlink\_t**,  
14492 **uid\_t**, **gid\_t**, **off\_t**, and **time\_t** types as described in <sys/types.h>.

14493 The <sys/stat.h> header shall define the **timespec** structure as described in <time.h>. Times  
14494 shall be given in seconds since the Epoch.

14495 Which structure members have meaningful values depends on the type of file. For further  
 14496 information, see the descriptions of *fstat()*, *lstat()*, and *stat()* in the System Interfaces volume of  
 14497 POSIX.1-2024.

14498 For compatibility with earlier versions of this standard, the *st\_atime* macro shall be defined with  
 14499 the value *st\_atim.tv\_sec*. Similarly, *st\_ctime* and *st\_mtime* shall be defined as macros with the  
 14500 values *st\_ctim.tv\_sec* and *st\_mtim.tv\_sec*, respectively.

14501 The <sys/stat.h> header shall define the following symbolic constants for the file types encoded  
 14502 in type **mode\_t**. The values shall be suitable for use in **#if** preprocessing directives:

14503	XSI	<b>S_IFMT</b>	Type of file.
14504		<b>S_IFBLK</b>	Block special.
14505		<b>S_IFCHR</b>	Character special.
14506		<b>S_IFIFO</b>	FIFO special.
14507		<b>S_IFREG</b>	Regular.
14508		<b>S_IFDIR</b>	Directory.
14509		<b>S_IFLNK</b>	Symbolic link.
14510		<b>S_IFSOCK</b>	Socket.

14511 The <sys/stat.h> header shall define the following symbolic constants for the file mode bits  
 14512 encoded in type **mode\_t**, with the indicated numeric values. These macros shall expand to an  
 14513 expression which has a type that allows them to be used, either singly or OR'ed together, as the  
 14514 third argument to *open()* without the need for a **mode\_t** cast. The values shall be suitable for use  
 14515 in **#if** preprocessing directives.

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
S_ISVTX	01000	On directories, restricted deletion flag.

14532 The following macros shall be provided to test whether a file is of the specified type. The value  
 14533 *m* supplied to the macros is the value of *st\_mode* from a **stat** structure. The macro shall evaluate  
 14534 to a non-zero value if the test is true; 0 if the test is false.

14535 **S\_ISBLK(*m*)** Test for a block special file.

14536		S_ISCHR( <i>m</i> )	Test for a character special file.
14537		S_ISDIR( <i>m</i> )	Test for a directory.
14538		S_ISFIFO( <i>m</i> )	Test for a pipe or FIFO special file.
14539		S_ISREG( <i>m</i> )	Test for a regular file.
14540		S_ISLNK( <i>m</i> )	Test for a symbolic link.
14541		S_ISSOCK( <i>m</i> )	Test for a socket.
14542		The implementation may implement message queues, semaphores, or shared memory objects as distinct file types, in which case these file types need not be encoded in type <b>mode_t</b> . The following macros shall be provided to test whether a file is of the specified type. The value of the <i>buf</i> argument supplied to the macros is a pointer to a <b>stat</b> structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the <b>stat</b> structure referenced by <i>buf</i> . Otherwise, the macro shall evaluate to zero.	
14543			
14544			
14545			
14546			
14547			
14548			
14549		S_TYPEISMQ( <i>buf</i> )	Test for a message queue.
14550		S_TYPEISSEM( <i>buf</i> )	Test for a semaphore.
14551		S_TYPEISSHM( <i>buf</i> )	Test for a shared memory object.
14552	TYM	The implementation may implement typed memory objects as a distinct file type, in which case this file type need not be encoded in type <b>mode_t</b> . The following macro shall test whether a file is of the specified type. The value of the <i>buf</i> argument supplied to the macros is a pointer to a <b>stat</b> structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the <b>stat</b> structure referenced by <i>buf</i> . Otherwise, the macro shall evaluate to zero.	
14553			
14554			
14555			
14556			
14557			
14558		S_TYPEISTMO( <i>buf</i> )	Test macro for a typed memory object.
14559		The <sys/stat.h> header shall define the following symbolic constants as distinct integer values outside of the range [0,999 999 999], for use with the <i>futimens()</i> and <i>utimensat()</i> functions:	
14560			
14561		UTIME_NOW	
14562		UTIME_OMIT	
14563		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.	
14564			
14565		int chmod(const char *, mode_t);	
14566		int fchmod(int, mode_t);	
14567		int fchmodat(int, const char *, mode_t, int);	
14568		int fstat(int, struct stat *);	
14569		int fstatat(int, const char *restrict, struct stat *restrict, int);	
14570		int futimens(int, const struct timespec [2]);	
14571		int lstat(const char *restrict, struct stat *restrict);	
14572		int mkdir(const char *, mode_t);	
14573		int mkdirat(int, const char *, mode_t);	
14574		int mkfifo(const char *, mode_t);	
14575		int mkfifoat(int, const char *, mode_t);	
14576	XSI	int mknod(const char *, mode_t, dev_t);	
14577		int mknodat(int, const char *, mode_t, dev_t);	
14578		int stat(const char *restrict, struct stat *restrict);	



```
14579     mode_t umask(mode_t);
14580     int    utimensat(int, const char *, const struct timespec [2], int);
```

14581 Inclusion of the <sys/stat.h> header may make visible all symbols from the <time.h> header.

#### 14582 APPLICATION USAGE

14583 Use of the macros is recommended for determining the type of a file.

#### 14584 RATIONALE

14585 A conforming C-language application must include <sys/stat.h> for functions that have  
14586 arguments or return values of type **mode\_t**, so that symbolic values for that type can be used.  
14587 An alternative would be to require that these constants are also defined by including  
14588 <sys/types.h>.

14589 The S\_ISUID and S\_ISGID bits may be cleared on any write, not just on *open()*, as some  
14590 historical implementations do.

14591 System calls that update the time entry fields in the **stat** structure must be documented by the  
14592 implementors. POSIX-conforming systems should not update the time entry fields for functions  
14593 listed in the System Interfaces volume of POSIX.1-2024 unless the standard requires that they do,  
14594 except in the case of documented extensions to the standard.

14595 Upon assignment, file timestamps are immediately converted to the resolution of the file system  
14596 by truncation (i.e., the recorded time can be older than the actual time). For example, if the file  
14597 system resolution is 1 microsecond, then a conforming *stat()* must always return an  
14598 *st\_mtim.tv\_nsec* that is a multiple of 1000. Some older implementations returned higher-  
14599 resolution timestamps while the *inode* information was cached, and then spontaneously  
14600 truncated the *tv\_nsec* fields when they were stored to and retrieved from disk, but this behavior  
14601 does not conform.

14602 Note that *st\_dev* must be unique within a Local Area Network (LAN) in a “system” made up of  
14603 multiple computers’ file systems connected by a LAN.

14604 Networked implementations of a POSIX-conforming system must guarantee that all files visible  
14605 within the file tree (including parts of the tree that may be remotely mounted from other  
14606 machines on the network) on each individual processor are uniquely identified by the  
14607 combination of the *st\_ino* and *st\_dev* fields.

14608 The unit for the *st\_blocks* member of the **stat** structure is not defined within POSIX.1-2024. In  
14609 some implementations it is 512 bytes. It may differ on a file system basis. There is no correlation  
14610 between values of the *st\_blocks* and *st\_blksize*, and the *f\_bsize* (from <sys/statvfs.h>) structure  
14611 members.

14612 Traditionally, some implementations defined the multiplier for *st\_blocks* in <sys/param.h> as the  
14613 symbol DEV\_BSIZE.

14614 Some earlier versions of this standard did not specify values for the file mode bit macros. The  
14615 expectation was that some implementors might choose to use a different encoding for these bits  
14616 than the traditional one, and that new applications would use symbolic file modes instead of  
14617 numeric. This version of the standard specifies the traditional encoding, in recognition that  
14618 nearly 20 years after the first publication of this standard numeric file modes are still in  
14619 widespread use by application developers, and that all conforming implementations still use the  
14620 traditional encoding.

14621 **FUTURE DIRECTIONS**

14622 No new S\_IFMT symbolic names for the file type values of **mode\_t** will be defined by  
 14623 POSIX.1-2024; if new file types are required, they will only be testable through *S\_ISxx()* or  
 14624 *S\_TYPEISxxx()* macros instead.

14625 **SEE ALSO**

14626 [<sys/statvfs.h>](#), [<sys/types.h>](#), [<time.h>](#)

14627 XSH *chmod()*, *fchmod()*, *fstat()*, *fstatat()*, *futimens()*, *mkdir()*, *mkfifo()*, *mknod()*, *umask()*

14628 **CHANGE HISTORY**

14629 First released in Issue 1. Derived from Issue 1 of the SVID.

14630 **Issue 5**

14631 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

14632 The type of *st\_blksize* is changed from **long** to **blksize\_t**; the type of *st\_blocks* is changed from  
 14633 **long** to **blkcnt\_t**.

14634 **Issue 6**

14635 The *S\_TYPEISMQ()*, *S\_TYPEISSEM()*, and *S\_TYPEISSHM()* macros are unconditionally  
 14636 mandated.

14637 The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize\_t**  
 14638 and **blkcnt\_t** have been described.

14639 The following new requirements on POSIX implementations derive from alignment with the  
 14640 Single UNIX Specification:

- 14641 • The **dev\_t**, **ino\_t**, **mode\_t**, **nlink\_t**, **uid\_t**, **gid\_t**, **off\_t**, and **time\_t** types are mandated.

14642 *S\_IFSOCK* and *S\_ISSOCK* are added for sockets.

14643 The description of **stat** structure members is changed to reflect contents when file type is a  
 14644 symbolic link.

14645 The test macro *S\_TYPEISTMO* is added for alignment with IEEE Std 1003.1j-2000.

14646 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*.

14647 The *lstat()* function is made mandatory.

14648 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the  
 14649 *st\_blocks* member of the **stat** structure to the RATIONALE.

14650 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/25 is applied, adding to the  
 14651 DESCRIPTION that the **timespec** structure may be defined as described in the [<time.h>](#) header.

14652 **Issue 7**

14653 SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the  
 14654 interfaces should be consulted in order to determine which structure members have meaningful  
 14655 values.

14656 The *fchmodat()*, *fstatat()*, *mkdirat()*, *mkfifoat()*, *mknodat()*, and *utimensat()* functions are added  
 14657 from The Open Group Technical Standard, 2006, Extended API Set Part 2.

14658 The *futimens()* function is added.

14659 This reference page is clarified with respect to macros and symbolic constants.

14660 Changes are made related to support for finegrained timestamps and the *UTIME\_NOW* and  
 14661 *UTIME\_OMIT* symbolic constants are added.

14662 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0068 [207] is applied.

14663 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0078 [531] is applied.

14664 **Issue 8**

14665 Austin Group Defect 732 is applied, clarifying that if message queues, semaphores, shared  
14666 memory objects, or typed memory objects are implemented as distinct file types, they need not  
14667 be encoded in type **mode\_t**.

14668 Austin Group Defect 1314 is applied, clarifying how the *st\_dev* and *st\_ino* values identify files.

14669 Austin Group Defect 1323 is applied, clarifying how the *st\_nlink* value relates to the number of  
14670 links to the file that can be found by traversing the file hierarchy.

14671 **NAME**

14672 sys/statvfs.h — VFS File System information structure

14673 **SYNOPSIS**

14674 #include <sys/statvfs.h>

14675 **DESCRIPTION**

14676 The <sys/statvfs.h> header shall define the **statvfs** structure, which shall include at least the  
14677 following members:

- 14678 unsigned long f\_bsize File system block size.
- 14679 unsigned long f\_frsize Fundamental file system block size.
- 14680 fsblkcnt\_t f\_blocks Total number of blocks on file system in units of *f\_frsize*.
- 14681 fsblkcnt\_t f\_bfree Total number of free blocks.
- 14682 fsblkcnt\_t f\_bavail Number of free blocks available to non-privileged process.
- 14683 fsfilcnt\_t f\_files Total number of file serial numbers.
- 14684 fsfilcnt\_t f\_ffree Total number of free file serial numbers.
- 14685 fsfilcnt\_t f\_favail Number of file serial numbers available to non-privileged process.
- 14686 fsfilcnt\_t f\_favail
- 14687
- 14688 unsigned long f\_fsid File system ID.
- 14689 unsigned long f\_flag Bit mask of *f\_flag* values.
- 14690 unsigned long f\_namemax Maximum filename length.

14691 The <sys/statvfs.h> header shall define the **fsblkcnt\_t** and **fsfilcnt\_t** types as described in  
14692 <sys/types.h>.

14693 The <sys/statvfs.h> header shall define the following symbolic constants for the *f\_flag* member:

- 14694 ST\_RDONLY Read-only file system.
- 14695 ST\_NOSUID Does not support the semantics of the ST\_ISUID and ST\_ISGID file mode bits.

14696 The following shall be declared as functions and may also be defined as macros. Function  
14697 prototypes shall be provided.

```
14698 int fstatvfs(int, struct statvfs *);
14699 int statvfs(const char *restrict, struct statvfs *restrict);
```

14700 **APPLICATION USAGE**

14701 None.

14702 **RATIONALE**

14703 None.

14704 **FUTURE DIRECTIONS**

14705 None.

14706 **SEE ALSO**

14707 <sys/types.h>

14708 XSH *fstatvfs()*

14709 **CHANGE HISTORY**

14710 First released in Issue 4, Version 2.

14711 **Issue 5**

14712 The type of *f\_blocks*, *f\_bfree*, and *f\_bavail* is changed from **unsigned long** to **fsblkcnt\_t**; the type of  
14713 *f\_files*, *f\_ffree*, and *f\_favail* is changed from **unsigned long** to **fsfilcnt\_t**.

14714 **Issue 6**

14715 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types `fsblkcnt_t`  
14716 and `fsfilcnt_t` have been described.

14717 The **restrict** keyword is added to the prototype for `statvfs()`.

14718 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of  
14719 ST\_NOSUID from “Does not support `setuid()/setgid()` semantics” to “Does not support the  
14720 semantics of the ST\_ISUID and ST\_ISGID file mode bits”.

14721 **Issue 7**

14722 The `<sys/statvfs.h>` header is moved from the XSI option to the Base.

14723 This reference page is clarified with respect to macros and symbolic constants.

14724 **NAME**

14725 sys/time.h — time types

14726 **SYNOPSIS**14727 XSI `#include <sys/time.h>`14728 **DESCRIPTION**14729 The **<sys/time.h>** header shall define the **fd\_set** type and the **timeval** structure, as described in  
14730 **<sys/select.h>**.14731 The **<sys/time.h>** header shall define the **time\_t** and **suseconds\_t** types as described in  
14732 **<sys/types.h>**.14733 The **<sys/time.h>** header shall define the following as described in **<sys/select.h>**:14734 `FD_CLR()`  
14735 `FD_ISSET()`  
14736 `FD_SET()`  
14737 `FD_ZERO()`  
14738 `FD_SETSIZE`14739 The following shall be declared as functions and may also be defined as macros. Function  
14740 prototypes shall be provided.14741 `int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,`  
14742 `struct timeval *restrict);`  
14743 `int utimes(const char *, const struct timeval [2]);`14744 Inclusion of the **<sys/time.h>** header may make visible all symbols from the **<sys/select.h>**  
14745 header.14746 **APPLICATION USAGE**

14747 None.

14748 **RATIONALE**14749 The **<sys/time.h>** header refers to **<sys/select.h>** for the definition of the **timeval** structure,  
14750 instead of the other way round, because **<sys/time.h>** is an optional (XSI) header whereas  
14751 **<sys/select.h>** is mandatory.14752 **FUTURE DIRECTIONS**

14753 None.

14754 **SEE ALSO**14755 **<sys/select.h>**, **<sys/types.h>**14756 XSH *futimens()*, *pselect()*14757 **CHANGE HISTORY**

14758 First released in Issue 4, Version 2.

14759 **Issue 5**14760 The type of *tv\_usec* is changed from **long** to **suseconds\_t**.14761 **Issue 6**14762 The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*.14763 The note is added that inclusion of this header may also make symbols visible from  
14764 **<sys/select.h>**.14765 The *utimes()* function is marked LEGACY.

14766 **Issue 7**

14767 This reference page is clarified with respect to macros and symbolic constants.

14768 **Issue 8**

14769 Austin Group Defect 1171 is applied, replacing the **timeval** structure definition with a reference  
14770 to its description in <sys/select.h>.

14771 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

**14772 NAME**

14773 `sys/times.h` — file access and modification times structure

**14774 SYNOPSIS**

14775 `#include <sys/times.h>`

**14776 DESCRIPTION**

14777 The **<sys/times.h>** header shall define the **tms** structure, which is returned by *times()* and shall  
14778 include at least the following members:

14779 `clock_t tms_utime` User CPU time.

14780 `clock_t tms_stime` System CPU time.

14781 `clock_t tms_cutime` User CPU time of terminated child processes.

14782 `clock_t tms_cstime` System CPU time of terminated child processes.

14783 The **<sys/times.h>** header shall define the **clock\_t** type as described in **<sys/types.h>**.

14784 The following shall be declared as a function and may also be defined as a macro. A function  
14785 prototype shall be provided.

14786 `clock_t times(struct tms *);`

**14787 APPLICATION USAGE**

14788 None.

**14789 RATIONALE**

14790 None.

**14791 FUTURE DIRECTIONS**

14792 None.

**14793 SEE ALSO**

14794 [<sys/types.h>](#)

14795 XSH *times()*

**14796 CHANGE HISTORY**

14797 First released in Issue 1. Derived from Issue 1 of the SVID.



14798	<b>NAME</b>	
14799		sys/types.h — data types
14800	<b>SYNOPSIS</b>	
14801		#include <sys/types.h>
14802	<b>DESCRIPTION</b>	
14803		The <sys/types.h> header shall define at least the following types:
14804	<b>blkcnt_t</b>	Used for file block counts.
14805	<b>blksize_t</b>	Used for block sizes.
14806	<b>clock_t</b>	Used for system times in clock ticks or CLOCKS_PER_SEC; see
14807		<time.h>.
14808	<b>clockid_t</b>	Used for clock ID type in the clock and timer functions.
14809	<b>dev_t</b>	Used for device IDs.
14810	<b>fsblkcnt_t</b>	Used for file system block counts.
14811	<b>fsfilcnt_t</b>	Used for file system file counts.
14812	<b>gid_t</b>	Used for group IDs.
14813	<b>id_t</b>	Used as a general identifier; can be used to contain at least a <b>pid_t</b> ,
14814		<b>uid_t</b> , or <b>gid_t</b> .
14815	<b>ino_t</b>	Used for file serial numbers.
14816	XSI <b>key_t</b>	Used for XSI interprocess communication.
14817	<b>mode_t</b>	Used for some file attributes.
14818	<b>nlink_t</b>	Used for link counts.
14819	<b>off_t</b>	Used for file sizes.
14820	<b>pid_t</b>	Used for process IDs and process group IDs.
14821	<b>pthread_attr_t</b>	Used to identify a thread attribute object.
14822	<b>pthread_barrier_t</b>	Used to identify a barrier.
14823	<b>pthread_barrierattr_t</b>	Used to define a barrier attributes object.
14824	<b>pthread_cond_t</b>	Used for condition variables.
14825	<b>pthread_condattr_t</b>	Used to identify a condition attribute object.
14826	<b>pthread_key_t</b>	Used for thread-specific data keys.
14827	<b>pthread_mutex_t</b>	Used for mutexes.
14828	<b>pthread_mutexattr_t</b>	Used to identify a mutex attribute object.
14829	<b>pthread_once_t</b>	Used for dynamic package initialization.
14830	<b>pthread_rwlock_t</b>	Used for read-write locks.
14831	<b>pthread_rwlockattr_t</b>	Used for read-write lock attributes.
14832	<b>pthread_spinlock_t</b>	Used to identify a spin lock.

14833	<b>pthread_t</b>	Used to identify a thread.
14834	<b>reclen_t</b>	Used for directory entry lengths.
14835	<b>size_t</b>	Used for sizes of objects.
14836	<b>ssize_t</b>	Used for a count of bytes or an error indication.
14837	<b>suseconds_t</b>	Used for time in microseconds.
14838	<b>time_t</b>	Used for time in seconds.
14839	<b>timer_t</b>	Used for timer ID returned by <i>timer_create()</i> .
14840	<b>uid_t</b>	Used for user IDs.
14841	All of the types shall be defined as arithmetic types of an appropriate length, with the following exceptions:	
14842		
14843	<b>pthread_attr_t</b>	
14844	<b>pthread_barrier_t</b>	
14845	<b>pthread_barrierattr_t</b>	
14846	<b>pthread_cond_t</b>	
14847	<b>pthread_condattr_t</b>	
14848	<b>pthread_key_t</b>	
14849	<b>pthread_mutex_t</b>	
14850	<b>pthread_mutexattr_t</b>	
14851	<b>pthread_once_t</b>	
14852	<b>pthread_rwlock_t</b>	
14853	<b>pthread_rwlockattr_t</b>	
14854	<b>pthread_spinlock_t</b>	
14855	<b>pthread_t</b>	
14856	<b>timer_t</b>	
14857	Additionally:	
14858	• <b>mode_t</b> shall be an integer type.	
14859	• <b>dev_t</b> shall be an integer type.	
14860	• <b>nlink_t</b> , <b>uid_t</b> , <b>gid_t</b> , and <b>id_t</b> shall be integer types.	
14861	• <b>blkcnt_t</b> and <b>off_t</b> shall be signed integer types.	
14862	• <b>fsblkcnt_t</b> , <b>fsfilcnt_t</b> , <b>reclen_t</b> , and <b>ino_t</b> shall be defined as unsigned integer types.	
14863	• <b>size_t</b> shall be an unsigned integer type.	
14864	• <b>blksize_t</b> , <b>pid_t</b> , and <b>ssize_t</b> shall be signed integer types.	
14865	• <b>clock_t</b> shall be an integer or real-floating type.	
14866	CX	• <b>time_t</b> shall be an integer type with a width (see <stdint.h>) of at least 64 bits.
14867	The type <b>ssize_t</b> shall be capable of storing values at least in the range $[-1, \{\text{SSIZE\_MAX}\}]$ .	
14868	XSI	The type <b>suseconds_t</b> shall be a signed integer type capable of storing values at least in the range $[-1, 1\,000\,000]$ .
14869		
14870	The implementation shall support one or more programming environments in which the widths of <b>blksize_t</b> , <b>pid_t</b> , <b>size_t</b> , <b>ssize_t</b> , and <b>suseconds_t</b> are no greater than the width of type <b>long</b> .	
14871		
14872	The names of these programming environments can be obtained using the <i>confstr()</i> function or the <i>getconf</i> utility.	
14873		

14874 There are no defined comparison or assignment operators for the following types:

- 14875 **pthread\_attr\_t**
- 14876 **pthread\_barrier\_t**
- 14877 **pthread\_barrierattr\_t**
- 14878 **pthread\_cond\_t**
- 14879 **pthread\_condattr\_t**
- 14880 **pthread\_mutex\_t**
- 14881 **pthread\_mutexattr\_t**
- 14882 **pthread\_rwlock\_t**
- 14883 **pthread\_rwlockattr\_t**
- 14884 **pthread\_spinlock\_t**
- 14885 **timer\_t**

14886 **APPLICATION USAGE**

14887 None.

14888 **RATIONALE**

14889 None.

14890 **FUTURE DIRECTIONS**

14891 None.

14892 **SEE ALSO**

14893 [<stdint.h>](#), [<time.h>](#)

14894 XSH *confstr()*

14895 XCU *getconf*

14896 **CHANGE HISTORY**

14897 First released in Issue 1. Derived from Issue 1 of the SVID.

14898 **Issue 5**

14899 The **clockid\_t** and **timer\_t** types are defined for alignment with the POSIX Realtime Extension.

14900 The types **blkcnt\_t**, **blksize\_t**, **fsblkcnt\_t**, **fsfilcnt\_t**, and **suseconds\_t** are added.

14901 Large File System extensions are added.

14902 Updated for alignment with the POSIX Threads Extension.

14903 **Issue 6**

14904 The **pthread\_barrier\_t**, **pthread\_barrierattr\_t**, and **pthread\_spinlock\_t** types are added for alignment with IEEE Std 1003.1j-2000.

14906 The margin code is changed from XSI to THR for the **pthread\_rwlock\_t** and **pthread\_rwlockattr\_t** types as Read-Write Locks have been absorbed into the POSIX Threads option. The threads types are marked THR.

14909 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/26 is applied, adding **pthread\_t** to the list of types that are not required to be arithmetic types, thus allowing **pthread\_t** to be defined as a structure.

14912 **Issue 7**

14913 Austin Group Interpretation 1003.1-2001 #033 is applied, requiring **key\_t** to be an arithmetic type.

14915 The Trace option types are marked obsolescent.

14916 The **clock\_t** and **id\_t** types are moved from the XSI option to the Base.

14917 The **pthread\_barrier\_t** and **pthread\_barrierattr\_t** types are moved from the Barriers option to  
14918 the Base.

14919 The **pthread\_spinlock\_t** type is moved from the Spin Locks option to the Base.

14920 Functionality relating to the Timers and Threads options is moved to the Base.

14921 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0069 [210], XBD/TC1-2008/0070 [28],  
14922 XBD/TC1-2008/0071 [376], XBD/TC1-2008/0072 [210], and XBD/TC1-2008/0073 [327] are  
14923 applied.

14924 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0079 [856] and XBD/TC2-2008/0080  
14925 [659] are applied.

14926 **Issue 8**

14927 Austin Group Defect 697 is applied, adding **reclen\_t**.

14928 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

14929 Austin Group Defect 1462 is applied, changing **time\_t** to have a width of at least 64 bits.

14930 **NAME**

14931 sys/uio.h — definitions for vector I/O operations

14932 **SYNOPSIS**14933 XSI `#include <sys/uio.h>`14934 **DESCRIPTION**14935 The <sys/uio.h> header shall define the **iovec** structure, which shall include at least the  
14936 following members:14937 `void *iov_base` Base address of a memory region for input or output.14938 `size_t iov_len` The size of the memory pointed to by *iov\_base*.14939 The <sys/uio.h> header uses the **iovec** structure for scatter/gather I/O.14940 The <sys/uio.h> header shall define the **ssize\_t** and **size\_t** types as described in <sys/types.h>.14941 The following shall be declared as functions and may also be defined as macros. Function  
14942 prototypes shall be provided.14943 `ssize_t readv(int, const struct iovec *, int);`14944 `ssize_t writev(int, const struct iovec *, int);`14945 **APPLICATION USAGE**14946 The implementation can put a limit on the number of scatter/gather elements which can be  
14947 processed in one call. The symbol {IOV\_MAX} defined in <limits.h> should always be used to  
14948 learn about the limits instead of assuming a fixed value.14949 **RATIONALE**14950 Traditionally, the maximum number of scatter/gather elements the system can process in one  
14951 call were described by the symbolic value {UIO\_MAXIOV}. In IEEE Std 1003.1-2001 this value is  
14952 replaced by the constant {IOV\_MAX} which can be found in <limits.h>.14953 **FUTURE DIRECTIONS**

14954 None.

14955 **SEE ALSO**

14956 &lt;limits.h&gt;, &lt;sys/types.h&gt;

14957 XSH *read()*, *readv()*, *write()*, *writev()*14958 **CHANGE HISTORY**

14959 First released in Issue 4, Version 2.

14960 **Issue 6**

14961 Text referring to scatter/gather I/O is added to the DESCRIPTION.

14962 **NAME**

14963 sys/un.h — definitions for UNIX domain sockets

14964 **SYNOPSIS**

14965 #include &lt;sys/un.h&gt;

14966 **DESCRIPTION**14967 The <sys/un.h> header shall define the **sockaddr\_un** structure, which shall include at least the  
14968 following members:14969 sa\_family\_t sun\_family Address family.  
14970 char sun\_path[size] Socket pathname storage.14971 The *sun\_path* member shall be the last member of the **sockaddr\_un** structure, where *size* shall be  
14972 an implementation-provided constant size of at least 92 bytes. This *size* value need not be  
14973 accessible as a constant available for use in the application namespace.14974 The **sockaddr\_un** structure is used to store addresses for UNIX domain sockets. Pointers to this  
14975 type shall be cast by applications to **struct sockaddr \*** for use with socket functions.14976 The <sys/un.h> header shall define the **sa\_family\_t** type as described in <sys/socket.h>.14977 **APPLICATION USAGE**14978 The size of *sun\_path* is required to be constant, but intentionally does not have a specified name  
14979 for that constant. Historically, different implementations used different sizes. For example, 4.3  
14980 BSD used a size of 108, and 4.4 BSD used a size of 104. Since most implementations originate  
14981 from BSD versions, the size is typically in the range 92 to 108. An application can deduce the size  
14982 by using `sizeof(((struct sockaddr_un *)0)->sun_path)`.14983 Applications should not assume a particular length for *sun\_path* or assume that it can hold  
14984 `{_POSIX_PATH_MAX}` bytes (256).14985 Although applications are required to initialize all members (including any non-standard ones)  
14986 of a **sockaddr\_in6** structure (see <netinet/in.h>, on page 318), the same is not required for the  
14987 **sockaddr\_un** structure, since historically many applications only initialized the standard  
14988 members. Despite this, applications are encouraged to initialize **sockaddr\_un** structures in a  
14989 manner similar to the required initialization of **sockaddr\_in6** structures.14990 **RATIONALE**14991 Some implementations expose a macro `SUN_LEN` for the size of a pathname stored in *sun\_path*.  
14992 However, this was not widely adopted, and differences on how a terminating null byte is  
14993 interpreted between implementations did not make it worth standardizing.14994 **FUTURE DIRECTIONS**

14995 None.

14996 **SEE ALSO**

14997 &lt;netinet/in.h&gt;, &lt;sys/socket.h&gt;

14998 XSH *bind()*, *socket()*, *socketpair()*14999 **CHANGE HISTORY**

15000 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

15001 **Issue 7**15002 The value for `{_POSIX_PATH_MAX}` is updated to 256.

15003 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0074 [355] is applied.

15004 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0081 [934] is applied.

15005 **Issue 8**

15006

15007

Austin Group Defect 561 is applied, changing the requirements for the *sun\_path* member of the **sockaddr\_un** structure.

**15008 NAME**

15009           sys/utsname.h — system name structure

**15010 SYNOPSIS**

15011           #include <sys/utsname.h>

**15012 DESCRIPTION**

15013           The **<sys/utsname.h>** header shall define the structure **utsname** which shall include at least the  
15014           following members:

15015           char   sysname[]   Name of this implementation of the operating system.  
15016           char   nodename[]   Name of this node within the communications  
15017                                  network to which this node is attached, if any.  
15018           char   release[]   Current release level of this implementation.  
15019           char   version[]   Current version level of this release.  
15020           char   machine[]   Name of the hardware type on which the system is running.

15021           The character arrays are of unspecified size, but the data stored in them shall be terminated by a  
15022           null byte.

15023           The following shall be declared as a function and may also be defined as a macro:

15024           int  uname(struct utsname \*);

**15025 APPLICATION USAGE**

15026           None.

**15027 RATIONALE**

15028           None.

**15029 FUTURE DIRECTIONS**

15030           None.

**15031 SEE ALSO**

15032           XSH *uname()*

**15033 CHANGE HISTORY**

15034           First released in Issue 1. Derived from Issue 1 of the SVID.

**15035 Issue 6**

15036           IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/27 is applied, changing the description of  
15037           *nodename* within the **utsname** structure from “an implementation-defined communications  
15038           network” to “the communications network to which this node is attached, if any”.



15039 **NAME**

15040 sys/wait.h — declarations for waiting

15041 **SYNOPSIS**

15042 #include <sys/wait.h>

15043 **DESCRIPTION**

15044 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

- 15045 XSI **WCONTINUED** Report status of continued child process.
- 15046 **WNOHANG** Do not hang if no status is available; return immediately.
- 15047 **WUNTRACED** Report status of stopped child process.

15048 The <sys/wait.h> header shall define the following macros for analysis of process status values:

- 15049 **WCOREDUMP** True if **WIFSIGNALED** is true and creation of a core image was
- 15050 attempted.

15051 **Note:** The use of the word “attempted” here means that the process terminated abnormally with  
 15052 additional actions (see **SIG\_DFL** in XSH Section 2.4.3, on page 516). A core image might or  
 15053 might not have been produced. Some implementations do not set this bit if a core image was not  
 15054 produced, but this is not a requirement.

- 15055 **WEXITSTATUS** Return exit status.
- 15056 XSI **WIFCONTINUED** True if child has been continued.
- 15057 **WIFEXITED** True if child exited normally.
- 15058 **WIFSIGNALED** True if child exited due to uncaught signal.
- 15059 **WIFSTOPPED** True if child stopped due to uncaught signal.
- 15060 **WSTOPSIG** Return signal number that caused process to stop.
- 15061 **WTERMSIG** Return signal number that caused process to terminate.

15062 The <sys/wait.h> header shall define the following symbolic constants as possible values for the  
 15063 *options* argument to *waitid()*:

- 15064 **WEXITED** Wait for processes that have terminated.
- 15065 **WNOWAIT** Keep the process whose status is returned in *infop* in a waitable state.
- 15066 **WSTOPPED** Status is returned for any child that has stopped upon receipt of a signal.

15067 XSI The **WCONTINUED** and **WNOHANG** constants, described above for *waitpid()*, can also be  
 15068 used with *waitid()*.

15069 The type **idtype\_t** shall be defined as an enumeration type whose possible values shall include  
 15070 at least the following:

- 15071 **P\_ALL**
- 15072 **P\_PGID**
- 15073 **P\_PID**

15074 The <sys/wait.h> header shall define the **id\_t** and **pid\_t** types as described in <sys/types.h>.

15075 The <sys/wait.h> header shall define the **siginfo\_t** type and the **sigval** union as described in  
 15076 <signal.h>.

15077 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h>.

15078           The following shall be declared as functions and may also be defined as macros. Function  
15079           prototypes shall be provided.

```
15080       pid_t  wait(int *);  
15081       int    waitid(idtype_t, id_t, siginfo_t *, int);  
15082       pid_t  waitpid(pid_t, int *, int);
```

**15083 APPLICATION USAGE**

15084           None.

**15085 RATIONALE**

15086           None.

**15087 FUTURE DIRECTIONS**

15088           None.

**15089 SEE ALSO**

15090           [<signal.h>](#), [<sys/resource.h>](#), [<sys/types.h>](#)

15091           XSH *wait()*, *waitid()*

**15092 CHANGE HISTORY**

15093           First released in Issue 3.

15094           Included for alignment with the POSIX.1-1988 standard.

**15095 Issue 6**

15096           The *wait3()* function is removed.

**15097 Issue 7**

15098           The *waitid()* function and symbolic constants for its *options* argument are moved to the Base.

15099           The description of the WNOHANG constant is clarified.

15100           The requirement for **<sys/wait.h>** to define the **rusage** structure as described in  
15101 **<sys/resource.h>** is removed, and **<sys/wait.h>** is no longer allowed to make visible all symbols  
15102 from **<sys/resource.h>**.

15103           POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0082 [579] and XBD/TC2-2008/0083  
15104 [564] are applied.

**15105 Issue 8**

15106           Austin Group Defect 1141 is applied, adding WCOREDUMP and changing the description of  
15107 WIFSTOPPED.

15108           Austin Group Defect 1332 is applied, changing the description of WEXITED.

15109 **NAME**

15110            syslog.h — definitions for system error logging

15111 **SYNOPSIS**

15112 XSI        #include <syslog.h>

15113 **DESCRIPTION**

15114        The <syslog.h> header shall define the following symbolic constants, zero or more of which  
15115        may be OR'ed together to form the *logopt* option of *openlog()*:

15116        LOG\_PID            Log the process ID with each message.

15117        LOG\_CONS          Log to the system console on error.

15118        LOG\_NDELAY         Connect to syslog daemon immediately.

15119        LOG\_ODELAY         Delay open until *syslog()* is called.

15120        LOG\_NOWAIT         Do not wait for child processes.

15121        The <syslog.h> header shall define the following symbolic constants for use as the *facility*  
15122        argument to *openlog()*:

15123        LOG\_KERN          Reserved for message generated by the system.

15124        LOG\_USER          Message generated by a process.

15125        LOG\_MAIL          Reserved for message generated by mail system.

15126        LOG\_NEWS         Reserved for message generated by news system.

15127        LOG\_UUCP         Reserved for message generated by UUCP system.

15128        LOG\_DAEMON        Reserved for message generated by system daemon.

15129        LOG\_AUTH         Reserved for message generated by authorization daemon.

15130        LOG\_CRON         Reserved for message generated by clock daemon.

15131        LOG\_LPR          Reserved for message generated by printer system.

15132        LOG\_LOCAL0        Reserved for local use.

15133        LOG\_LOCAL1        Reserved for local use.

15134        LOG\_LOCAL2        Reserved for local use.

15135        LOG\_LOCAL3        Reserved for local use.

15136        LOG\_LOCAL4        Reserved for local use.

15137        LOG\_LOCAL5        Reserved for local use.

15138        LOG\_LOCAL6        Reserved for local use.

15139        LOG\_LOCAL7        Reserved for local use.

15140        The <syslog.h> header shall define the following macros for constructing the *maskpri* argument  
15141        to *setlogmask()*. The following macros expand to an expression of type **int** when the argument  
15142        *pri* is an expression of type **int**:

15143        LOG\_MASK(*pri*)     A mask for priority *pri*.

15144 LOG\_UPTO(*pri*) A mask for all priorities from LOG\_EMERG through *pri* inclusive in the  
15145 order listed in the list of priority symbolic constants below. Any  
15146 additional implementation-defined priorities not included in the list  
15147 below shall not be included in the mask.

15148 The **<syslog.h>** header shall define the following symbolic constants for use as the severity level  
15149 portion of the *priority* argument of *syslog()* and the *pri* argument of the LOG\_MASK(*pri*) and  
15150 LOG\_UPTO(*pri*) macros:

15151 LOG\_EMERG A panic condition was reported to all processes.  
15152 LOG\_ALERT A condition that should be corrected immediately.  
15153 LOG\_CRIT A critical condition.  
15154 LOG\_ERR An error message.  
15155 LOG\_WARNING A warning message.  
15156 LOG\_NOTICE A condition requiring special handling.  
15157 LOG\_INFO A general information message.  
15158 LOG\_DEBUG A message useful for debugging programs.

15159 The following shall be declared as functions and may also be defined as macros. Function  
15160 prototypes shall be provided.

```
15161 void closelog(void);  
15162 void openlog(const char *, int, int);  
15163 int setlogmask(int);  
15164 void syslog(int, const char *, ...);
```

#### 15165 APPLICATION USAGE

15166 None.

#### 15167 RATIONALE

15168 None.

#### 15169 FUTURE DIRECTIONS

15170 None.

#### 15171 SEE ALSO

15172 XSH [closelog\(\)](#)

#### 15173 CHANGE HISTORY

15174 First released in Issue 4, Version 2.

#### 15175 Issue 5

15176 Moved from X/Open UNIX to BASE.

#### 15177 Issue 7

15178 This reference page is clarified with respect to macros and symbolic constants.

#### 15179 Issue 8

15180 Austin Group Defect 1033 is applied, adding the LOG\_UPTO macro.

15181 **NAME**  
 15182 tar.h — extended tar definitions

15183 **SYNOPSIS**  
 15184 #include <tar.h>

15185 **DESCRIPTION**  
 15186 The <tar.h> header shall define the following symbolic constants with the indicated values.  
 15187 General definitions:

Name	Value	Description
TMAGIC	"ustar"	Used in the magic field in the <b>ustar</b> header block, including the trailing null byte.
TMAGLEN	6	Length in octets of the magic field.
TVERSION	"00"	Used in the version field in the <b>ustar</b> header block, excluding the trailing null byte.
TVERSLEN	2	Length in octets of the version field.

15195 *Typeflag* field definitions:

Name	Value	Description
REGTYPE	' 0'	Regular file.
AREGTYPE	' \0'	Regular file.
LNKTYPE	' 1'	Hard Link.
SYMTYPE	' 2'	Symbolic link.
CHRTYPE	' 3'	Character special.
BLKTYPE	' 4'	Block special.
DIRTYPE	' 5'	Directory.
FIFOTYPE	' 6'	FIFO special.
CONTTYPE	' 7'	Reserved.

15206 *Mode* field bit definitions (octal):

Name	Value	Description
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

15220 **APPLICATION USAGE**

15221 None.

15222 **RATIONALE**

15223 None.

15224 **FUTURE DIRECTIONS**

15225 None.

15226 **SEE ALSO**

15227 XCU *pax*

15228 **CHANGE HISTORY**

15229 First released in Issue 3. Derived from the POSIX.1-1988 standard.

15230 **Issue 6**

15231 The SEE ALSO section is updated to refer to *pax*.

15232 **Issue 7**

15233 This reference page is clarified with respect to macros and symbolic constants.

15234 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0084 [707] is applied.

15235 **Issue 8**

15236 Austin Group Defect 1380 is applied, changing the description of LNKTYPE.

15237 **NAME**

15238 `termios.h` — define values for `termios`

15239 **SYNOPSIS**

15240 `#include <termios.h>`

15241 **DESCRIPTION**

15242 The **<termios.h>** header shall contain the definitions used by the terminal I/O interfaces (see  
15243 [Chapter 11](#) (on page 199) for the structures and names defined).

15244 **The `termios` Structure**

15245 The **<termios.h>** header shall define the following data types through **typedef**:

15246 **cc\_t** Used for terminal special characters.

15247 **speed\_t** Used for terminal baud rates.

15248 **tcflag\_t** Used for terminal modes.

15249 The above types shall be all unsigned integer types.

15250 The implementation shall support one or more programming environments in which the widths  
15251 of **cc\_t**, **speed\_t**, and **tcflag\_t** are no greater than the width of type **long**. The names of these  
15252 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

15253 The **<termios.h>** header shall define the **termios** structure, which shall include at least the  
15254 following members:

- 15255 `tcflag_t c_iflag` Input modes.
- 15256 `tcflag_t c_oflag` Output modes.
- 15257 `tcflag_t c_cflag` Control modes.
- 15258 `tcflag_t c_lflag` Local modes.
- 15259 `cc_t c_cc[NCCS]` Control characters.

15260 The **<termios.h>** header shall define the following symbolic constant:

15261 **NCCS** Size of the array `c_cc` for control characters.

15262 The **<termios.h>** header shall define the following symbolic constants for use as subscripts for  
15263 the array `c_cc`:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR	VINTR	INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

15277 The subscript values shall be suitable for use in **#if** preprocessing directives and shall be distinct,  
15278 except that the **VMIN** and **VTIME** subscripts may have the same values as the **VEOF** and **VEOL**  
15279 subscripts, respectively.

15280 **Input Modes**

15281 The <termios.h> header shall define the following symbolic constants for use as flags in the  
15282 *c\_iflag* field. The *c\_iflag* field describes the basic terminal input control.

- 15283 BRKINT Signal interrupt on break.
- 15284 ICRNL Map CR to NL on input.
- 15285 IGNBRK Ignore break condition.
- 15286 IGNCR Ignore CR.
- 15287 IGNPAR Ignore characters with parity errors.
- 15288 INLCR Map NL to CR on input.
- 15289 INPCK Enable input parity check.
- 15290 ISTRIP Strip character.
- 15291 IXANY Enable any character to restart output.
- 15292 IXOFF Enable start/stop input control.
- 15293 IXON Enable start/stop output control.
- 15294 PARMRK Mark parity errors.

15295 **Output Modes**

15296 The <termios.h> header shall define the following symbolic constants for use as flags in the  
15297 *c\_oflag* field. The *c\_oflag* field specifies the system treatment of output.

- 15298 OPOST Post-process output.
- 15299 XSI ONLCR Map NL to CR-NL on output.
- 15300 XSI OCRNL Map CR to NL on output.
- 15301 XSI ONOCR No CR output at column 0.
- 15302 XSI ONLRET NL performs CR function.
- 15303 XSI OFDEL Fill is DEL.
- 15304 XSI OFILL Use fill characters for delay.
- 15305 XSI NLDLY Select newline delays:
  - 15306 NL0 Newline type 0.
  - 15307 NL1 Newline type 1.
- 15308 XSI CRDLY Select carriage-return delays:
  - 15309 CR0 Carriage-return delay type 0.
  - 15310 CR1 Carriage-return delay type 1.
  - 15311 CR2 Carriage-return delay type 2.
  - 15312 CR3 Carriage-return delay type 3.



15313	XSI	<b>TABDLY</b>	Select horizontal-tab delays:
15314		TAB0	Horizontal-tab delay type 0.
15315		TAB1	Horizontal-tab delay type 1.
15316		TAB2	Horizontal-tab delay type 2.
15317		TAB3	Expand tabs to spaces.

15318	XSI	<b>BSDLY</b>	Select backspace delays:
15319		BS0	Backspace-delay type 0.
15320		BS1	Backspace-delay type 1.

15321	XSI	<b>VTDLY</b>	Select vertical-tab delays:
15322		VT0	Vertical-tab delay type 0.
15323		VT1	Vertical-tab delay type 1.

15324	XSI	<b>FFDLY</b>	Select form-feed delays:
15325		FF0	Form-feed delay type 0.
15326		FF1	Form-feed delay type 1.

15327      **Baud Rate Selection**

15328      The <termios.h> header shall define the following symbolic constants for use as values of  
 15329      objects of type **speed\_t**.

15330      The input and output baud rates are stored in the **termios** structure. These are the valid values  
 15331      for objects of type **speed\_t**. Not all baud rates need be supported by the underlying hardware.

15332	B0	Hang up
15333	B50	50 baud
15334	B75	75 baud
15335	B110	110 baud
15336	B134	134.5 baud
15337	B150	150 baud
15338	B200	200 baud
15339	B300	300 baud
15340	B600	600 baud
15341	B1200	1 200 baud
15342	B1800	1 800 baud
15343	B2400	2 400 baud

15344	B4800	4 800 baud
15345	B9600	9 600 baud
15346	B19200	19 200 baud
15347	B38400	38 400 baud

### 15348 **Control Modes**

15349 The **<termios.h>** header shall define the following symbolic constants for use as flags in the  
 15350 *c\_cflag* field. The *c\_cflag* field describes the hardware control of the terminal; not all values  
 15351 specified are required to be supported by the underlying hardware.

15352	CSIZE	Character size:
15353		CS5 5 bits
15354		CS6 6 bits
15355		CS7 7 bits
15356		CS8 8 bits
15357	CSTOPB	Send two stop bits, else one.
15358	CREAD	Enable receiver.
15359	PARENB	Parity enable.
15360	PARODD	Odd parity, else even.
15361	HUPCL	Hang up on last close.
15362	CLOCAL	Ignore modem status lines.

15363 The implementation shall support the functionality associated with the symbols CS7, CS8,  
 15364 CSTOPB, PARODD, and PARENB.

### 15365 **Local Modes**

15366 The **<termios.h>** header shall define the following symbolic constants for use as flags in the  
 15367 *c\_lflag* field. The *c\_lflag* field of the argument structure is used to control various terminal  
 15368 functions.

15369	ECHO	Enable echo.
15370	ECHOE	Echo erase character as error-correcting backspace.
15371	ECHOK	Echo KILL.
15372	ECHONL	Echo NL.
15373	ICANON	Canonical input (erase and kill processing).
15374	IEXTEN	Enable extended input character processing.
15375	ISIG	Enable signals.
15376	NOFLSH	Disable flush after interrupt or quit.
15377	TOSTOP	Send SIGTTOU for background output.

15378       **The winsize Structure**

15379       The <termios.h> header shall define the **winsize** structure, which shall include at least the  
 15380       following members:

15381       unsigned short ws\_row        Rows, in characters.  
 15382       unsigned short ws\_col       Columns, in characters.

15383       **Attribute Selection**

15384       The <termios.h> header shall define the following symbolic constants for use with *tcsetattr()*:

15385       TCSANOW       Change attributes immediately.  
 15386       TCSADRAIN      Change attributes when output has drained.  
 15387       TCSAFLUSH     Change attributes when output has drained; also flush pending input.

15388       **Line Control**

15389       The <termios.h> header shall define the following symbolic constants for use with *tcflush()*:

15390       TCIFLUSH      Flush pending input.  
 15391       TCIOFLUSH     Flush both pending input and untransmitted output.  
 15392       TCOFLUSH      Flush untransmitted output.

15393       The <termios.h> header shall define the following symbolic constants for use with *tcflow()*:

15394       TCIOFF        Transmit a STOP character, intended to suspend input data.  
 15395       TCION          Transmit a START character, intended to restart input data.  
 15396       TCOOFF        Suspend output.  
 15397       TCOON         Restart output.

15398       The <termios.h> header shall define the **pid\_t** type as described in <sys/types.h>.

15399       The following shall be declared as functions and may also be defined as macros. Function  
 15400       prototypes shall be provided.

```

15401       speed_t cfgetispeed(const struct termios *);
15402       speed_t cfgetospeed(const struct termios *);
15403       int       cfsetispeed(struct termios *, speed_t);
15404       int       cfsetospeed(struct termios *, speed_t);
15405       int       tcdrain(int);
15406       int       tcflow(int, int);
15407       int       tcflush(int, int);
15408       int       tcgetattr(int, struct termios *);
15409       pid_t     tcgetsid(int);
15410       int       tcgetwinsize(int, struct winsize *);
15411       int       tcsendbreak(int, int);
15412       int       tcsetattr(int, int, const struct termios *);
15413       int       tcsetwinsize(int, const struct winsize *);

```

15414 **APPLICATION USAGE**

15415 The following names are reserved for XSI-conformant systems to use as an extension to the  
15416 above; therefore strictly conforming applications shall not use them:

15417	CBAUD	EXTB	VDSUSP
15418	DEFECHO	FLUSHO	VLNEXT
15419	ECHOCTL	LOBLK	VREPRINT
15420	ECHOKE	PENDIN	VSTATUS
15421	ECHOPRT	SWTCH	VWERASE
15422	EXTA	VDISCARD	

15423 **RATIONALE**

15424 None.

15425 **FUTURE DIRECTIONS**

15426 None.

15427 **SEE ALSO**

15428 [<sys/types.h>](#)

15429 XSH [cfgetispeed\(\)](#), [cfgetospeed\(\)](#), [cfsetispeed\(\)](#), [cfsetospeed\(\)](#), [confstr\(\)](#), [tcdrain\(\)](#), [tcflow\(\)](#), [tcflush\(\)](#),  
15430 [tcgetattr\(\)](#), [tcgetsid\(\)](#), [tcgetwinsize\(\)](#), [tcsendbreak\(\)](#), [tcsetattr\(\)](#), [tcsetwinsize\(\)](#)

15431 XCU Chapter 11 (on page 199), [getconf](#)

15432 **CHANGE HISTORY**

15433 First released in Issue 3.

15434 Included for alignment with the ISO POSIX-1 standard.

15435 **Issue 6**

15436 The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.

15437 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are  
15438 reaffirmed.

15439 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to  
15440 ECHOKE in the APPLICATION USAGE section.

15441 **Issue 7**

15442 Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the  
15443 IXANY symbol from the XSI option to the Base.

15444 SD5-XBD-ERN-35 is applied, adding the OFDEL output mode.

15445 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
15446 for the `pid_t` type is added.

15447 **Issue 8**

15448 Austin Group Defects 1151 and 1484 are applied, adding the `winsize` structure and the  
15449 `tcgetwinsize()` and `tcsetwinsize()` functions.

15450 **NAME**

15451 tgmath.h — type-generic macros

15452 **SYNOPSIS**

15453 #include <tgmath.h>

15454 **DESCRIPTION**

15455 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15456 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15457 volume of POSIX.1-2024 defers to the ISO C standard.

15458 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define  
 15459 several type-generic macros.

15460 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**)  
 15461 or *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is  
 15462 XSI **double**. For each such function, except *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*, there shall  
 15463 be a corresponding type-generic macro. The parameters whose corresponding real type is  
 15464 **double** in the function synopsis are generic parameters. Use of the macro invokes a function  
 15465 whose corresponding real type and type domain are determined by the arguments for the  
 15466 generic parameters.

15467 Use of the macro invokes a function whose generic parameters have the corresponding real type  
 15468 determined as follows:

- 15469 • First, if any argument for generic parameters has type **long double**, the type determined is  
 15470 **long double**.
- 15471 • Otherwise, if any argument for generic parameters has type **double** or is of integer type,  
 15472 the type determined is **double**.
- 15473 • Otherwise, the type determined is **float**.

15474 For each unsuffixed function in the <math.h> header for which there is a function in the  
 15475 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic  
 15476 macro (for both functions) has the same name as the function in the <math.h> header. The  
 15477 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

	<math.h> Function	<complex.h> Function	Type-Generic Macro
15478			
15479	<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
15480	<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
15481	<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
15482	<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
15483	<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
15484	<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
15485	<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
15486	<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
15487	<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
15488	<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
15489	<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
15490	<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
15491	<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
15492	<i>log()</i>	<i>clog()</i>	<i>log()</i>
15493	<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
15494	<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
15495	<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

15496 If at least one argument for a generic parameter is complex, then use of the macro invokes a  
 15497 complex function; otherwise, use of the macro invokes a real function.

15498 For each unsuffixed function in the **<math.h>** header without a *c*-prefixed counterpart in the  
 15499 XSI **<complex.h>** header, except for *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*, the corresponding  
 15500 type-generic macro has the same name as the function. These type-generic macros are:

15501	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
15502	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
15503	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
15504	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
15505	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbln()</i>
15506	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbn()</i>
15507	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
15508	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
15509	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
15510	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

15511 If all arguments for generic parameters are real, then use of the macro invokes a real function;  
 15512 otherwise, use of the macro results in undefined behavior.

15513 For each unsuffixed function in the **<complex.h>** header that is not a *c*-prefixed counterpart to a  
 15514 function in the **<math.h>** header, the corresponding type-generic macro has the same name as  
 15515 the function. These type-generic macros are:

15516	<i>carg()</i>
15517	<i>cimag()</i>
15518	<i>conj()</i>
15519	<i>cproj()</i>
15520	<i>creal()</i>

15521 Use of the macro with any real or complex argument invokes a complex function.

15522 MXC Type-generic macros that accept complex arguments shall also accept imaginary arguments. If  
 15523 an argument is imaginary, the macro shall expand to an expression whose type is real,  
 15524 imaginary, or complex, as appropriate for the particular function: if the argument is imaginary,  
 15525 then the types of *cos()*, *cosh()*, *fabs()*, *carg()*, *cimag()*, and *creal()* shall be real; the types of *sin()*,  
 15526 *tan()*, *sinh()*, *tanh()*, *asin()*, *atan()*, *asinh()*, and *atanh()* shall be imaginary; and the types of the  
 15527 others shall be complex.

15528 Given an imaginary argument, each of the type-generic macros *cos()*, *sin()*, *tan()*, *cosh()*, *sinh()*,  
 15529 *tanh()*, *asin()*, *atan()*, *asinh()*, and *atanh()* is specified by a formula in terms of real functions:

15530	<i>cos(iy)</i>	=	<i>cosh(y)</i>
15531	<i>sin(iy)</i>	=	<i>i sinh(y)</i>
15532	<i>tan(iy)</i>	=	<i>i tanh(y)</i>
15533	<i>cosh(iy)</i>	=	<i>cos(y)</i>
15534	<i>sinh(iy)</i>	=	<i>i sin(y)</i>
15535	<i>tanh(iy)</i>	=	<i>i tan(y)</i>
15536	<i>asin(iy)</i>	=	<i>i asinh(y)</i>
15537	<i>atan(iy)</i>	=	<i>i atanh(y)</i>
15538	<i>asinh(iy)</i>	=	<i>i asin(y)</i>
15539	<i>atanh(iy)</i>	=	<i>i atan(y)</i>

15540 **APPLICATION USAGE**

15541 With the declarations:

```
15542 #include <tgmath.h>
15543 int n;
15544 float f;
15545 double d;
15546 long double ld;
15547 float complex fc;
15548 double complex dc;
15549 long double complex ldc;
```

15550 functions invoked by use of type-generic macros are shown in the following table:

	Macro	Use Invokes
15551		
15552	<i>exp(n)</i>	<i>exp(n)</i> , the function
15553	<i>acosh(f)</i>	<i>acoshf(f)</i>
15554	<i>sin(d)</i>	<i>sin(d)</i> , the function
15555	<i>atan(ld)</i>	<i>atanl(ld)</i>
15556	<i>log(fc)</i>	<i>clogf(fc)</i>
15557	<i>sqrt(dc)</i>	<i>csqrt(dc)</i>
15558	<i>pow(ldc,f)</i>	<i>cpowl(ldc, f)</i>
15559	<i>remainder(n,n)</i>	<i>remainder(n, n)</i> , the function
15560	<i>nextafter(d,f)</i>	<i>nextafter(d, f)</i> , the function
15561	<i>nexttoward(f,ld)</i>	<i>nexttowardf(f, ld)</i>
15562	<i>copysign(n,ld)</i>	<i>copysignl(n, ld)</i>
15563	<i>ceil(fc)</i>	Undefined behavior
15564	<i>rint(dc)</i>	Undefined behavior
15565	<i>fmax(ldc,ld)</i>	Undefined behavior
15566	<i>carg(n)</i>	<i>carg(n)</i> , the function
15567	<i>cproj(f)</i>	<i>cprojf(f)</i>
15568	<i>creal(d)</i>	<i>creal(d)</i> , the function
15569	<i>cimag(ld)</i>	<i>cimagl(ld)</i>
15570	<i>cabs(fc)</i>	<i>cabsf(fc)</i>
15571	<i>carg(dc)</i>	<i>carg(dc)</i> , the function
15572	<i>cproj(ldc)</i>	<i>cprojl(ldc)</i>

15573 **RATIONALE**

15574 Type-generic macros allow calling a function whose type is determined by the argument type, as  
 15575 is the case for C operators such as '+' and '\*'. For example, with a type-generic *cos()* macro,  
 15576 the expression *cos((float)x)* will have type **float**. This feature enables writing more portably  
 15577 efficient code and alleviates need for awkward casting and suffixing in the process of porting or  
 15578 adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

15579 The only arguments that affect the type resolution are the arguments corresponding to the  
 15580 parameters that have type **double** in the synopsis. Hence the type of a type-generic call to  
 15581 *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by  
 15582 the type of the first argument.

15583 The term "type-generic" was chosen over the proposed alternatives of intrinsic and overloading.  
 15584 The term is more specific than intrinsic, which already is widely used with a more general  
 15585 meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

15586 The macros are placed in their own header in order not to silently break old programs that

15587 include the **<math.h>** header; for example, with:

```
15588 printf ("%e", sin(x))
```

15589 *modf(double, double \*)* is excluded because no way was seen to make it safe without  
15590 complicating the type resolution.

15591 The implementation might, as an extension, endow appropriate ones of the macros that  
15592 POSIX.1-2024 specifies only for real arguments with the ability to invoke the complex functions.

15593 POSIX.1-2024 does not prescribe any particular implementation mechanism for generic macros.  
15594 It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for example,  
15595 could be implemented with:

```
15596 #undef sqrt  
15597 #define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

15598 Generic macros are designed for a useful level of consistency with C++ overloaded math  
15599 functions.

15600 The great majority of existing C programs are expected to be unaffected when the **<tgmath.h>**  
15601 header is included instead of the **<math.h>** or **<complex.h>** headers. Generic macros are similar  
15602 to the ISO C standard library masking macros, though the semantic types of return values differ.

15603 The ability to overload on integer as well as floating types would have been useful for some  
15604 functions; for example, *copysign()*. Overloading with different numbers of arguments would  
15605 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities  
15606 would have complicated the specification; and their natural consistent use, such as for a floating  
15607 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the  
15608 ISO/IEC 9899:1999 standard for insufficient benefit.

15609 The ISO C standard in no way limits the implementation's options for efficiency, including  
15610 inlining library functions.

#### 15611 **FUTURE DIRECTIONS**

15612 None.

#### 15613 **SEE ALSO**

15614 **<math.h>**, **<complex.h>**

15615 XSH *cabs()*, *fabs()*, *modf()*

#### 15616 **CHANGE HISTORY**

15617 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

#### 15618 **Issue 7**

15619 Austin Group Interpretation 1003.1-2001 #184 is applied, clarifying the functions for which a  
15620 corresponding type-generic macro exists with the same name as the function.

15621 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0075 [357,427] is applied.

#### 15622 **Issue 8**

15623 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.



15624 **NAME**

15625 threads.h — ISO C threads

15626 **SYNOPSIS**

15627 #include <threads.h>

15628 **DESCRIPTION**

15629 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 15630 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15631 volume of POSIX.1-2024 defers to the ISO C standard.

15632 Implementations shall not define the macro `__STDC_NO_THREADS__`, except for profile  
 15633 implementations that define `_POSIX_SUBPROFILE` (see [Section 2.1.5.1](#), on page 20) in  
 15634 `<unistd.h>`, which may define `__STDC_NO_THREADS__` and, if they do so, need not provide  
 15635 this header nor support any of its facilities.

15636 The `<threads.h>` header shall define the following macros:

15637 `thread_local` Expands to `_Thread_local`.

15638 `ONCE_FLAG_INIT` Expands to a value that can be used to initialize an object of type  
 15639 `once_flag`.

15640 `TSS_DTOR_ITERATIONS`

15641 Expands to an integer constant expression representing the maximum  
 15642 number of times that destructors will be called when a thread terminates  
 15643 and shall be suitable for use in `#if` preprocessing directives.

15644 CX If `{PTHREAD_DESTRUCTOR_ITERATIONS}` is defined in `<limits.h>`, the value of  
 15645 `TSS_DTOR_ITERATIONS` shall be equal to `{PTHREAD_DESTRUCTOR_ITERATIONS}`;  
 15646 otherwise, the value of `TSS_DTOR_ITERATIONS` shall be greater than or equal to the value of  
 15647 `{_POSIX_THREAD_DESTRUCTOR_ITERATIONS}` and shall be less than or equal to the  
 15648 maximum positive value that can be returned by a call to  
 15649 `sysconf(_SC_THREAD_DESTRUCTOR_ITERATIONS)` in any process.

15650 The `<threads.h>` header shall define the types `cnd_t`, `mtx_t`, `once_flag`, `thrd_t`, and `tss_t` as  
 15651 complete object types, the type `thrd_start_t` as the function pointer type `int (*)(void*)`, and the  
 15652 CX type `tss_dtor_t` as the function pointer type `void (*)(void*)`. The type `thrd_t` shall be defined to  
 15653 be the same type that `pthread_t` is defined to be in `<pthread.h>`.

15654 The `<threads.h>` header shall define the enumeration constants `mtx_plain`, `mtx_recursive`,  
 15655 `mtx_timed`, `thrd_busy`, `thrd_error`, `thrd_nomem`, `thrd_success` and `thrd_timedout`.

15656 The following shall be declared as functions and may also be defined as macros. Function  
 15657 prototypes shall be provided.

```

15658 void      call_once(once_flag *, void (*)(void));
15659 int       cnd_broadcast(cnd_t *);
15660 void      cnd_destroy(cnd_t *);
15661 int       cnd_init(cnd_t *);
15662 int       cnd_signal(cnd_t *);
15663 int       cnd_timedwait(cnd_t * restrict, mtx_t * restrict,
15664                        const struct timespec * restrict);
15665 int       cnd_wait(cnd_t *, mtx_t *);
15666 void      mtx_destroy(mtx_t *);
15667 int       mtx_init(mtx_t *, int);
15668 int       mtx_lock(mtx_t *);
15669 int       mtx_timedlock(mtx_t * restrict,
```

```

15670             const struct timespec * restrict);
15671     int         mtx_trylock(mtx_t *);
15672     int         mtx_unlock(mtx_t *);
15673     int         thrd_create(thrd_t *, thrd_start_t, void *);
15674     thrd_t     thrd_current(void);
15675     int         thrd_detach(thrd_t);
15676     int         thrd_equal(thrd_t, thrd_t);
15677     _Noreturn void thrd_exit(int);
15678     int         thrd_join(thrd_t, int *);
15679     int         thrd_sleep(const struct timespec *, struct timespec *);
15680     void        thrd_yield(void);
15681     int         tss_create(tss_t *, tss_dtor_t);
15682     void        tss_delete(tss_t);
15683     void        *tss_get(tss_t);
15684     int         tss_set(tss_t, void *);

```

15685 Inclusion of the **<threads.h>** header shall make symbols defined in the header **<time.h>** visible.

#### 15686 APPLICATION USAGE

15687 The **<threads.h>** header is optional in the ISO C standard but is mandated by POSIX.1-2024.  
 15688 Note however that subprofiles can choose to make this header optional (see [Section 2.1.5.1](#), on  
 15689 page 20), and therefore application portability to subprofile implementations would benefit from  
 15690 checking whether `__STDC_NO_THREADS__` is defined before inclusion of **<threads.h>**.

15691 The features provided by **<threads.h>** are not as extensive as those provided by **<pthread.h>**. It  
 15692 is present on POSIX.1 implementations in order to facilitate porting of ISO C programs that use  
 15693 it. It is recommended that applications intended for use on POSIX.1 implementations use  
 15694 **<pthread.h>** rather than **<threads.h>** even if none of the additional features are needed initially,  
 15695 to save the need to convert should the need to use them arise later in the application's lifecycle.

#### 15696 RATIONALE

15697 Although the **<threads.h>** header is optional in the ISO C standard, it is mandated by  
 15698 POSIX.1-2024 because **<pthread.h>** is mandatory and the interfaces in **<threads.h>** can easily be  
 15699 implemented as a thin wrapper for interfaces in **<pthread.h>**.

15700 The type `thrd_t` is required to be defined as the same type that `pthread_t` is defined to be in  
 15701 **<pthread.h>** because `thrd_current()` and `pthread_self()` need to return the same thread ID when  
 15702 called from the initial thread. However, these types are not fully interchangeable (that is, it is  
 15703 not always possible to pass a thread ID obtained as a `thrd_t` to a function that takes a `pthread_t`,  
 15704 and vice versa) because threads created using `thrd_create()` have a different exit status than  
 15705 `pthread_create()` threads, which is reflected in differences between the prototypes for `thrd_create()` and  
 15706 `pthread_create()`, `thrd_exit()` and `pthread_exit()`, and `thrd_join()` and `pthread_join()`; also,  
 15707 `thrd_join()` has no way to indicate that a thread was cancelled.

15708 The standard developers considered making it implementation-defined whether the types  
 15709 `cond_t`, `mtx_t` and `tss_t` are interchangeable with the corresponding types `pthread_cond_t`,  
 15710 `pthread_mutex_t` and `pthread_key_t` defined in **<pthread.h>** (that is, whether any function that  
 15711 can be called with a valid `cond_t` can also be called with a valid `pthread_cond_t`, and vice versa,  
 15712 and likewise for the other types). However, this would have meant extending `mtx_lock()` to  
 15713 provide a way for it to indicate that the owner of a mutex has terminated (equivalent to  
 15714 `[EOWNERDEAD]`). It was felt that such an extension would be invention. Although there was  
 15715 no similar concern for `cond_t` and `tss_t`, they were treated the same way as `mtx_t` for consistency.  
 15716 See also the RATIONALE for `mtx_lock()` concerning the inability of `mtx_t` to contain information  
 15717 about whether or not a mutex supports timeout if it is the same type as `pthread_mutex_t`.

15718 **FUTURE DIRECTIONS**

15719 None.

15720 **SEE ALSO**15721 [<limits.h>](#), [<pthread.h>](#), [<time.h>](#)

15722 XSH Section 2.9, [call\\_once\(\)](#), [cnd\\_broadcast\(\)](#), [cnd\\_destroy\(\)](#), [cnd\\_timedwait\(\)](#), [mtx\\_destroy\(\)](#),  
15723 [mtx\\_lock\(\)](#), [sysconf\(\)](#), [thrd\\_create\(\)](#), [thrd\\_current\(\)](#), [thrd\\_detach\(\)](#), [thrd\\_equal\(\)](#), [thrd\\_exit\(\)](#),  
15724 [thrd\\_join\(\)](#), [thrd\\_sleep\(\)](#), [thrd\\_yield\(\)](#), [tss\\_create\(\)](#), [tss\\_delete\(\)](#), [tss\\_get\(\)](#)

15725 **CHANGE HISTORY**

15726 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

15727 **NAME**

15728 time.h — time types

15729 **SYNOPSIS**

15730 #include &lt;time.h&gt;

15731 **DESCRIPTION**

15732 CX Some of the functionality described on this reference page extends the ISO C standard.  
 15733 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 15734 enable the visibility of these symbols in this header.

15735 The **<time.h>** header shall define the **clock\_t**, **size\_t**, **time\_t**, types as described in  
 15736 **<sys/types.h>**.

15737 CX The **<time.h>** header shall define the **clockid\_t** and **timer\_t** types as described in **<sys/types.h>**.

15738 The **<time.h>** header shall define the **locale\_t** type as described in **<locale.h>**.

15739 CPT The **<time.h>** header shall define the **pid\_t** type as described in **<sys/types.h>**.

15740 CX The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which  
 15741 are described in the **<signal.h>** header.

15742 The **<time.h>** header shall declare the **tm** structure, which shall include at least the following  
 15743 members:

15744	int	tm_sec	Seconds [0,60].
15745	int	tm_min	Minutes [0,59].
15746	int	tm_hour	Hour [0,23].
15747	int	tm_mday	Day of month [1,31].
15748	int	tm_mon	Month of year [0,11].
15749	int	tm_year	Years since 1900.
15750	int	tm_wday	Day of week [0,6] (Sunday =0).
15751	int	tm_yday	Day of year [0,365].
15752	int	tm_isdst	Daylight Saving flag.
15753	long	tm_gmtoff	Seconds east of UTC.
15754	const char *	tm_zone	Timezone abbreviation.

15755 When *tm\_isdst* is set by an interface defined in this standard, its value shall be positive if  
 15756 CX Daylight Saving Time (DST) is in effect and 0 if DST is not in effect. It shall not be set to a  
 15757 negative value by any interface defined in this standard. When *tm\_isdst* is passed to the *mktime()*  
 15758 function, it specifies how *mktime()* is to handle DST when calculating the time since the Epoch  
 15759 value; see *mktime()*.

15760 CX If the value of *tm\_zone* is accessed after the value of *TZ* is subsequently modified, and the  
 15761 *tm\_zone* value was not set by a call to *gmtime()* or *gmtime\_r()*, the behavior is undefined.

15762 The **<time.h>** header shall declare the **timespec** structure, which shall include at least the  
 15763 following members:

15764	time_t	tv_sec	Whole seconds.
15765	long	tv_nsec	Nanoseconds [0, 999 999 999].

15766 CX The **<time.h>** header shall also declare the **itimerspec** structure, which shall include at least the  
 15767 following members:

15768	struct timespec	it_interval	Timer period.
15769	struct timespec	it_value	Timer expiration.

15770 The <time.h> header shall define the following macros:

15771 NULL As described in <stddef.h>.

15772 CLOCKS\_PER\_SEC A number used to convert the value returned by the *clock()* function into  
 15773 XSI seconds. The value shall be an expression with type **clock\_t**. The value of  
 15774 CLOCKS\_PER\_SEC shall be 1 million on XSI-conformant systems.  
 15775 However, it may be variable on other systems, and it should not be  
 15776 assumed that CLOCKS\_PER\_SEC is a compile-time constant.

15777 TIME\_UTC An integer constant greater than 0 that designates the UTC time base in  
 15778 calls to *timespec\_get()*. The value shall be suitable for use in **#if**  
 15779 preprocessing directives.

15780 CX The <time.h> header shall define the following symbolic constants. The values shall have a type  
 15781 that is assignment-compatible with **clockid\_t**.

15782 CX CLOCK\_MONOTONIC  
 15783 The identifier for the system-wide monotonic clock, which is defined as a  
 15784 clock measuring real time, whose value cannot be set via *clock\_settime()*  
 15785 and which cannot have negative clock jumps. The maximum possible  
 15786 clock jump shall be implementation-defined.

15787 CPT CLOCK\_PROCESS\_CPUTIME\_ID  
 15788 The identifier of the CPU-time clock associated with the process making a  
 15789 *clock\*()* or *timer\*()* function call.

15790 CX CLOCK\_REALTIME The identifier of the system-wide clock measuring real time.

15791 TCT CLOCK\_THREAD\_CPUTIME\_ID  
 15792 The identifier of the CPU-time clock associated with the thread making a  
 15793 *clock\*()* or *timer\*()* function call.

15794 CX The <time.h> header shall define the following symbolic constant:

15795 TIMER\_ABSTIME Flag indicating time is absolute. For functions taking timer objects, this  
 15796 refers to the clock associated with the timer.

15797 XSI The <time.h> header shall provide a declaration or definition for *getdate\_err*. The *getdate\_err*  
 15798 symbol shall expand to an expression of type **int**. It is unspecified whether *getdate\_err* is a macro  
 15799 or an identifier declared with external linkage, and whether or not it is a modifiable lvalue. If a  
 15800 macro definition is suppressed in order to access an actual object, or a program defines an  
 15801 identifier with the name *getdate\_err*, the behavior is undefined.

15802 The following shall be declared as functions and may also be defined as macros. Function  
 15803 prototypes shall be provided.

15804 OB `char *asctime(const struct tm *);`  
 15805 `clock_t clock(void);`  
 15806 CPT `int clock_getcpuclockid(pid_t, clockid_t *);`  
 15807 CX `int clock_getres(clockid_t, struct timespec *);`  
 15808 `int clock_gettime(clockid_t, struct timespec *);`  
 15809 `int clock_nanosleep(clockid_t, int, const struct timespec *,`  
 15810 `struct timespec *);`  
 15811 `int clock_settime(clockid_t, const struct timespec *);`  
 15812 OB `char *ctime(const time_t *);`  
 15813 `double difftime(time_t, time_t);`

```

15814 XSI struct tm *getdate(const char *);
15815 struct tm *gmtime(const time_t *);
15816 CX struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
15817 struct tm *localtime(const time_t *);
15818 CX struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
15819 time_t mktime(struct tm *);
15820 CX int nanosleep(const struct timespec *, struct timespec *);
15821 size_t strftime(char *restrict, size_t, const char *restrict,
15822 const struct tm *restrict);
15823 CX size_t strftime_l(char *restrict, size_t, const char *restrict,
15824 const struct tm *restrict, locale_t);
15825 XSI char *strptime(const char *restrict, const char *restrict,
15826 struct tm *restrict);
15827 time_t time(time_t *);
15828 CX int timer_create(clockid_t, struct sigevent *restrict,
15829 timer_t *restrict);
15830 int timer_delete(timer_t);
15831 int timer_getoverrun(timer_t);
15832 int timer_gettime(timer_t, struct itimerspec *);
15833 int timer_settime(timer_t, int, const struct itimerspec *restrict,
15834 struct itimerspec *restrict);
15835 int timespec_get(struct timespec *, int);
15836 CX void tzset(void);

```

15837 The **<time.h>** header shall declare the following as variables:

```

15838 XSI extern int daylight;
15839 extern long timezone;
15840 CX extern char *tzname[];

```

15841 CX Inclusion of the **<time.h>** header may make visible all symbols from the **<signal.h>** header.

## 15842 APPLICATION USAGE

15843 The range [0,60] for *tm\_sec* allows for the occasional leap second.

15844 *tm\_year* is a signed value; therefore, years before 1900 may be represented.

15845 To obtain the number of clock ticks per second returned by the *times()* function, applications  
 15846 should call *sysconf(\_SC\_CLK\_TCK)*.

## 15847 RATIONALE

15848 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of  
 15849 UTC does not permit double leap seconds, so all mention of double leap seconds has been  
 15850 removed, and the range shortened from the former [0,61] seconds seen in earlier versions of this  
 15851 standard.

## 15852 FUTURE DIRECTIONS

15853 None.

## 15854 SEE ALSO

15855 [<locale.h>](#), [<signal.h>](#), [<stddef.h>](#), [<sys/types.h>](#)

15856 XSH Section 2.2 (on page 496), *asctime()*, *clock()*, *clock\_getcpuclockid()*, *clock\_getres()*,  
 15857 *clock\_nanosleep()*, *ctime()*, *difftime()*, *futimens()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*,  
 15858 *mq\_receive()*, *mq\_send()*, *nanosleep()*, *pthread\_getcpuclockid()*, *pthread\_mutex\_clocklock()*,

15859 *pthread\_rwlock\_clockrdlock(), pthread\_rwlock\_clockwrlock(), sem\_clockwait(), strftime(), strptime(),*  
 15860 *sysconf(), time(), timer\_create(), timer\_delete(), timer\_getoverrun(), timespec\_get(), tzset()*

## 15861 CHANGE HISTORY

15862 First released in Issue 1. Derived from Issue 1 of the SVID.

### 15863 Issue 5

15864 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 15865 Threads Extension.

### 15866 Issue 6

15867 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid\_t**  
 15868 and **timer\_t** have been described.

15869 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 15870 • The POSIX timer-related functions are marked as part of the Timers option.

15871 The symbolic name CLK\_TCK is removed. Application usage is added describing how its  
 15872 equivalent functionality can be obtained using *sysconf()*.

15873 The *clock\_getcpuclockid()* function and manifest constants CLOCK\_PROCESS\_CPUTIME\_ID and  
 15874 CLOCK\_THREAD\_CPUTIME\_ID are added for alignment with IEEE Std 1003.1d-1999.

15875 The manifest constant CLOCK\_MONOTONIC and the *clock\_nanosleep()* function are added for  
 15876 alignment with IEEE Std 1003.1j-2000.

15877 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 15878 • The range for seconds is changed from [0,61] to [0,60].
- 15879 • The **restrict** keyword is added to the prototypes for *asctime\_r()*, *gmtime\_r()*, *localtime\_r()*,  
 15880 *strftime()*, *strptime()*, *timer\_create()*, and *timer\_settime()*.

15881 IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the  
 15882 <signal.h> header may be made visible when the <time.h> header is included.

15883 Extensions beyond the ISO C standard are marked.

### 15884 Issue 7

15885 Austin Group Interpretation 1003.1-2001 #111 is applied.

15886 SD5-XBD-ERN-74 is applied.

15887 The *strftime\_1()* function is added from The Open Group Technical Standard, 2006, Extended API  
 15888 Set Part 4.

15889 Functionality relating to the Timers option is moved to the Base.

15890 This reference page is clarified with respect to macros and symbolic constants, and declarations  
 15891 for the **locale\_t** and **pid\_t** types and the **sigevent** structure are added.

15892 The description of the *getdate\_err* value is expanded.

15893 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0076 [212] and XBD/TC1-2008/0077  
 15894 [212] are applied.

### 15895 Issue 8

15896 Austin Group Defect 1253 is applied, changing “Daylight Savings” to “Daylight Saving”.

15897 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.

15898 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

- 15899 Austin Group Defect 1410 is applied, removing the *asctime\_r()* and *ctime\_r()* functions.
- 15900 Austin Group Defect 1533 is applied, adding *tm\_gmtoff* and *tm\_zone* to the **tm** structure.
- 15901 Austin Group Defect 1597 is applied, changing *clock()* to *clock\*()* in the descriptions of
- 15902 CLOCK\_PROCESS\_CPUTIME\_ID and CLOCK\_THREAD\_CPUTIME\_ID.
- 15903 Austin Group Defect 1614 is applied, clarifying the requirements for the *tm\_isdst* member of the
- 15904 **tm** structure.



15905 **NAME**

15906           uchar.h — Unicode character handling

15907 **SYNOPSIS**

15908           #include &lt;uchar.h&gt;

15909 **DESCRIPTION**

15910 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 15911 conflict between the requirements described here and the ISO C standard is unintentional. This  
 15912 volume of POSIX.1-2024 defers to the ISO C standard.

15913       The &lt;uchar.h&gt; header shall define the following types:

15914       **mbstate\_t**   As described in <wchar.h>.15915       **size\_t**       As described in <stddef.h>.15916       **char16\_t**   The same type as **uint\_least16\_t**, described in <stdint.h>.15917       **char32\_t**   The same type as **uint\_least32\_t**, described in <stdint.h>.

15918       The following shall be declared as functions and may also be defined as macros. Function  
 15919 prototypes shall be provided.

```
15920       size_t  c16rtomb(char *restrict, char16_t, mbstate_t *restrict);
15921       size_t  c32rtomb(char *restrict, char32_t, mbstate_t *restrict);
15922       size_t  mbrtoc16(char16_t *restrict, const char *restrict, size_t,
15923                        mbstate_t *restrict);
15924       size_t  mbrtoc32(char32_t *restrict, const char *restrict, size_t,
15925                        mbstate_t *restrict);
```

15926 CX       Inclusion of the <uchar.h> header may make visible all symbols from the headers <stddef.h>,  
 15927 <stdint.h>, and <wchar.h>.

15928 **APPLICATION USAGE**

15929       None.

15930 **RATIONALE**

15931       None.

15932 **FUTURE DIRECTIONS**

15933       None.

15934 **SEE ALSO**

15935       &lt;stddef.h&gt;, &lt;stdint.h&gt;, &lt;wchar.h&gt;

15936       XSH *c16rtomb()*, *mbrtoc16()*15937 **CHANGE HISTORY**

15938       First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

**15939 NAME**

15940 unistd.h — standard symbolic constants and types

**15941 SYNOPSIS**

15942 #include <unistd.h>

**15943 DESCRIPTION**

15944 The **<unistd.h>** header defines miscellaneous symbolic constants and types, and declares  
 15945 miscellaneous functions. The actual values of the constants are unspecified except as shown. The  
 15946 contents of this header are shown below.

**15947 Version Test Macros**

15948 The **<unistd.h>** header shall define the following symbolic constants. The values shall be  
 15949 suitable for use in **#if** preprocessing directives.

**15950 \_POSIX\_VERSION**

15951 Integer value indicating version of this standard (C-language binding) to which the  
 15952 implementation conforms. For implementations conforming to POSIX.1-2024, the value  
 15953 shall be 202405L.

**15954 \_POSIX2\_VERSION**

15955 Integer value indicating version of the Shell and Utilities volume of POSIX.1 to which the  
 15956 implementation conforms. For implementations conforming to POSIX.1-2024, the value  
 15957 shall be 202405L. For profile implementations that define `_POSIX_SUBPROFILE` (see  
 15958 [Section 2.1.5.1](#)) in **<unistd.h>**, `_POSIX2_VERSION` may be left undefined or be defined with  
 15959 the value `-1` to indicate that the Shell and Utilities volume of POSIX.1 is not supported. In  
 15960 this case, a call to `sysconf(_SC_2_VERSION)` shall return either 202405L or `-1` indicating that  
 15961 the Shell and Utilities volume of POSIX.1 is or is not, respectively, supported at runtime.

15962 The **<unistd.h>** header shall define the following symbolic constant only if the implementation  
 15963 supports the XSI option; see [Section 2.1.4](#) (on page 19). If defined, its value shall be suitable for  
 15964 use in **#if** preprocessing directives.

**15965 XSI \_XOPEN\_VERSION**

15966 Integer value indicating version of the X/Open Portability Guide to which the  
 15967 implementation conforms. The value shall be 800.

**15968 Constants for Options and Option Groups**

15969 The following symbolic constants, if defined in **<unistd.h>**, shall have a value of `-1`, `0`, or  
 15970 greater, unless otherwise specified below. For profile implementations that define  
 15971 `_POSIX_SUBPROFILE` (see [Section 2.1.5.1](#)) in **<unistd.h>**, constants described below as always  
 15972 having a value greater than zero need not be defined and, if defined, may have a value of `-1`, `0`,  
 15973 or greater. The values shall be suitable for use in **#if** preprocessing directives.

15974 If a symbolic constant is not defined or is defined with the value `-1`, the option is not supported  
 15975 for compilation. If it is defined with a value greater than zero, the option shall always be  
 15976 supported when the application is executed. If it is defined with the value zero, the option shall  
 15977 be supported for compilation and might or might not be supported at runtime. See [Section 2.1.6](#)  
 15978 (on page 25) for further information about the conformance requirements of these three  
 15979 categories of support.

**15980 ADV \_POSIX\_ADVISORY\_INFO**

15981 The implementation supports the Advisory Information option. If this symbol is defined in  
 15982 **<unistd.h>**, it shall be defined to be `-1`, `0`, or 202405L. The value of this symbol reported by  
 15983 `sysconf()` shall either be `-1` or 202405L.

15984		<code>_POSIX_ASYNCHRONOUS_IO</code>
15985		The implementation supports asynchronous input and output. This symbol shall always be
15986		set to the value 202405L.
15987		<code>_POSIX_BARRIERS</code>
15988		The implementation supports barriers. This symbol shall always be set to the value
15989		202405L.
15990		<code>_POSIX_CHOWN_RESTRICTED</code>
15991		The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and
15992		to changing the group ID of a file only to the effective group ID of the process or to one of
15993		its supplementary group IDs. This symbol shall be defined with a value other than -1.
15994		<code>_POSIX_CLOCK_SELECTION</code>
15995		The implementation supports clock selection. This symbol shall always be set to the value
15996		202405L.
15997	CPT	<code>_POSIX_CPUTIME</code>
15998		The implementation supports the Process CPU-Time Clocks option. If this symbol is
15999		defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol
16000		reported by <i>sysconf()</i> shall either be -1 or 202405L.
16001	DC	<code>_POSIX_DEVICE_CONTROL</code>
16002		The implementation supports the device control option. If this symbol is defined in
16003		<unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by
16004		<i>sysconf()</i> shall either be -1 or 202405L.
16005	FSC	<code>_POSIX_FSYNC</code>
16006		The implementation supports the File Synchronization option. If this symbol is defined in
16007		<unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by
16008		<i>sysconf()</i> shall either be -1 or 202405L.
16009	IP6	<code>_POSIX_IPV6</code>
16010		The implementation supports the IPv6 option. If this symbol is defined in <unistd.h>, it
16011		shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall
16012		either be -1 or 202405L.
16013		<code>_POSIX_JOB_CONTROL</code>
16014		The implementation supports job control. This symbol shall always be set to a value greater
16015		than zero.
16016		<code>_POSIX_MAPPED_FILES</code>
16017		The implementation supports memory mapped Files. This symbol shall always be set to the
16018		value 202405L.
16019	ML	<code>_POSIX_MEMLOCK</code>
16020		The implementation supports the Process Memory Locking option. If this symbol is defined
16021		in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported
16022		by <i>sysconf()</i> shall either be -1 or 202405L.
16023	MLR	<code>_POSIX_MEMLOCK_RANGE</code>
16024		The implementation supports the Range Memory Locking option. If this symbol is defined
16025		in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported
16026		by <i>sysconf()</i> shall either be -1 or 202405L.
16027		<code>_POSIX_MEMORY_PROTECTION</code>
16028		The implementation supports memory protection. This symbol shall always be set to the
16029		value 202405L.

16030 MSG **\_POSIX\_MESSAGE\_PASSING**  
 16031 The implementation supports the Message Passing option. If this symbol is defined in  
 16032 <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by  
 16033 *sysconf()* shall either be -1 or 202405L.

16034 **\_POSIX\_MONOTONIC\_CLOCK**  
 16035 The implementation supports a monotonic clock. This symbol shall always be set to the  
 16036 value 202405L.

16037 **\_POSIX\_NO\_TRUNC**  
 16038 Pathname components longer than {NAME\_MAX} generate an error. This symbol shall be  
 16039 defined with a value other than -1.

16040 PIO **\_POSIX\_PRIORITIZED\_IO**  
 16041 The implementation supports the Prioritized Input and Output option. If this symbol is  
 16042 defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol  
 16043 reported by *sysconf()* shall either be -1 or 202405L.

16044 PS **\_POSIX\_PRIORITY\_SCHEDULING**  
 16045 The implementation supports the Process Scheduling option. If this symbol is defined in  
 16046 <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by  
 16047 *sysconf()* shall either be -1 or 202405L.

16048 RS **\_POSIX\_RAW\_SOCKETS**  
 16049 The implementation supports the Raw Sockets option. If this symbol is defined in  
 16050 <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by  
 16051 *sysconf()* shall either be -1 or 202405L.

16052 **\_POSIX\_READER\_WRITER\_LOCKS**  
 16053 The implementation supports read-write locks. This symbol shall always be set to the value  
 16054 202405L.

16055 **\_POSIX\_REALTIME\_SIGNALS**  
 16056 The implementation supports realtime signals. This symbol shall always be set to the value  
 16057 202405L.

16058 **\_POSIX\_REGEX**  
 16059 The implementation supports the Regular Expression Handling option. This symbol shall  
 16060 always be set to a value greater than zero.

16061 **\_POSIX\_SAVED\_IDS**  
 16062 Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always be  
 16063 set to a value greater than zero.

16064 **\_POSIX\_SEMAPHORES**  
 16065 The implementation supports semaphores. This symbol shall always be set to the value  
 16066 202405L.

16067 SHM **\_POSIX\_SHARED\_MEMORY\_OBJECTS**  
 16068 The implementation supports the Shared Memory Objects option. If this symbol is defined  
 16069 in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported  
 16070 by *sysconf()* shall either be -1 or 202405L.

16071 **\_POSIX\_SHELL**  
 16072 The implementation supports the POSIX shell. This symbol shall always be set to a value  
 16073 greater than zero.

16074	SPN	<b>_POSIX_SPAWN</b>	The implementation supports the Spawn option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16075			
16076			
16077			
16078		<b>_POSIX_SPIN_LOCKS</b>	The implementation supports spin locks. This symbol shall always be set to the value 202405L.
16079			
16080			
16081	SS	<b>_POSIX_SPORADIC_SERVER</b>	The implementation supports the Process Sporadic Server option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16082			
16083			
16084			
16085	SIO	<b>_POSIX_SYNCHRONIZED_IO</b>	The implementation supports the Synchronized Input and Output option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16086			
16087			
16088			
16089	TSA	<b>_POSIX_THREAD_ATTR_STACKADDR</b>	The implementation supports the Thread Stack Address Attribute option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16090			
16091			
16092			
16093	TSS	<b>_POSIX_THREAD_ATTR_STACKSIZE</b>	The implementation supports the Thread Stack Size Attribute option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16094			
16095			
16096			
16097	TCT	<b>_POSIX_THREAD_CPU_TIME</b>	The implementation supports the Thread CPU-Time Clocks option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16098			
16099			
16100			
16101	TPI	<b>_POSIX_THREAD_PRIO_INHERIT</b>	The implementation supports the Non-Robust Mutex Priority Inheritance option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16102			
16103			
16104			
16105	TPP	<b>_POSIX_THREAD_PRIO_PROTECT</b>	The implementation supports the Non-Robust Mutex Priority Protection option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16106			
16107			
16108			
16109	TPS	<b>_POSIX_THREAD_PRIORITY_SCHEDULING</b>	The implementation supports the Thread Execution Scheduling option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16110			
16111			
16112			
16113	TSH	<b>_POSIX_THREAD_PROCESS_SHARED</b>	The implementation supports the Thread Process-Shared Synchronization option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16114			
16115			
16116			
16117	RPI	<b>_POSIX_THREAD_ROBUST_PRIO_INHERIT</b>	The implementation supports the Robust Mutex Priority Inheritance option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol
16118			
16119			

16120 reported by *sysconf()* shall either be `-1` or `202405L`.

16121 RPP `_POSIX_THREAD_ROBUST_PRIO_PROTECT`

16122 The implementation supports the Robust Mutex Priority Protection option. If this symbol is

16123 defined in `<unistd.h>`, it shall be defined to be `-1`, `0`, or `202405L`. The value of this symbol

16124 reported by *sysconf()* shall either be `-1` or `202405L`.

16125 `_POSIX_THREAD_SAFE_FUNCTIONS`

16126 The implementation supports thread-safe functions. This symbol shall always be set to the

16127 value `202405L`.

16128 TSP `_POSIX_THREAD_SPORADIC_SERVER`

16129 The implementation supports the Thread Sporadic Server option. If this symbol is defined

16130 in `<unistd.h>`, it shall be defined to be `-1`, `0`, or `202405L`. The value of this symbol reported

16131 by *sysconf()* shall either be `-1` or `202405L`.

16132 `_POSIX_THREADS`

16133 The implementation supports threads. This symbol shall always be set to the value

16134 `202405L`.

16135 `_POSIX_TIMEOUTS`

16136 The implementation supports timeouts. This symbol shall always be set to the value

16137 `202405L`.

16138 `_POSIX_TIMERS`

16139 The implementation supports timers. This symbol shall always be set to the value `202405L`.

16140 TYM `_POSIX_TYPED_MEMORY_OBJECTS`

16141 The implementation supports the Typed Memory Objects option. If this symbol is defined

16142 in `<unistd.h>`, it shall be defined to be `-1`, `0`, or `202405L`. The value of this symbol reported

16143 by *sysconf()* shall either be `-1` or `202405L`.

16144 OB `_POSIX_V7_ILP32_OFF32`

16145 The implementation provides a C-language compilation environment with 32-bit `int`, `long`,

16146 `off_t`, and all pointer types.

16147 OB `_POSIX_V7_ILP32_OFFBIG`

16148 The implementation provides a C-language compilation environment with 32-bit `int`, `long`,

16149 and all pointer types, and an `off_t` type using at least 64 bits.

16150 OB `_POSIX_V7_LP64_OFF64`

16151 The implementation provides a C-language compilation environment with a 32-bit `int` type

16152 and 64-bit `long`, `off_t`, and all pointer types.

16153 OB `_POSIX_V7_LP64_OFFBIG`

16154 The implementation provides a C-language compilation environment with an `int` type

16155 using at least 32 bits and `long`, `off_t`, and all pointer types using at least 64 bits.

16156 `_POSIX_V8_ILP32_OFF32`

16157 The implementation provides a C-language compilation environment with 32-bit `int`, `long`,

16158 `off_t`, and all pointer types.

16159 `_POSIX_V8_ILP32_OFFBIG`

16160 The implementation provides a C-language compilation environment with 32-bit `int`, `long`,

16161 and all pointer types, and an `off_t` type using at least 64 bits.

16162 `_POSIX_V8_LP64_OFF64`

16163 The implementation provides a C-language compilation environment with a 32-bit `int` type

16164 and 64-bit `long`, `off_t`, and all pointer types.

16165		<code>_POSIX_V8_LPBIG_OFFBIG</code>	
16166			The implementation provides a C-language compilation environment with an <b>int</b> type using at least 32 bits and <b>long</b> , <b>off_t</b> , and all pointer types using at least 64 bits.
16167			
16168		<code>_POSIX2_C_BIND</code>	
16169			The implementation supports the C-Language Binding option. This symbol shall always have the value 202405L.
16170			
16171	CD	<code>_POSIX2_C_DEV</code>	
16172			The implementation supports the C-Language Development Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16173			
16174			
16175		<code>_POSIX2_CHAR_TERM</code>	
16176			The implementation supports the Terminal Characteristics option. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or a value greater than zero.
16177			
16178	FR	<code>_POSIX2_FORT_RUN</code>	
16179			The implementation supports the FORTRAN Runtime Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16180			
16181			
16182		<code>_POSIX2_LOCALEDEF</code>	
16183			The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16184			
16185			
16186	SD	<code>_POSIX2_SW_DEV</code>	
16187			The implementation supports the Software Development Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16188			
16189			
16190	UP	<code>_POSIX2_UPE</code>	
16191			The implementation supports the User Portability Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 202405L.
16192			
16193			
16194	XSI	<code>_XOPEN_CRYPT</code>	
16195			The implementation supports the X/Open Encryption Option Group.
16196		<code>_XOPEN_ENH_I18N</code>	
16197			The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option Group. This symbol shall always be set to a value other than -1.
16198			
16199		<code>_XOPEN_REALTIME</code>	
16200			The implementation supports the X/Open Realtime Option Group.
16201		<code>_XOPEN_REALTIME_THREADS</code>	
16202			The implementation supports the X/Open Realtime Threads Option Group.
16203		<code>_XOPEN_SHM</code>	
16204			The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This symbol shall always be set to a value other than -1.
16205			
16206		<code>_XOPEN_UNIX</code>	
16207			The implementation supports the XSI option.

16208 UU **\_XOPEN\_UUCP**  
 16209 The implementation supports the UUCP Utilities option. If this symbol is defined in  
 16210 <unistd.h>, it shall be defined to be -1, 0, or 202405L. The value of this symbol reported by  
 16211 *sysconf()* shall be either -1 or 202405L.

16212 **Execution-Time Symbolic Constants**

16213 If any of the following symbolic constants are not defined in the <unistd.h> header, the value  
 16214 shall vary depending on the file to which it is applied. If defined, they shall have values suitable  
 16215 for use in #if preprocessing directives.

16216 If any of the following symbolic constants are defined to have value -1 in the <unistd.h> header,  
 16217 the implementation shall not provide the option on any file; if any are defined to have a value  
 16218 other than -1 in the <unistd.h> header, the implementation shall provide the option on all  
 16219 applicable files.

16220 All of the following values, whether defined as symbolic constants in <unistd.h> or not, may be  
 16221 queried with respect to a specific file using the *pathconf()* or *fpathconf()* functions:

16222 **\_POSIX\_ASYNC\_IO**  
 16223 Asynchronous input or output operations may be performed for the associated file.

16224 **\_POSIX\_FALLOC**  
 16225 The *posix\_fallocate()* function is supported by the associated file.

16226 **\_POSIX\_PRIO\_IO**  
 16227 Prioritized input or output operations may be performed for the associated file.

16228 **\_POSIX\_SYNC\_IO**  
 16229 Synchronized input or output operations may be performed for the associated file.

16230 If the following symbolic constants are defined in the <unistd.h> header, they apply to files and  
 16231 all paths in all file systems on the implementation:

16232 **\_POSIX\_TIMESTAMP\_RESOLUTION**  
 16233 The resolution in nanoseconds for all file timestamps.

16234 **\_POSIX2\_SYMLINKS**  
 16235 Symbolic links can be created.

16236 **Constants for Functions**

16237 The <unistd.h> header shall define NULL as described in <stddef.h>.

16238 The <unistd.h> header shall define the symbolic constants O\_CLOEXEC and O\_CLOFORK as  
 16239 described in <fcntl.h>.

16240 The <unistd.h> header shall define the following symbolic constants for use with the *access()*  
 16241 function. The values shall be suitable for use in #if preprocessing directives.

16242 **F\_OK** Test for existence of file.

16243 **R\_OK** Test for read permission.

16244 **W\_OK** Test for write permission.

16245 **X\_OK** Test for execute (search) permission.

16246 The constants F\_OK, R\_OK, W\_OK, and X\_OK and the expressions R\_OK | W\_OK, R\_OK | X\_OK,  
 16247 and R\_OK | W\_OK | X\_OK shall all have distinct values.



16248 The <unistd.h> header shall define the following symbolic constants for the *confstr()* function:

16249 `_CS_PATH`  
 16250 This is the value for the *PATH* environment variable that finds all of the standard utilities  
 16251 that are provided in a manner accessible via the *exec* family of functions.

16252 `_CS_POSIX_V8_ILP32_OFF32_CFLAGS`  
 16253 If *sysconf*(`_SC_V8_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.  
 16254 Otherwise, this value is the set of initial options to be given to the *c17* utility to build an  
 16255 application using a programming model with 32-bit **int**, **long**, **off\_t**, and all pointer types.

16256 `_CS_POSIX_V8_ILP32_OFF32_LDFLAGS`  
 16257 If *sysconf*(`_SC_V8_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.  
 16258 Otherwise, this value is the set of final options to be given to the *c17* utility to build an  
 16259 application using a programming model with 32-bit **int**, **long**, **off\_t**, and all pointer types.

16260 `_CS_POSIX_V8_ILP32_OFF32_LIBS`  
 16261 If *sysconf*(`_SC_V8_ILP32_OFF32`) returns `-1`, the meaning of this value is unspecified.  
 16262 Otherwise, this value is the set of libraries to be given to the *c17* utility to build an  
 16263 application using a programming model with 32-bit **int**, **long**, **off\_t**, and all pointer types.

16264 `_CS_POSIX_V8_ILP32_OFFBIG_CFLAGS`  
 16265 If *sysconf*(`_SC_V8_ILP32_OFFBIG`) returns `-1`, the meaning of this value is unspecified.  
 16266 Otherwise, this value is the set of initial options to be given to the *c17* utility to build an  
 16267 application using a programming model with 32-bit **int**, **long**, and all pointer types, and an  
 16268 **off\_t** type using at least 64 bits.

16269 `_CS_POSIX_V8_ILP32_OFFBIG_LDFLAGS`  
 16270 If *sysconf*(`_SC_V8_ILP32_OFFBIG`) returns `-1`, the meaning of this value is unspecified.  
 16271 Otherwise, this value is the set of final options to be given to the *c17* utility to build an  
 16272 application using a programming model with 32-bit **int**, **long**, and all pointer types, and an  
 16273 **off\_t** type using at least 64 bits.

16274 `_CS_POSIX_V8_ILP32_OFFBIG_LIBS`  
 16275 If *sysconf*(`_SC_V8_ILP32_OFFBIG`) returns `-1`, the meaning of this value is unspecified.  
 16276 Otherwise, this value is the set of libraries to be given to the *c17* utility to build an  
 16277 application using a programming model with 32-bit **int**, **long**, and all pointer types, and an  
 16278 **off\_t** type using at least 64 bits.

16279 `_CS_POSIX_V8_LP64_OFF64_CFLAGS`  
 16280 If *sysconf*(`_SC_V8_LP64_OFF64`) returns `-1`, the meaning of this value is unspecified.  
 16281 Otherwise, this value is the set of initial options to be given to the *c17* utility to build an  
 16282 application using a programming model with a 32-bit **int** type and 64-bit **long**, **off\_t**, and all  
 16283 pointer types.

16284 `_CS_POSIX_V8_LP64_OFF64_LDFLAGS`  
 16285 If *sysconf*(`_SC_V8_LP64_OFF64`) returns `-1`, the meaning of this value is unspecified.  
 16286 Otherwise, this value is the set of final options to be given to the *c17* utility to build an  
 16287 application using a programming model with a 32-bit **int** type and 64-bit **long**, **off\_t**, and all  
 16288 pointer types.

16289 `_CS_POSIX_V8_LP64_OFF64_LIBS`  
 16290 If *sysconf*(`_SC_V8_LP64_OFF64`) returns `-1`, the meaning of this value is unspecified.  
 16291 Otherwise, this value is the set of libraries to be given to the *c17* utility to build an  
 16292 application using a programming model with a 32-bit **int** type and 64-bit **long**, **off\_t**, and all  
 16293 pointer types.

16294 `_CS_POSIX_V8_LPBIG_OFFBIG_CFLAGS`  
 16295 If `sysconf(_SC_V8_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 16296 Otherwise, this value is the set of initial options to be given to the `c17` utility to build an  
 16297 application using a programming model with an `int` type using at least 32 bits and `long`,  
 16298 `off_t`, and all pointer types using at least 64 bits.

16299 `_CS_POSIX_V8_LPBIG_OFFBIG_LDFLAGS`  
 16300 If `sysconf(_SC_V8_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 16301 Otherwise, this value is the set of final options to be given to the `c17` utility to build an  
 16302 application using a programming model with an `int` type using at least 32 bits and `long`,  
 16303 `off_t`, and all pointer types using at least 64 bits.

16304 `_CS_POSIX_V8_LPBIG_OFFBIG_LIBS`  
 16305 If `sysconf(_SC_V8_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.  
 16306 Otherwise, this value is the set of libraries to be given to the `c17` utility to build an  
 16307 application using a programming model with an `int` type using at least 32 bits and `long`,  
 16308 `off_t`, and all pointer types using at least 64 bits.

16309 `_CS_POSIX_V8_THREADS_CFLAGS`  
 16310 This value is the set of initial options to be given to the `c17` utility to build a multi-threaded  
 16311 application. These flags are in addition to those associated with any of the other  
 16312 `_CS_POSIX_V8*_CFLAGS` values used to specify particular type size programming  
 16313 environments.

16314 `_CS_POSIX_V8_THREADS_LDFLAGS`  
 16315 This value is the set of final options to be given to the `c17` utility to build a multi-threaded  
 16316 application. These flags are in addition to those associated with any of the other  
 16317 `_CS_POSIX_V8*_LDFLAGS` values used to specify particular type size programming  
 16318 environments.

16319 `_CS_POSIX_V8_WIDTH_RESTRICTED_ENVS`  
 16320 This value is a <newline>-separated list of names of programming environments supported  
 16321 by the implementation in which the widths of the `blksize_t`, `cc_t`, `mode_t`, `nfds_t`, `pid_t`,  
 16322 `ptrdiff_t`, `size_t`, `speed_t`, `ssize_t`, `suseconds_t`, `tcflag_t`, `wchar_t`, and `wint_t` types are no  
 16323 greater than the width of type `long`. The format of each name shall be suitable for use with  
 16324 the `getconf -v` option.

16325 `_CS_V8_ENV`  
 16326 This is the value that provides the environment variable information (other than that  
 16327 provided by `_CS_PATH`) that is required by the implementation to create a conforming  
 16328 environment, as described in the implementation's conformance documentation.

16329 OB The following symbolic constants are reserved for compatibility with Issue 7:

- 16330 `_CS_POSIX_V7_ILP32_OFF32_CFLAGS`
- 16331 `_CS_POSIX_V7_ILP32_OFF32_LDFLAGS`
- 16332 `_CS_POSIX_V7_ILP32_OFF32_LIBS`
- 16333 `_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS`
- 16334 `_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS`
- 16335 `_CS_POSIX_V7_ILP32_OFFBIG_LIBS`
- 16336 `_CS_POSIX_V7_LP64_OFF64_CFLAGS`
- 16337 `_CS_POSIX_V7_LP64_OFF64_LDFLAGS`
- 16338 `_CS_POSIX_V7_LP64_OFF64_LIBS`
- 16339 `_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS`
- 16340 `_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS`
- 16341 `_CS_POSIX_V7_LPBIG_OFFBIG_LIBS`

```

16342 _CS_POSIX_V7_THREADS_CFLAGS
16343 _CS_POSIX_V7_THREADS_LDFLAGS
16344 _CS_POSIX_V7_WIDTH_RESTRICTED_ENVS
16345 _CS_V7_ENV
    
```

16346 The implementation may define additional symbolic constants beginning with `_CS_` for use by  
 16347 `confstr()`.

16348 The <unistd.h> header shall define `SEEK_CUR`, `SEEK_END`, and `SEEK_SET` as described in  
 16349 <stdio.h>.

16350 Additionally, it shall define the following macros which shall expand to integer constant  
 16351 expressions with values that are distinct from each other and from `SEEK_CUR`, `SEEK_END`, and  
 16352 `SEEK_SET`:

```

16353 SEEK_HOLE    Seek forwards from offset relative to start-of-file for a position within a hole.
16354 SEEK_DATA    Seek forwards from offset relative to start-of-file for a position not within a
16355              hole
    
```

16356 XSI The <unistd.h> header shall define the following symbolic constants as possible values for the  
 16357 *function* argument to the `lockf()` function:

```

16358 F_LOCK      Lock a section for exclusive use.
16359 F_TEST      Test section for locks by other processes.
16360 F_TLOCK     Test and lock a section for exclusive use.
16361 F_ULOCK     Unlock locked sections.
    
```

16362 The <unistd.h> header shall define the following symbolic constants for `pathconf()`:

```

16363 _PC_2_SYMLINKS
16364 _PC_ALLOC_SIZE_MIN
16365 _PC_ASYNC_IO
16366 _PC_CHOWN_RESTRICTED
16367 _PC_FALLOC
16368 _PC_FILESIZEBITS
16369 _PC_LINK_MAX
16370 _PC_MAX_CANON
16371 _PC_MAX_INPUT
16372 _PC_NAME_MAX
16373 _PC_NO_TRUNC
16374 _PC_PATH_MAX
16375 _PC_PIPE_BUF
16376 _PC_PRIO_IO
16377 _PC_REC_INCR_XFER_SIZE
16378 _PC_REC_MAX_XFER_SIZE
16379 _PC_REC_MIN_XFER_SIZE
16380 _PC_REC_XFER_ALIGN
16381 _PC_SYMLINK_MAX
16382 _PC_SYNC_IO
16383 _PC_TEXTDOMAIN_MAX
16384 _PC_TIMESTAMP_RESOLUTION
16385 _PC_VDISABLE
    
```

16386 The implementation may define additional symbolic constants beginning with `_PC_` for use by  
16387 `pathconf()`.

16388 The **<unistd.h>** header shall define the following symbolic constants for `sysconf()`:

16389 `_SC_2_C_BIND`  
16390 `_SC_2_C_DEV`  
16391 `_SC_2_CHAR_TERM`  
16392 `_SC_2_FORT_RUN`  
16393 `_SC_2_LOCALEDEF`  
16394 `_SC_2_SW_DEV`  
16395 `_SC_2_UPE`  
16396 `_SC_2_VERSION`  
16397 `_SC_ADVISORY_INFO`  
16398 `_SC_AIO_LISTIO_MAX`  
16399 `_SC_AIO_MAX`  
16400 `_SC_AIO_PRIO_DELTA_MAX`  
16401 `_SC_ARG_MAX`  
16402 `_SC_ASYNCHRONOUS_IO`  
16403 `_SC_ATEXIT_MAX`  
16404 `_SC_BARRIERS`  
16405 `_SC_BC_BASE_MAX`  
16406 `_SC_BC_DIM_MAX`  
16407 `_SC_BC_SCALE_MAX`  
16408 `_SC_BC_STRING_MAX`  
16409 `_SC_CHILD_MAX`  
16410 `_SC_CLK_TCK`  
16411 `_SC_CLOCK_SELECTION`  
16412 `_SC_COLL_WEIGHTS_MAX`  
16413 `_SC_CPUTIME`  
16414 `_SC_DELAYTIMER_MAX`  
16415 `_SC_DEVICE_CONTROL`  
16416 `_SC_EXPR_NEST_MAX`  
16417 `_SC_FSYNC`  
16418 `_SC_GETGR_R_SIZE_MAX`  
16419 `_SC_GETPW_R_SIZE_MAX`  
16420 `_SC_HOST_NAME_MAX`  
16421 `_SC_IOV_MAX`  
16422 `_SC_IPV6`  
16423 `_SC_JOB_CONTROL`  
16424 `_SC_LINE_MAX`  
16425 `_SC_LOGIN_NAME_MAX`  
16426 `_SC_MAPPED_FILES`  
16427 `_SC_MEMLOCK`  
16428 `_SC_MEMLOCK_RANGE`  
16429 `_SC_MEMORY_PROTECTION`  
16430 `_SC_MESSAGE_PASSING`  
16431 `_SC_MONOTONIC_CLOCK`  
16432 `_SC_MQ_OPEN_MAX`  
16433 `_SC_MQ_PRIO_MAX`  
16434 `_SC_NGROUPS_MAX`  
16435 `_SC_NPROCESSORS_CONF`  
16436 `_SC_NPROCESSORS_ONLN`

16437 \_SC\_NSIG  
 16438 \_SC\_OPEN\_MAX  
 16439 \_SC\_PAGE\_SIZE  
 16440 \_SC\_PAGESIZE  
 16441 \_SC\_PRIORITIZED\_IO  
 16442 \_SC\_PRIORITY\_SCHEDULING  
 16443 \_SC\_RAW\_SOCKETS  
 16444 \_SC\_RE\_DUP\_MAX  
 16445 \_SC\_READER\_WRITER\_LOCKS  
 16446 \_SC\_REALTIME\_SIGNALS  
 16447 \_SC\_REGEX  
 16448 \_SC\_RTSIG\_MAX  
 16449 \_SC\_SAVED\_IDS  
 16450 \_SC\_SEM\_NSEMS\_MAX  
 16451 \_SC\_SEM\_VALUE\_MAX  
 16452 \_SC\_SEMAPHORES  
 16453 \_SC\_SHARED\_MEMORY\_OBJECTS  
 16454 \_SC\_SHELL  
 16455 \_SC\_SIGQUEUE\_MAX  
 16456 \_SC\_SPAWN  
 16457 \_SC\_SPIN\_LOCKS  
 16458 \_SC\_SPORADIC\_SERVER  
 16459 \_SC\_SS\_REPL\_MAX  
 16460 \_SC\_STREAM\_MAX  
 16461 \_SC\_SYMLOOP\_MAX  
 16462 \_SC\_SYNCHRONIZED\_IO  
 16463 \_SC\_THREAD\_ATTR\_STACKADDR  
 16464 \_SC\_THREAD\_ATTR\_STACKSIZE  
 16465 \_SC\_THREAD\_CPUTIME  
 16466 \_SC\_THREAD\_DESTRUCTOR\_ITERATIONS  
 16467 \_SC\_THREAD\_KEYS\_MAX  
 16468 \_SC\_THREAD\_PRIO\_INHERIT  
 16469 \_SC\_THREAD\_PRIO\_PROTECT  
 16470 \_SC\_THREAD\_PRIORITY\_SCHEDULING  
 16471 \_SC\_THREAD\_PROCESS\_SHARED  
 16472 \_SC\_THREAD\_ROBUST\_PRIO\_INHERIT  
 16473 \_SC\_THREAD\_ROBUST\_PRIO\_PROTECT  
 16474 \_SC\_THREAD\_SAFE\_FUNCTIONS  
 16475 \_SC\_THREAD\_SPARADIC\_SERVER  
 16476 \_SC\_THREAD\_STACK\_MIN  
 16477 \_SC\_THREAD\_THREADS\_MAX  
 16478 \_SC\_THREADS  
 16479 \_SC\_TIMEOUTS  
 16480 \_SC\_TIMER\_MAX  
 16481 \_SC\_TIMERS  
 16482 \_SC\_TTY\_NAME\_MAX  
 16483 \_SC\_TYPED\_MEMORY\_OBJECTS  
 16484 \_SC\_TZNAME\_MAX  
 16485 \_SC\_V8\_ILP32\_OFF32  
 16486 \_SC\_V8\_ILP32\_OFFBIG  
 16487 \_SC\_V8\_LP64\_OFF64  
 16488 \_SC\_V8\_LP64\_OFFBIG

```

16489 OB  _SC_V7_ILP32_OFF32
16490      _SC_V7_ILP32_OFFBIG
16491      _SC_V7_LP64_OFF64
16492      _SC_V7_LPBIG_OFFBIG
16493      _SC_VERSION
16494      _SC_XOPEN_CRYPT
16495      _SC_XOPEN_ENH_I18N
16496      _SC_XOPEN_REALTIME
16497      _SC_XOPEN_REALTIME_THREADS
16498      _SC_XOPEN_SHM
16499      _SC_XOPEN_UNIX
16500      _SC_XOPEN_UUCP
16501      _SC_XOPEN_VERSION

```

16502 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same value.

16503 The implementation may define additional symbolic constants beginning with `_SC_` for use by  
16504 `sysconf()`.

16505 The <unistd.h> header shall define the following symbolic constants for file streams:

- 16506 `STDERR_FILENO` File number of `stderr`; 2.
- 16507 `STDIN_FILENO` File number of `stdin`; 0.
- 16508 `STDOUT_FILENO` File number of `stdout`; 1.

16509 The <unistd.h> header shall define the following symbolic constant for terminal special  
16510 character handling:

- 16511 `_POSIX_VDISABLE` This symbol shall be defined to be the value of a character that shall  
16512 disable terminal special character handling as described in [Section 11.2.6](#)  
16513 (on page 212). This symbol shall always be set to a value other than `-1`.

16514 The <unistd.h> header shall define the following symbolic constant as a value for the flag used  
16515 by `posix_close()`:

- 16516 `POSIX_CLOSE_RESTART`  
16517 Allows restarts if a signal interrupts a close operation. This constant shall  
16518 not be 0 unless `posix_close()` never returns `-1` with `errno` set to `[EINTR]`.

### 16519 Type Definitions

16520 The <unistd.h> header shall define the `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types as  
16521 described in <sys/types.h>.

16522 The <unistd.h> header shall define the `intptr_t` type as described in <stdint.h>.

### 16523 Declarations

16524 The following shall be declared as functions and may also be defined as macros. Function  
16525 prototypes shall be provided.

```

16526 int      access(const char *, int);
16527 unsigned alarm(unsigned);
16528 int      chdir(const char *);
16529 int      chown(const char *, uid_t, gid_t);
16530 int      close(int);
16531 size_t   confstr(int, char *, size_t);

```

16532	XSI	char	*crypt(const char *, const char *);
16533		int	dup(int);
16534		int	dup2(int, int);
16535		int	dup3(int, int, int);
16536		_Noreturn	void
16537			_exit(int);
16538	OB XSI	void	encrypt(char [64], int);
16539		int	execl(const char *, const char *, ...);
16540		int	execle(const char *, const char *, ...);
16541		int	execlp(const char *, const char *, ...);
16542		int	execv(const char *, char *const []);
16543		int	execve(const char *, char *const [], char *const []);
16544		int	execvp(const char *, char *const []);
16545		int	faccessat(int, const char *, int, int);
16546		int	fchdir(int);
16547		int	fchown(int, uid_t, gid_t);
16548		int	fchownat(int, const char *, uid_t, gid_t, int);
16549	SIO	int	fdatasync(int);
16550		int	fexecve(int, char *const [], char *const []);
16551		pid_t	_Fork(void);
16552		pid_t	fork(void);
16553		long	fpathconf(int, int);
16554	FSC	int	fsync(int);
16555		int	ftruncate(int, off_t);
16556		char	*getcwd(char *, size_t);
16557		gid_t	getegid(void);
16558		int	getentropy(void *, size_t);
16559		uid_t	geteuid(void);
16560		gid_t	getgid(void);
16561		int	getgroups(int, gid_t []);
16562	XSI	long	gethostid(void);
16563		int	gethostname(char *, size_t);
16564		char	*getlogin(void);
16565		int	getlogin_r(char *, size_t);
16566		int	getopt(int, char * const [], const char *);
16567		pid_t	getpgid(pid_t);
16568		pid_t	getpgrp(void);
16569		pid_t	getpid(void);
16570		pid_t	getppid(void);
16571	XSI	int	getresgid(gid_t *restrict, gid_t *restrict, gid_t *restrict);
16572			
16573		int	getresuid(uid_t *restrict, uid_t *restrict, uid_t *restrict);
16574			
16575		pid_t	getsid(pid_t);
16576		uid_t	getuid(void);
16577		int	isatty(int);
16578		int	lchown(const char *, uid_t, gid_t);
16579		int	link(const char *, const char *);
16580		int	linkat(int, const char *, int, const char *, int);
16581	XSI	int	lockf(int, int, off_t);
16582		off_t	lseek(int, off_t, int);

```

16583 XSI      int      nice(int);
16584      long   pathconf(const char *, int);
16585      int    pause(void);
16586      int    pipe(int [2]);
16587      int    pipe2(int [2], int);
16588      int    posix_close(int, int);
16589      ssize_t  pread(int, void *, size_t, off_t);
16590      ssize_t  pwrite(int, const void *, size_t, off_t);
16591      ssize_t  read(int, void *, size_t);
16592      ssize_t  readlink(const char *restrict, char *restrict, size_t);
16593      ssize_t  readlinkat(int, const char *restrict, char *restrict,
16594                          size_t);
16595      int     rmdir(const char *);
16596      int     setegid(gid_t);
16597      int     seteuid(uid_t);
16598      int     setgid(gid_t);
16599      int     setpgid(pid_t, pid_t);
16600 XSI      int     setregid(gid_t, gid_t);
16601      int     setresgid(gid_t, gid_t, gid_t);
16602      int     setresuid(uid_t, uid_t, uid_t);
16603      int     setreuid(uid_t, uid_t);
16604      pid_t   setsid(void);
16605      int     setuid(uid_t);
16606      unsigned sleep(unsigned);
16607 XSI      void    swab(const void *restrict, void *restrict, ssize_t);
16608      int     symlink(const char *, const char *);
16609      int     symlinkat(const char *, int, const char *);
16610 XSI      void    sync(void);
16611      long   sysconf(int);
16612      pid_t   tcgetpgrp(int);
16613      int     tcsetpgrp(int, pid_t);
16614      int     truncate(const char *, off_t);
16615      char    *ttyname(int);
16616      int     ttyname_r(int, char *, size_t);
16617      int     unlink(const char *);
16618      int     unlinkat(int, const char *, int);
16619      ssize_t  write(int, const void *, size_t);

```

16620 The **<unistd.h>** header shall declare the following external variables:

```

16621 extern char *optarg;
16622 extern int  opterr, optind, optopt;

```

16623 Inclusion of the **<unistd.h>** header may make visible all symbols from the headers **<fcntl.h>**,  
16624 **<stddef.h>**, **<stdint.h>**, and **<stdio.h>**.



16625 **APPLICATION USAGE**

16626 POSIX.1-2024 only describes the behavior of systems that claim conformance to it. However,  
 16627 application developers who want to write applications that adapt to other versions of this  
 16628 standard (or to systems that do not conform to any POSIX standard) may find it useful to code  
 16629 them so as to conditionally compile different code depending on the value of  
 16630 `_POSIX_VERSION`, for example:

```
16631 #if _POSIX_VERSION >= 200112L
16632 /* Use the newer function that copes with large files. */
16633 off_t pos=ftello(fp);
16634 #else
16635 /* Either this is an old version of POSIX, or _POSIX_VERSION is
16636    not even defined, so use the traditional function. */
16637 long pos=ftell(fp);
16638 #endif
```

16639 Earlier versions of POSIX.1-2024 and of the Single UNIX Specification can be identified by the  
 16640 following macros:

16641 POSIX.1-1988 standard  
 16642 `_POSIX_VERSION == 198808L`

16643 POSIX.1-1990 standard  
 16644 `_POSIX_VERSION == 199009L`

16645 POSIX.1-1996 standard  
 16646 `_POSIX_VERSION == 199506L`

16647 Single UNIX Specification, Version 1  
 16648 `_XOPEN_UNIX` and `_XOPEN_VERSION == 4`

16649 Single UNIX Specification, Version 2  
 16650 `_XOPEN_UNIX` and `_XOPEN_VERSION == 500`

16651 POSIX.1-2001 and Single UNIX Specification, Version 3  
 16652 `_POSIX_VERSION == 200112L`, plus (if the XSI option is supported) `_XOPEN_UNIX` and  
 16653 `_XOPEN_VERSION == 600`

16654 POSIX.1-2008, POSIX.1-2017, and Single UNIX Specification, Version 4  
 16655 `_POSIX_VERSION == 200809L`, plus (if the XSI option is supported) `_XOPEN_UNIX` and  
 16656 `_XOPEN_VERSION == 700`

16657 Note that `_POSIX_VERSION` did not change in POSIX.1-2017 as it was technically identical to  
 16658 POSIX.1-2008 with its two technical corrigenda applied.

16659 POSIX.1-2024 does not make any attempt to define application binary interaction with the  
 16660 underlying operating system. However, application developers may find it useful to query  
 16661 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the  
 16662 operating system supports the necessary functionality as in the following program fragment:

```
16663 if (sysconf(_SC_VERSION) < 202405L) {
16664     fprintf(stderr, "POSIX.1-2024 system required, terminating \n");
16665     exit(1);
16666 }
```

16667 Implementations may support multiple programming environments with some of them  
 16668 conforming to this standard and some not conforming. The `_POSIX_Vn_ILP*` and  
 16669 `_POSIX_Vn_LP*` constants, and corresponding `sysconf()` and `getconf` calls, only indicate whether

16670 each programming environment is supported; they do not indicate anything about conformance  
16671 of the environments that are supported. For example, an implementation may support the  
16672 ILP32\_OFF32 environment for legacy reasons with a 32-bit `time_t`, whereas in a conforming  
16673 environment `time_t` is required to have a width of at least 64 bits. Application writers should  
16674 consult an implementation's POSIX Conformance Document for information about the  
16675 conformance of each supported programming environment.

16676 New applications should not use `_XOPEN_SHM` or `_XOPEN_ENH_I18N`.

#### 16677 RATIONALE

16678 As POSIX.1-2024 evolved, certain options became sufficiently standardized that it was  
16679 concluded that simply requiring one of the option choices was simpler than retaining the option.  
16680 However, for backwards-compatibility, the option flags (with required constant values) are  
16681 retained.

#### 16682 Version Test Macros

16683 The standard developers considered altering the definition of `_POSIX_VERSION` and removing  
16684 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed  
16685 by some to be minimal, and since the implementation of the functionality is potentially  
16686 problematic. However, they recognized that support for existing application binaries is a  
16687 concern to manufacturers, application developers, and the users of implementations conforming  
16688 to POSIX.1-2024.

16689 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide  
16690 the greatest degree of imaginable utility to the application developer or user, it is arguably better  
16691 than the production of a core image or some other equally obscure result. (It is also possible for  
16692 implementations to encode and recognize application binaries compiled in various  
16693 POSIX.1-conforming environments, and modify the semantics of the underlying system to  
16694 conform to the expectations of the application.) For the reasons outlined in the preceding  
16695 paragraphs and in the APPLICATION USAGE section, the standard developers elected to retain  
16696 the `_POSIX_VERSION` and `_SC_VERSION` functionality.

#### 16697 Compile-Time Symbolic Constants for System-Wide Options

16698 POSIX.1-2024 includes support in certain areas for the newly adopted policy governing options  
16699 and stubs.

16700 This policy provides flexibility for implementations in how they support options. It also  
16701 specifies how conforming applications can adapt to different implementations that support  
16702 different sets of options. It allows the following:

- 16703 1. If an implementation has no interest in supporting an option, it does not have to provide  
16704 anything associated with that option beyond the announcement that it does not support  
16705 it.
- 16706 2. An implementation can support a partial or incompatible version of an option (as a non-  
16707 standard extension) as long as it does not claim to support the option.
- 16708 3. An application can determine whether the option is supported. A strictly conforming  
16709 application must check this announcement mechanism before first using anything  
16710 associated with the option.

16711 There is an important implication of this policy. POSIX.1-2024 cannot dictate the behavior of  
16712 interfaces associated with an option when the implementation does not claim to support the  
16713 option. In particular, it cannot require that a function associated with an unsupported option  
16714 will fail if it does not perform as specified. However, this policy does not prevent a standard

16715 from requiring certain functions to always be present, but that they shall always fail on some  
 16716 implementations. The `setpgid()` function in the POSIX.1-1990 standard, for example, is  
 16717 considered appropriate.

16718 The POSIX standards include various options, and the C-language binding support for an  
 16719 option implies that the implementation must supply data types and function interfaces. An  
 16720 application must be able to discover whether the implementation supports each option.

16721 Any application must consider the following three cases for each option:

16722 1. Option never supported.

16723 The implementation advertises at compile time that the option will never be supported.  
 16724 In this case, it is not necessary for the implementation to supply any of the data types or  
 16725 function interfaces that are provided only as part of the option. The implementation  
 16726 might provide data types and functions that are similar to those defined by POSIX.1-2024,  
 16727 but there is no guarantee for any particular behavior.

16728 2. Option always supported.

16729 The implementation advertises at compile time that the option will always be supported.  
 16730 In this case, all data types and function interfaces shall be available and shall operate as  
 16731 specified.

16732 3. Option might or might not be supported.

16733 Some implementations might not provide a mechanism to specify support of options at  
 16734 compile time. In addition, the implementation might be unable or unwilling to specify  
 16735 support or non-support at compile time. In either case, any application that might use the  
 16736 option at runtime must be able to compile and execute. The implementation must  
 16737 provide, at compile time, all data types and function interfaces that are necessary to allow  
 16738 this. In this situation, there must be a mechanism that allows the application to query, at  
 16739 runtime, whether the option is supported. If the application attempts to use the option  
 16740 when it is not supported, the result is unspecified unless explicitly specified otherwise in  
 16741 POSIX.1-2024.

## 16742 FUTURE DIRECTIONS

16743 None.

## 16744 SEE ALSO

16745 <limits.h>, <stddef.h>, <stdint.h>, <stdio.h>, <sys/socket.h>, <sys/types.h>, <termios.h>,  
 16746 <wctype.h>

16747 XSH *access()*, *alarm()*, *chown()*, *close()*, *confstr()*, *crypt()*, *ctermid()*, *dup()*, *\_Exit()*, *encrypt()*, *exec*,  
 16748 *fchdir()*, *fchown()*, *fdatasync()*, *fork()*, *fpathconf()*, *fsync()*, *ftruncate()*, *getcwd()*, *getentropy()*,  
 16749 *getegid()*, *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*, *getlogin()*, *getopt()*, *getpgid()*,  
 16750 *getpgrp()*, *getpid()*, *getppid()*, *getresgid()*, *getresuid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*,  
 16751 *lockf()*, *lseek()*, *nice()*, *pause()*, *pipe()*, *read()*, *readlink()*, *rmdir()*, *setegid()*, *seteuid()*, *setgid()*,  
 16752 *setpgid()*, *setregid()*, *setresgid()*, *setresuid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*,  
 16753 *sync()*, *sysconf()*, *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *unlink()*, *write()*

## 16754 CHANGE HISTORY

16755 First released in Issue 1. Derived from Issue 1 of the SVID.

## 16756 Issue 5

16757 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 16758 Threads Extension.

16759 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.

16760 `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other  
16761 than `-1` by a conforming implementation.

16762 Large File System extensions are added.

16763 The type of the argument to `sbrk()` is changed from `int` to `intptr_t`.

16764 `_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of  
16765 constants for the `confstr()` function, and to the list of constants to the `sysconf()` function. These  
16766 are all marked EX.

#### 16767 Issue 6

16768 `_POSIX2_C_VERSION` is removed.

16769 The Open Group Corrigendum U026/4 is applied, adding the prototype for `fdatasync()`.

16770 The Open Group Corrigendum U026/1 is applied, adding the symbols `_SC_XOPEN_LEGACY`,  
16771 `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.

16772 The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI  
16773 STREAMS Option Group.

16774 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in  
16775 IEEE Std 1003.1-2001.

16776 The LEGACY symbol `_SC_PASS_MAX` is removed.

16777 The following new requirements on POSIX implementations derive from alignment with the  
16778 Single UNIX Specification:

- 16779 • The `_CS_POSIX_*` and `_CS_XBS5_*` constants are added for the `confstr()` function.
- 16780 • The `_SC_XBS5_*` constants are added for the `sysconf()` function.
- 16781 • The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.
- 16782 • The `uid_t`, `gid_t`, `off_t`, `pid_t`, and `useconds_t` types are mandated.

16783 The `gethostname()` prototype is added for sockets.

16784 A new section is added for System-Wide Options.

16785 Function prototypes for `setegid()` and `seteuid()` are added.

16786 Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`,  
16787 `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`,  
16788 `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and `pathconf()` variables are  
16789 added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`,  
16790 `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with IEEE Std  
16791 1003.1d-1999.

16792 The following are added for alignment with IEEE Std 1003.1j-2000:

- 16793 • Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`,  
16794 `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`,  
16795 `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`
- 16796 • `sysconf()` variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,  
16797 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and  
16798 `_SC_TYPED_MEMORY_OBJECTS`

16799 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,  
16800 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are

16801 introduced.

16802 The `getwd()` function is marked LEGACY.

16803 The **restrict** keyword is added to the prototypes for `readlink()` and `swab()`.

16804 Constants for options are now harmonized, so when supported they take the year of approval of  
16805 IEEE Std 1003.1-2001 as the value.

16806 The following are added for alignment with IEEE Std 1003.1q-2000:

- 16807 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,  
16808 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`
- 16809 • The `sysconf()` symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,  
16810 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`

16811 The `brk()` and `sbrk()` LEGACY functions are removed.

16812 The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning  
16813 information.

16814 The Open Group Base Resolution bwg2001-008 is applied, changing the `namelen` parameter for  
16815 `gethostname()` from `socklen_t` to `size_t`.

16816 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing “Thread Stack  
16817 Address Size” to “Thread Stack Size Attribute”.

16818 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the `_POSIX_IPV6`,  
16819 `_SC_V6`, and `_SC_RAW_SOCKETS` symbols.

16820 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description  
16821 in “Constants for Functions” for the `_CS_POSIX_V6_LP64_OFF64_CFLAGS`,  
16822 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`, and `_CS_POSIX_V6_LP64_OFF64_LIBS` symbols.

16823 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for  
16824 the `_PC*` and `_SC*` constants, since these are mandatory on all implementations.

16825 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/23 is applied, adding the  
16826 `_PC_SYMLINK_MAX` and `_SC_SYMLINK_MAX` constants.

16827 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/24 is applied, correcting the shading and  
16828 margin code for the `fsync()` function.

16829 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/25 is applied, adding the following text to  
16830 the APPLICATION USAGE section: “New applications should not use `_XOPEN_SHM` or  
16831 `_XOPEN_ENH_I18N`.”.

16832 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/29 is applied, clarifying the requirements  
16833 for when constants for Options and Option Groups can be defined or undefined.

16834 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/30 is applied, changing the  
16835 `_V6_ILP32_OFF32`, `_V6_ILP32_OFFBIG`, `_V6_LP64_OFF64`, and `_V6_LP64_OFFBIG` symbols to  
16836 `_POSIX_V6_ILP32_OFF32`, `_POSIX_V6_ILP32_OFFBIG`, `_POSIX_V6_LP64_OFF64`, and  
16837 `_POSIX_V6_LP64_OFFBIG`, respectively. This is for consistency with the `sysconf()` and `c99`  
16838 reference pages.

16839 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/31 is applied, adding that the format of  
16840 names of programming environments can be obtained using the `getconf -v` option.

16841 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/32 is applied, deleting the  
16842 `_SC_FILE_LOCKING`, `_SC_2_C_VERSION`, and `_SC_XOPEN_XCU_VERSION` constants.

16843 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/33 is applied, adding  
16844 `_SC_SS_REPL_MAX`, `_SC_TRACE_EVENT_NAME_MAX`, `_SC_TRACE_NAME_MAX`,  
16845 `_SC_TRACE_SYS_MAX`, and `_SC_TRACE_USER_EVENT_MAX` to the list of symbolic constants  
16846 for `sysconf()`.

16847 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/34 is applied, updating the prototype for  
16848 the `symlink()` function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.

16849 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/35 is applied, adding `_PC_2_SYMLINKS`  
16850 to the symbolic constants list for `pathconf()`. This corresponds to the definition of  
16851 `POSIX2_SYMLINKS` in the Shell and Utilities volume of IEEE Std 1003.1-2001.

## 16852 Issue 7

16853 Austin Group Interpretations 1003.1-2001 #026 and #047 are applied.

16854 Austin Group Interpretation 1003.1-2001 #166 is applied to permit an additional compiler flag to  
16855 enable threads.

16856 Austin Group Interpretation 1003.1-2001 #178 is applied, clarifying the values allowed for  
16857 `_POSIX2_CHAR_TERM`.

16858 SD5-XBD-ERN-41 is applied, adding the `_POSIX2_SYMLINKS` constant.

16859 SD5-XBD-ERN-76 and SD5-XBD-ERN-77 are applied.

16860 Symbols to support the UUCP Utilities option are added.

16861 The variables for the supported programming environments are updated to be V7.

16862 The LEGACY and obsolescent symbols are removed.

16863 The `faccessat()`, `fchownat()`, `fexecve()`, `linkat()`, `readlinkat()`, `symlinkat()`, and `unlinkat()` functions  
16864 are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

16865 The `_POSIX_TRACE*` constants from the Trace option are marked obsolescent.

16866 The `_POSIX2_PBS*` constants from the Batch Environment Services and Utilities option are  
16867 marked obsolescent.

16868 Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory  
16869 Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks,  
16870 Threads, Timeouts, and Timers options is moved to the Base.

16871 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options  
16872 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex  
16873 or Robust Mutex Priority Inheritance, respectively.

16874 This reference page is clarified with respect to macros and symbolic constants.

16875 Changes are made related to support for finegrained timestamps and the  
16876 `_POSIX_TIMESTAMP_RESOLUTION` constant is added.

16877 The `_SC_THREAD_ROBUST_PRIO_INHERIT` and `_SC_THREAD_ROBUST_PRIO_PROTECT`  
16878 symbolic constants are added.

16879 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0078 [311], XBD/TC1-2008/0079 [209],  
16880 and XBD/TC1-2008/0080 [360] are applied.

16881 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0085 [783], XBD/TC2-2008/0086 [911],  
16882 and XBD/TC2-2008/0087 [566] are applied.

16883 **Issue 8**

- 16884 Austin Group Defect 62 is applied, adding the `_Fork()` function.
- 16885 Austin Group Defect 339 is applied, adding `_SC_NPROCESSORS_CONF` and  
16886 `_SC_NPROCESSORS_ONLN`.
- 16887 Austin Group Defects 411 and 598 are applied, adding `dup3()` and `pipe2()`.
- 16888 Austin Group Defects 415 and 1357 are applied, adding `SEEK_HOLE` and `SEEK_DATA`.
- 16889 Austin Group Defect 529 is applied, adding the `POSIX_CLOSE_RESTART` symbolic constant  
16890 and the `posix_close()` function.
- 16891 Austin Group Defect 687 is applied, adding `_POSIX_FALLOC` and `_PC_FALLOC`.
- 16892 Austin Group Defect 729 is applied, adding `_POSIX_DEVICE_CONTROL` and  
16893 `_SC_DEVICE_CONTROL`.
- 16894 Austin Group Defect 741 is applied, adding `_SC_NSIG`.
- 16895 Austin Group Defects 1074 and 1116 are applied, changing the descriptions of  
16896 `_CS_POSIX_V8_THREADS_CFLAGS` and `_CS_POSIX_V8_THREADS_LDFLAGS`.
- 16897 Austin Group Defect 1122 is applied, adding `_PC_TEXTDOMAIN_MAX`.
- 16898 Austin Group Defect 1134 is applied, adding `getentropy()`.
- 16899 Austin Group Defect 1141 is applied, changing the RATIONALE section.
- 16900 Austin Group Defect 1192 is applied, marking the `encrypt()` function as obsolescent.
- 16901 Austin Group Defect 1302 is applied, adding `_Noreturn` to `_exit()`.
- 16902 Austin Group Defects 1330 and 1595 are applied, removing obsolescent interfaces and changing  
16903 ```_V7``` to ```_V8``` and ```_V6``` to ```_V7```.
- 16904 Austin Group Defects 1344 and 1666 are applied, adding `getresgid()`, `getresuid()`, `setresgid()`, and  
16905 `setresuid()`.
- 16906 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.
- 16907 Austin Group Defect 1456 is applied, clarifying the reservation of symbolic constants with the  
16908 prefix `_CS_`, `_PC_`, and `_SC_`.
- 16909 Austin Group Defect 1462 is applied, adding a paragraph to APPLICATION USAGE about  
16910 conformance of supported programming environments.
- 16911 Austin Group Defect 1473 is applied, updating the list of earlier versions of this standard in the  
16912 APPLICATION USAGE section.
- 16913 Austin Group Defect 1518 is applied, correcting the spelling of ```programming```.
- 16914 Austin Group Defect 1569 is applied, changing references to a **pointer** type to ```all pointer  
16915 types```.

16916 **NAME**

16917 utmpx.h — user accounting database definitions

16918 **SYNOPSIS**16919 XSI `#include <utmpx.h>`16920 **DESCRIPTION**16921 The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following  
16922 members:

16923	char	ut_user[]	User login name.
16924	char	ut_id[]	Unspecified initialization process identifier.
16925	char	ut_line[]	Device name.
16926	pid_t	ut_pid	Process ID.
16927	short	ut_type	Type of entry.
16928	struct timeval	ut_tv	Time entry was made.

16929 The **<utmpx.h>** header shall define the **pid\_t** type through **typedef**, as described in  
16930 **<sys/types.h>**.16931 The **<utmpx.h>** header shall define the **timeval** structure as described in **<sys/time.h>**.16932 Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.16933 The **<utmpx.h>** header shall define the following symbolic constants as possible values for the  
16934 *ut\_type* member of the **utmpx** structure:

16935	EMPTY	No valid user accounting information.
16936	BOOT_TIME	Identifies time of system boot.
16937	OLD_TIME	Identifies time when system clock changed.
16938	NEW_TIME	Identifies time after system clock changed.
16939	USER_PROCESS	Identifies a process.
16940	INIT_PROCESS	Identifies a process spawned by the init process.
16941	LOGIN_PROCESS	Identifies the session leader of a logged-in user.
16942	DEAD_PROCESS	Identifies a session leader who has exited.

16943 The following shall be declared as functions and may also be defined as macros. Function  
16944 prototypes shall be provided.

```

16945 void          endutxent(void);
16946 struct utmpx *getutxent(void);
16947 struct utmpx *getutxid(const struct utmpx *);
16948 struct utmpx *getutxline(const struct utmpx *);
16949 struct utmpx *pututxline(const struct utmpx *);
16950 void          setutxent(void);

```



- 16951 **APPLICATION USAGE**  
16952       None.
- 16953 **RATIONALE**  
16954       None.
- 16955 **FUTURE DIRECTIONS**  
16956       None.
- 16957 **SEE ALSO**  
16958       [<sys/time.h>](#), [<sys/types.h>](#)  
16959       XSH *endutxent()*
- 16960 **CHANGE HISTORY**  
16961       First released in Issue 4, Version 2.

16962 **NAME**16963            **wchar.h** — wide-character handling16964 **SYNOPSIS**

16965            #include &lt;wchar.h&gt;

16966 **DESCRIPTION**

16967 CX        Some of the functionality described on this reference page extends the ISO C standard.  
 16968            Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 16969            enable the visibility of these symbols in this header.

16970            The &lt;wchar.h&gt; header shall define the following types:

16971 CX        **FILE**            As described in <stdio.h>.16972 CX        **locale\_t**        As described in <locale.h>.

16973            **mbstate\_t**        A complete object type other than an array type that can hold the conversion  
 16974            state information necessary to convert between sequences of (possibly multi-  
 16975 CX            byte) characters and wide characters. If a codeset is being used such that an  
 16976            **mbstate\_t** needs to preserve more than two levels of reserved state, the results  
 16977            are unspecified.

16978            **size\_t**            As described in <stddef.h>.16979 CX        **va\_list**        As described in <stdarg.h>.16980            **wchar\_t**        As described in <stddef.h>.16981            **wint\_t**        An integer type capable of storing any valid value of **wchar\_t** or WEOF.

16982            The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are  
 16983            described in the <time.h> header.

16984            The implementation shall support one or more programming environments in which the width  
 16985            of **wint\_t** is no greater than the width of type **long**. The names of these programming  
 16986            environments can be obtained using the *confstr()* function or the *getconf* utility.

16987            The &lt;wchar.h&gt; header shall define the following macros:

16988            **WCHAR\_MAX**        As described in <stdint.h>.16989            **WCHAR\_MIN**        As described in <stdint.h>.

16990            **WEOF**            Constant expression of type **wint\_t** that is returned by several WP functions to  
 16991            indicate end-of-file.

16992            **NULL**            As described in <stddef.h>.

16993 CX        Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>,  
 16994            <string.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, and <time.h>.

16995            The following shall be declared as functions and may also be defined as macros. Function  
 16996            prototypes shall be provided for use with ISO C standard compilers. Arguments to functions in  
 16997            this list can point to arrays containing **wchar\_t** values that do not correspond to members of the  
 16998            character set of the current locale. Such values shall be processed according to the specified  
 16999            semantics, unless otherwise stated.

```
17000        wint_t            btowc (int);
17001        wint_t            fgetwc (FILE *);
17002        wchar_t        *fgetws (wchar_t *restrict, int, FILE *restrict);
17003        wint_t            fputwc (wchar_t, FILE *);
```

```

17004     int          fputws(const wchar_t *restrict, FILE *restrict);
17005     int          fwide(FILE *, int);
17006     int          fwprintf(FILE *restrict, const wchar_t *restrict, ...);
17007     int          fwscanf(FILE *restrict, const wchar_t *restrict, ...);
17008     wint_t       getwc(FILE *);
17009     wint_t       getwchar(void);
17010     size_t       mbrlen(const char *restrict, size_t, mbstate_t *restrict);
17011     size_t       mbrtowc(wchar_t *restrict, const char *restrict, size_t,
17012                 mbstate_t *restrict);
17013     int          mbsinit(const mbstate_t *);
17014     CX size_t     mbsnrtowcs(wchar_t *restrict, const char **restrict,
17015                             size_t, size_t, mbstate_t *restrict);
17016     size_t       mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
17017                             mbstate_t *restrict);
17018     CX FILE      *open_wmemstream(wchar_t **, size_t *);
17019     wint_t       putwc(wchar_t, FILE *);
17020     wint_t       putwchar(wchar_t);
17021     int          swprintf(wchar_t *restrict, size_t,
17022                          const wchar_t *restrict, ...);
17023     int          swscanf(const wchar_t *restrict,
17024                          const wchar_t *restrict, ...);
17025     wint_t       ungetwc(wint_t, FILE *);
17026     int          vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
17027     int          vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
17028     int          vswprintf(wchar_t *restrict, size_t,
17029                          const wchar_t *restrict, va_list);
17030     int          vswscanf(const wchar_t *restrict, const wchar_t *restrict,
17031                          va_list);
17032     int          vwprintf(const wchar_t *restrict, va_list);
17033     int          vwscanf(const wchar_t *restrict, va_list);
17034     CX wchar_t   *wcpcpy(wchar_t *restrict, const wchar_t *restrict);
17035     wchar_t     *wcpncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
17036     size_t       wcrtoomb(char *restrict, wchar_t, mbstate_t *restrict);
17037     CX int       wcscasecmp(const wchar_t *, const wchar_t *);
17038     int         wcscasecmp_l(const wchar_t *, const wchar_t *, locale_t);
17039     wchar_t     *wcscat(wchar_t *restrict, const wchar_t *restrict);
17040     wchar_t     *wcschr(const wchar_t *, wchar_t);
17041     int         wcscmp(const wchar_t *, const wchar_t *);
17042     int         wscoll(const wchar_t *, const wchar_t *);
17043     CX int       wscoll_l(const wchar_t *, const wchar_t *, locale_t);
17044     wchar_t     *wcscpy(wchar_t *restrict, const wchar_t *restrict);
17045     size_t       wcsncpy(const wchar_t *, const wchar_t *);
17046     CX wchar_t   *wcsdup(const wchar_t *);
17047     size_t       wcsftime(wchar_t *restrict, size_t,
17048                          const wchar_t *restrict, const struct tm *restrict);
17049     CX size_t     wcsncat(wchar_t *restrict, const wchar_t *restrict,
17050                          size_t);
17051     size_t       wcsncpy(wchar_t *restrict, const wchar_t *restrict,
17052                          size_t);
17053     size_t       wcslen(const wchar_t *);
17054     CX int       wcsncasecmp(const wchar_t *, const wchar_t *, size_t);
17055     int         wcsncasecmp_l(const wchar_t *, const wchar_t *, size_t,

```

```

17056         locale_t);
17057 wchar_t    *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
17058 int        wcsncmp(const wchar_t *, const wchar_t *, size_t);
17059 wchar_t    *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
17060 CX size_t    wcsnlen(const wchar_t *, size_t);
17061 size_t     wcsnrtombs(char *restrict, const wchar_t **restrict, size_t,
17062                    size_t, mbstate_t *restrict);
17063 wchar_t    *wcsprbrk(const wchar_t *, const wchar_t *);
17064 wchar_t    *wcsrchr(const wchar_t *, wchar_t);
17065 size_t     wcsrtombs(char *restrict, const wchar_t **restrict,
17066                    size_t, mbstate_t *restrict);
17067 size_t     wcsspncpy(const wchar_t *, const wchar_t *);
17068 wchar_t    *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
17069 double     wcstod(const wchar_t *restrict, wchar_t **restrict);
17070 float      wcstof(const wchar_t *restrict, wchar_t **restrict);
17071 wchar_t    *wcstok(wchar_t *restrict, const wchar_t *restrict,
17072                  wchar_t **restrict);
17073 long       wcstol(const wchar_t *restrict, wchar_t **restrict, int);
17074 long double wcstold(const wchar_t *restrict, wchar_t **restrict);
17075 long long  wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
17076 unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
17077 unsigned long long
17078 wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
17079 XSI int      wcswidth(const wchar_t *, size_t);
17080 size_t     wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
17081 CX size_t     wcsxfrm_l(wchar_t *restrict, const wchar_t *restrict,
17082                       size_t, locale_t);
17083 int        wctob(wint_t);
17084 XSI int      wcwidth(wchar_t);
17085 wchar_t    *wmemchr(const wchar_t *, wchar_t, size_t);
17086 int        wmemcmp(const wchar_t *, const wchar_t *, size_t);
17087 wchar_t    *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
17088 wchar_t    *wmemmove(wchar_t *, const wchar_t *, size_t);
17089 wchar_t    *wmemset(wchar_t *, wchar_t, size_t);
17090 int        wprintf(const wchar_t *restrict, ...);
17091 int        wscanf(const wchar_t *restrict, ...);

```

**APPLICATION USAGE**

None.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

[<ctype.h>](#), [<locale.h>](#), [<stdarg.h>](#), [<stddef.h>](#), [<stdint.h>](#), [<stdio.h>](#), [<stdlib.h>](#), [<string.h>](#), [<time.h>](#), [<wctype.h>](#)

XSH Section 2.2 (on page 496), [btowc\(\)](#), [confstr\(\)](#), [fgetwc\(\)](#), [fgetwts\(\)](#), [fputwc\(\)](#), [fputwts\(\)](#), [fwide\(\)](#), [fwprintf\(\)](#), [fwscanf\(\)](#), [getwc\(\)](#), [getwchar\(\)](#), [iswalnum\(\)](#), [iswalpchar\(\)](#), [iswcntrl\(\)](#), [iswctype\(\)](#), [iswdigit\(\)](#), [iswgraph\(\)](#), [iswlower\(\)](#), [iswprint\(\)](#), [iswpunct\(\)](#), [iswspace\(\)](#), [iswupper\(\)](#), [iswxdigit\(\)](#), [mbrlen\(\)](#), [mbrtowc\(\)](#), [mbsinit\(\)](#), [mbsrtowcs\(\)](#), [open\\_memstream\(\)](#), [putwc\(\)](#), [putwchar\(\)](#), [towlower\(\)](#),

17105 *towupper()*, *ungetwc()*, *vwprintf()*, *vwscanf()*, *wcrtomb()*, *wscasecmp()*, *wscat()*, *wcschr()*,  
 17106 *wscmp()*, *wscoll()*, *wscpy()*, *wscspn()*, *wcsdup()*, *wcsftime()*, *wcslcat()*, *wcslen()*, *wcsncat()*,  
 17107 *wcsncmp()*, *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsrtombs()*, *wcsspn()*, *wcsstr()*, *wcstod()*, *wcstok()*,  
 17108 *wcstol()*, *wcstoul()*, *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*,  
 17109 *wmemcpy()*, *wmemmove()*, *wmemset()*

17110 XCU *getconf*

## 17111 CHANGE HISTORY

17112 First released in Issue 4.

### 17113 Issue 5

17114 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

### 17115 Issue 6

17116 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and  
 17117 *wcwidth()* are marked as extensions.

17118 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()*  
 17119 function.

17120 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 17121 • Various function prototypes are updated to add the **restrict** keyword.
- 17122 • The functions *vwscanf()*, *vwscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are  
 17123 added.

17124 The type **wctype\_t**, the *isw\*()*, *to\*()*, and *wctype()* functions are marked as XSI extensions.

17125 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/26 is applied, adding the APPLICATION  
 17126 USAGE section.

### 17127 Issue 7

17128 The *mbsnrtowcs()*, *open\_wmemstream()*, *wcpcpy()*, *wcpncpy()*, *wscasecmp()*, *wcsdup()*,  
 17129 *wcsncasecmp()*, *wcsnlen()*, and *wcsnrtombs()* functions are added from The Open Group  
 17130 Technical Standard, 2006, Extended API Set Part 1.

17131 The *wscasecmp\_l()*, *wcsncasecmp\_l()*, *wscoll\_l()*, and *wcsxfrm\_l()* functions are added from The  
 17132 Open Group Technical Standard, 2006, Extended API Set Part 4.

17133 The **wctype\_t** type, and the *isw\*()*, *towlower()*, and *towupper()* functions are marked obsolescent  
 17134 in <wchar.h> since the ISO C standard requires the declarations to be in <wctype.h>.

17135 This reference page is clarified with respect to macros and symbolic constants, and a declaration  
 17136 for the **locale\_t** type is added.

17137 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0081 [380] is applied.

17138 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0088 [73] is applied.

### 17139 Issue 8

17140 Austin Group Defect 986 is applied, adding *wcslcat()* and *wcslcpy()*.

17141 Austin Group Defect 1302 is applied, aligning this header with the ISO/IEC 9899:2018 standard.

17142 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

17143 Austin Group Defect 1352 is applied, changing the APPLICATION USAGE and RATIONALE  
 17144 sections.

17145 **NAME**

17146 wctype.h — wide-character classification and mapping utilities

17147 **SYNOPSIS**

17148 #include &lt;wctype.h&gt;

17149 **DESCRIPTION**

17150 CX Some of the functionality described on this reference page extends the ISO C standard.  
 17151 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 496) to  
 17152 enable the visibility of these symbols in this header.

17153 The &lt;wctype.h&gt; header shall define the following types:

17154 **wint\_t** As described in <wchar.h>.

17155 **wctrans\_t** A scalar type that can hold values which represent locale-specific character  
 17156 mappings.

17157 **wctype\_t** A scalar type of a data object that can hold values which represent locale-  
 17158 specific character classification.

17159 CX The <wctype.h> header shall define the **locale\_t** type as described in <locale.h>.

17160 The &lt;wctype.h&gt; header shall define the following macro:

17161 WEOF As described in &lt;wchar.h&gt;.

17162 For all functions described in this header that accept an argument of type **wint\_t**, the value is  
 17163 representable as a **wchar\_t** or equals the value of WEOF. If this argument has any other value,  
 17164 the behavior is undefined.

17165 The behavior of these functions shall be affected by the *LC\_CTYPE* category of the current locale.

17166 CX Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>,  
 17167 <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, and <wchar.h>.

17168 The following shall be declared as functions and may also be defined as macros. Function  
 17169 prototypes shall be provided for use with ISO C standard compilers.

```

17170 int iswalnum(wint_t);
17171 CX int iswalnum_l(wint_t, locale_t);
17172 int iswalpha(wint_t);
17173 CX int iswalpha_l(wint_t, locale_t);
17174 int iswblank(wint_t);
17175 CX int iswblank_l(wint_t, locale_t);
17176 int iswcntrl(wint_t);
17177 CX int iswcntrl_l(wint_t, locale_t);
17178 int iswctype(wint_t, wctype_t);
17179 CX int iswctype_l(wint_t, wctype_t, locale_t);
17180 int iswdigit(wint_t);
17181 CX int iswdigit_l(wint_t, locale_t);
17182 int iswgraph(wint_t);
17183 CX int iswgraph_l(wint_t, locale_t);
17184 int iswlower(wint_t);
17185 CX int iswlower_l(wint_t, locale_t);
17186 int iswprint(wint_t);
17187 CX int iswprint_l(wint_t, locale_t);
17188 int iswpunct(wint_t);

```

```

17189 CX      int      iswpunct_l(wint_t, locale_t);
17190        int      iswspace(wint_t);
17191 CX      int      iswspace_l(wint_t, locale_t);
17192        int      iswupper(wint_t);
17193 CX      int      iswupper_l(wint_t, locale_t);
17194        int      iswxdigit(wint_t);
17195 CX      int      iswxdigit_l(wint_t, locale_t);
17196        wint_t    towctrans(wint_t, wctrans_t);
17197 CX      wint_t    towctrans_l(wint_t, wctrans_t, locale_t);
17198        wint_t    towlower(wint_t);
17199 CX      wint_t    towlower_l(wint_t, locale_t);
17200        wint_t    towupper(wint_t);
17201 CX      wint_t    towupper_l(wint_t, locale_t);
17202        wctrans_t wctrans(const char *);
17203 CX      wctrans_t wctrans_l(const char *, locale_t);
17204        wctype_t  wctype(const char *);
17205 CX      wctype_t  wctype_l(const char *, locale_t);

```

17206 **APPLICATION USAGE**

17207       None.

17208 **RATIONALE**

17209       None.

17210 **FUTURE DIRECTIONS**

17211       None.

17212 **SEE ALSO**

17213       <ctype.h>, <locale.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>,  
17214       <wchar.h>

17215       XSH Section 2.2 (on page 496), *iswalnum()*, *iswalphalpha()*, *iswblank()*, *iswcntrl()*, *iswctype()*,  
17216       *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*,  
17217       *setlocale()*, *towctrans()*, *towlower()*, *towupper()*, *wctrans()*, *wctype()*

17218 **CHANGE HISTORY**

17219       First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

17220 **Issue 6**

17221       The *iswblank()* function is added for alignment with the ISO/IEC 9899:1999 standard.

17222 **Issue 7**

17223       SD5-XBD-ERN-6 is applied.

17224       The *\*\_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set  
17225       Part 4.

17226       This reference page is clarified with respect to macros and symbolic constants.

17227 **Issue 8**

17228       Austin Group Defect 1330 is applied, moving the description of **wctype\_t** from <wchar.h> to  
17229       <wctype.h>.

17230 **NAME**

17231 wordexp.h — word-expansion types

17232 **SYNOPSIS**

17233 #include &lt;wordexp.h&gt;

17234 **DESCRIPTION**17235 The **<wordexp.h>** header shall define the structures and symbolic constants used by the  
17236 *wordexp()* and *wordfree()* functions.17237 The **<wordexp.h>** header shall define the **wordexp\_t** structure type, which shall include at least  
17238 the following members:

17239	size_t	we_wordc	Count of words matched by <i>words</i> .
17240	char	**we_wordv	Pointer to list of expanded words.
17241	size_t	we_offs	Slots to reserve at the beginning of <i>we_wordv</i> .

17242 The **<wordexp.h>** header shall define the following symbolic constants for use as flags for the  
17243 *wordexp()* function:

17244	WRDE_APPEND	Append words to those previously generated.
17245	WRDE_DOOFFS	Number of null pointers to prepend to <i>we_wordv</i> .
17246	WRDE_NOCMD	Fail if command substitution is requested.
17247	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result is the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.

17251 WRDE\_SHOWERR Do not redirect *stderr* to **/dev/null**.

17252 WRDE\_UNDEF Report error on an attempt to expand an undefined shell variable.

17253 The **<wordexp.h>** header shall define the following symbolic constants as error return values:

17254	WRDE_BADCHAR	One of the unquoted characters—<newline>, ' ', '&', ';', '<', '>', 17255 '(', ')', '{', '}'—appears in <i>words</i> in an inappropriate context.
17256	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
17257	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
17258	WRDE_NOSPACE	Attempt to allocate memory failed.
17259	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated 17260 string.

17261 The **<wordexp.h>** header shall define the **size\_t** type as described in **<stddef.h>**.17262 The following shall be declared as functions and may also be defined as macros. Function  
17263 prototypes shall be provided.

```
17264 int wordexp(const char *restrict, wordexp_t *restrict, int);
17265 void wordfree(wordexp_t *);
```



17266 **APPLICATION USAGE**

17267 None.

17268 **RATIONALE**

17269 None.

17270 **FUTURE DIRECTIONS**

17271 None.

17272 **SEE ALSO**

17273 &lt;stddef.h&gt;

17274 XSH *wordexp()*17275 **CHANGE HISTORY**

17276 First released in Issue 4. Derived from the ISO POSIX-2 standard.

17277 **Issue 6**17278 The **restrict** keyword is added to the prototype for *wordexp()*.

17279 The WRDE\_NOSYS constant is marked obsolescent.

17280 **Issue 7**

17281 The obsolescent WRDE\_NOSYS constant is removed.

17282 This reference page is clarified with respect to macros and symbolic constants.

17283 **Issue 8**17284 Austin Group Defect 1444 is applied, correcting cross-references to *wordexp()*.



17285

 *The Open Group Standard*

17286

**Vol. 2:**

17287

**System Interfaces, Issue 8**

17288

*The Open Group*

17289

*The Institute of Electrical and Electronics Engineers, Inc.*



17292 The System Interfaces volume of POSIX.1-2024 describes the interfaces offered to application  
17293 programs by POSIX-conformant systems.

## 17294 **1.1 Relationship to Other Formal Standards**

17295 This volume of POSIX.1-2024 is aligned with the following standards, except where stated  
17296 otherwise:

17297 ISO C (C17)

17298 ISO/IEC 9899:2018, Programming Languages — C.

17299 Parts of the ISO/IEC 9899:2018 standard (hereinafter referred to as the ISO C standard) are  
17300 referenced to describe requirements also mandated by this volume of POSIX.1-2024. Some  
17301 functions and headers included within this volume of POSIX.1-2024 have a version in the ISO C  
17302 standard; in this case CX markings are added as appropriate to show where the ISO C standard  
17303 has been extended (see [Section 1.8.1](#), on page 7). Any conflict between this volume of  
17304 POSIX.1-2024 and the ISO C standard is unintentional, except where stated otherwise.

17305 This volume of POSIX.1-2024 also allows, but does not require, mathematics functions to  
17306 support IEEE Std 754-1985 and IEEE Std 854-1987.

## 17307 **1.2 Format of Entries**

17308 The entries in [Chapter 3](#) are based on a common format as follows. The only sections relating to  
17309 conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

17310 **NAME**

17311 This section gives the name or names of the entry and briefly states its purpose.

17312 **SYNOPSIS**

17313 This section summarizes the use of the entry being described. If it is necessary to  
17314 include a header to use this function, the names of such headers are shown, for  
17315 example:

```
17316 #include <stdio.h>
```

17317 **DESCRIPTION**

17318 This section describes the functionality of the function or header.

17319 **RETURN VALUE**

17320 This section indicates the possible return values, if any.

17321 If the implementation can detect errors, “successful completion” means that no error  
17322 has been detected during execution of the function. If the implementation does detect  
17323 an error, the error is indicated.

17324 For functions where no errors are defined, “successful completion” means that if the

17325 implementation checks for errors, no error has been detected. If the implementation can  
17326 detect errors, and an error is detected, the indicated return value is returned and *errno*  
17327 may be set.

#### 17328 **ERRORS**

17329 This section gives the symbolic names of the error values returned by a function or  
17330 stored into a variable accessed through the symbol *errno* if an error occurs.

17331 “No errors are defined” means that error values returned by a function or stored into a  
17332 variable accessed through the symbol *errno*, if any, depend on the implementation.

#### 17333 **EXAMPLES**

17334 This section is informative.

17335 This section gives examples of usage, where appropriate. In the event of conflict  
17336 between an example and a normative part of this volume of POSIX.1-2024, the  
17337 normative material is to be taken as correct.

#### 17338 **APPLICATION USAGE**

17339 This section is informative.

17340 This section gives warnings and advice to application developers about the entry. In the  
17341 event of conflict between warnings and advice and a normative part of this volume of  
17342 POSIX.1-2024, the normative material is to be taken as correct.

#### 17343 **RATIONALE**

17344 This section is informative.

17345 This section contains historical information concerning the contents of this volume of  
17346 POSIX.1-2024 and why features were included or discarded by the standard  
17347 developers.

#### 17348 **FUTURE DIRECTIONS**

17349 This section is informative.

17350 This section provides comments which should be used as a guide to current thinking;  
17351 there is not necessarily a commitment to adopt these future directions.

#### 17352 **SEE ALSO**

17353 This section is informative.

17354 This section gives references to related information.

#### 17355 **CHANGE HISTORY**

17356 This section is informative.

17357 This section shows the derivation of the entry and any significant changes that have  
17358 been made to it.

# General Information

This chapter covers information that is relevant to all the functions specified in [Chapter 3](#) and [XBD Chapter 14](#) (on page 221).

## 2.1 Use and Implementation of Interfaces

### 2.1.1 Use and Implementation of Functions

Each of the following statements shall apply to all functions unless explicitly stated otherwise in the detailed descriptions that follow:

1. If an argument to a function has an invalid value, such as a value outside the domain of the function, a pointer to an object whose lifetime has ended (even if a new object now has the same address), a pointer outside the address space of the program, or a null pointer, the behavior is undefined.

2. Any function declared in a header may also be implemented as a macro defined in the header, so a function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the <left-parenthesis> that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a function even if it is also defined as a macro. The use of the C-language **#undef** construct to remove any such macro definition shall also ensure that an actual function is referred to.

3. Any invocation of a function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments.

4. For functions from the ISO C standard only, provided that the function can be declared without reference to any type defined in a header from the ISO C standard, it is also permissible to declare the function explicitly and use it without including its associated header.

5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

6. Functions shall prevent data races as follows: A function shall not directly or indirectly access objects accessible by threads other than the current thread unless the objects are accessed directly or indirectly via the function's arguments. A function shall not directly or indirectly modify objects accessible by threads other than the current thread unless the objects are accessed directly or indirectly via the function's non-const arguments. Implementations may share their own internal objects between threads if the objects are not visible to applications and are protected against data races.

- 17395 7. Functions shall perform all operations solely within the current thread if those operations  
17396 have effects that are visible to applications.

## 17397 2.1.2 Use and Implementation of Macros

17398 Each of the following statements shall apply to all macros unless explicitly stated otherwise:

- 17399 1. Any definition of an object-like macro in a header shall expand to code that is fully  
17400 protected by parentheses where necessary, so that it groups in an arbitrary expression as  
17401 if it were a single identifier.
- 17402 2. All object-like macros listed as expanding to integer constant expressions shall  
17403 additionally be suitable for use in **#if** preprocessing directives.
- 17404 3. Any definition of a function-like macro in a header shall expand to code that evaluates  
17405 each of its arguments exactly once, fully protected by parentheses where necessary, so  
17406 that it is generally safe to use arbitrary expressions as arguments.
- 17407 4. Any definition of a function-like macro in a header can be invoked in an expression  
17408 anywhere a function with a compatible return type could be called.

## 17409 2.2 The Compilation Environment

### 17410 2.2.1 POSIX.1 Symbols

17411 Certain symbols in this volume of POSIX.1-2024 are defined in headers (see XBD [Chapter 14](#), on  
17412 page 221). Some of those headers could also define symbols other than those defined by  
17413 POSIX.1-2024, potentially conflicting with symbols used by the application. Also, POSIX.1-2024  
17414 defines symbols that are not permitted by other standards to appear in those headers without  
17415 some control on the visibility of those symbols.

17416 Symbols called “feature test macros” are used to control the visibility of symbols that might be  
17417 included in a header. Implementations, future versions of this standard, and other standards  
17418 may define additional feature test macros.

17419 In the compilation of an application that **#defines** a feature test macro specified by  
17420 POSIX.1-2024, no header defined by POSIX.1-2024 shall be included prior to the definition of the  
17421 feature test macro. This restriction also applies to any implementation-provided header in  
17422 which these feature test macros are used. If the definition of the macro does not precede the  
17423 **#include**, the result is undefined.

17424 Feature test macros shall begin with the <underscore> character ('\_').

#### 17425 2.2.1.1 *The `_POSIX_C_SOURCE` Feature Test Macro*

17426 A POSIX-conforming application shall ensure that the feature test macro `_POSIX_C_SOURCE` is  
17427 defined before inclusion of any header.

17428 When an application includes a header described by POSIX.1-2024, and when this feature test  
17429 macro is defined to have the value 202405L:



- 17430 1. All symbols required by POSIX.1-2024 to appear when the header is included shall be  
17431 made visible.
- 17432 2. Symbols that are explicitly permitted, but not required, by POSIX.1-2024 to appear in that  
17433 header (including those in reserved name spaces) may be made visible.
- 17434 3. Additional symbols not required or explicitly permitted by POSIX.1-2024 to be in that  
17435 header shall not be made visible, except when enabled by another feature test macro.
- 17436 Identifiers in POSIX.1-2024 may only be undefined using the **#undef** directive as described in  
17437 [Section 2.1](#) (on page 495) or [Section 2.2.2](#) (on page 498). These **#undef** directives shall follow all  
17438 **#include** directives of any header in POSIX.1-2024.
- 17439 **Note:** The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been  
17440 superseded by `_POSIX_C_SOURCE`.

17441 2.2.1.2 *The `_XOPEN_SOURCE` Feature Test Macro*

- 17442 XSI An XSI-conforming application shall ensure that the feature test macro `_XOPEN_SOURCE` is  
17443 defined with the value 800 before inclusion of any header. This is needed to enable the  
17444 functionality described in [Section 2.2.1.1](#) (on page 496) and to ensure that the XSI option is  
17445 enabled.
- 17446 Since this volume of POSIX.1-2024 is aligned with the ISO C standard, and since all functionality  
17447 enabled by `_POSIX_C_SOURCE` set equal to 202405L is enabled by `_XOPEN_SOURCE` set equal  
17448 to 800, there should be no need to define `_POSIX_C_SOURCE` if `_XOPEN_SOURCE` is so  
17449 defined. Therefore, if `_XOPEN_SOURCE` is set equal to 800 and `_POSIX_C_SOURCE` is set equal  
17450 to 202405L, the behavior is the same as if only `_XOPEN_SOURCE` is defined and set equal to  
17451 800. However, should `_POSIX_C_SOURCE` be set to a value greater than 202405L, the behavior  
17452 is unspecified.
- 17453 If `_XOPEN_SOURCE` is defined with the value 800 and `_POSIX_C_SOURCE` is undefined before  
17454 inclusion of any header, then the header may define the `_POSIX_C_SOURCE` macro with the  
17455 value 202405L.

17456 2.2.1.3 *The `__STDC_WANT_LIB_EXT1__` Feature Test Macro*

- 17457 XSI A POSIX-conforming or XSI-conforming application can define the feature test macro  
17458 `__STDC_WANT_LIB_EXT1__` before inclusion of any header.
- 17459 When an application includes a header described by POSIX.1-2024, and when this feature test  
17460 macro is defined to have the value 1, the header may make visible those symbols specified for  
17461 the header in Annex K of the ISO C standard that are not already explicitly permitted by  
17462 POSIX.1-2024 to be made visible in the header. These symbols are listed in [Section 2.2.2](#) below.
- 17463 When an application includes a header described by POSIX.1-2024, and when this feature test  
17464 macro is either undefined or defined to have the value 0, the header shall not make any  
17465 additional symbols visible that are not already made visible by the feature test macro  
17466 XSI `_POSIX_C_SOURCE` or `_XOPEN_SOURCE` as described above, except when enabled by  
17467 another feature test macro.

17468 **2.2.2 The Name Space**

17469 All identifiers in this volume of POSIX.1-2024, except *environ*, are defined in at least one of the  
17470 XSI headers, as shown in XBD [Chapter 14](#) (on page 221). When `_XOPEN_SOURCE` or  
17471 `_POSIX_C_SOURCE` is defined, each header defines or declares some identifiers, potentially  
17472 conflicting with identifiers used by the application. The set of identifiers visible to the  
17473 application consists of precisely those identifiers from the header pages of the included headers,  
17474 as well as additional identifiers reserved for the implementation. In addition, some headers may  
17475 make visible identifiers from other headers as indicated on the relevant header pages.

17476 Implementations may also add members to a structure or union without controlling the  
17477 visibility of those members with a feature test macro, as long as a user-defined macro with the  
17478 same name cannot interfere with the correct interpretation of the program. The identifiers  
17479 reserved for use by the implementation are described below:

- 17480 1. Each identifier with external linkage described in the header section is reserved for use as  
17481 an identifier with external linkage if the header is included.
- 17482 2. Each macro described in the header section is reserved for any use if the header is  
17483 included.
- 17484 3. Each identifier with file scope described in the header section is reserved for use as a  
17485 macro name and as an identifier with file scope in the same name space if the header is  
17486 included.

17487 As described in [Chapter 13](#) (on page 219), the prefixes `posix_`, `POSIX_`, and `_POSIX_` are  
17488 reserved for use by POSIX.1-2024 and other POSIX standards. Implementations may add  
17489 symbols to the headers shown in the following table, provided the identifiers for those symbols  
17490 either:

- 17491 1. Begin with the corresponding reserved prefixes in the table, or
- 17492 2. Have one of the corresponding complete names in the table, or
- 17493 3. End in the string indicated as a reserved suffix in the table and do not use the reserved  
17494 prefixes `posix_`, `POSIX_`, or `_POSIX_`, as long as the reserved suffix is in that part of the  
17495 name considered significant by the implementation.

17496 Symbols that use the reserved prefix `_POSIX_` may be made visible by implementations in any  
17497 header defined by POSIX.1-2024.

	Header	Prefix	Suffix	Complete Name
17498	< aio.h >	aio_, lio_, AIO_, LIO_		
17499	<arpa/inet.h>	inet_		
17500	<ctype.h>	to[a-z], is[a-z]		
17501	<dlfcn.h>	RTLD_, dli_		
17502	<dirent.h>	d_, DT_		
17503	<fcntl.h>	l_		
17504	<fmtmsg.h>	MM_		
17505 XSI	<fnmatch.h>	FNM_		
17506	<ftw.h>	FTW		
17507 XSI	<glob.h>	gl_, GLOB_		
17508	<grp.h>	gr_		
17509	<libintl.h>			TEXTDOMAINMAX
17510	<limits.h>		_MAX, _MIN	
17511	<math.h>	M_		
17512 XSI	<mqueue.h>	mq_, MQ_		
17513 MSG	<ndbm.h>	dbm_, DBM_		
17514 XSI	<netdb.h>	ai_, h_, n_, p_, s_		
17515	<net/if.h>	if_, IF_		
17516	<netinet/in.h>	in_, ip_, s_, sin_, INADDR_, IPPROTO_		
17517				
17518 IP6	<netinet/tcp.h>	TCP_		
17519	<nl_types.h>	NL_		
17520	<poll.h>	pd_, ph_, ps_, POLL		
17521	<pthread.h>	pthread_, PTHREAD_		
17522	<pwd.h>	pw_		
17523	<regex.h>	re_, rm_, REG_		
17524	<sched.h>	sched_, SCHED_		
17525	<semaphore.h>	sem_, SEM_		
17526	<signal.h>	sa_, si_, sigev_, sival_, uc_, BUS_, CLD_, FPE_, ILL_, SA_, SEGV_, SI_, SIGEV_		
17527 CX		ss_, sv_, SS_, TRAP_		
17528 XSI	<stdatomic.h>	atomic_[a-z], memory_[a-z]		
17529	<stdlib.h>	str[a-z]		
17530	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
17531	<sys/ipc.h>	ipc_, IPC_		key, pad, seq
17532 XSI	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
17533	<sys/msg.h>	msg, MSG_[A-Z]		msg
17534 XSI	<sys/resource.h>	rlim_, ru_, PRIO_, RLIMIT_, RUSAGE_		
17535				
17536	<sys/select.h>	fd_, fds_, FD_		

	Header	Prefix	Suffix	Complete Name
17543				
17544				
17545	XSI <sys/sem.h>	sem, SEM_		sem
17546	XSI <sys/shm.h>	shm, SHM[A-Z], SHM_[A-Z]		
17547	<sys/socket.h>	cmsg_, if_, ifc_, ifra_, ifru_, infu_, l_, msg_, sa_, ss_, AF_, MSG_, PF_, SCM_, SHUT_, SO		
17548				
17549	XSI			
17550				
17551	<sys/stat.h>	st_		
17552	<sys/statvfs.h>	f_, ST_		
17553	XSI <sys/time.h>	tv_		
17554	<sys/times.h>	tms_		
17555	XSI <sys/uio.h>	iov_		UIO_MAXIOV
17556	<sys/un.h>	sun_		
17557	<sys/utsname.h>	uts_		
17558	<sys/wait.h>	P_, W[A-Z]		
17559	XSI <syslog.h>	LOG_		
17560	<termios.h>	c_, B[0-9], TC, ws_		
17561	<threads.h>	cnd_[a-z], mtx_[a-z], thrd_[a-z], tss_[a-z]		
17562				
17563	CX <time.h>	clock_, it_, timer_, tm_, tv_, CLOCK_, TIMER_		
17564				
17565	XSI <utmpx.h>	ut_	_LVL, _PROCESS, _TIME	
17566				
17567	<wchar.h>	wcs[a-z]		
17568	<wctype.h>	is[a-z], to[a-z]		
17569	<wordexp.h>	we_, WRDE_		
17570	CX ANY header		_t	

17571 **Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the  
 17572 portable character set. The notation [a-z] indicates any lowercase letter in the portable character  
 17573 set. Commas and spaces in the lists of prefixes and complete names in the above table are not  
 17574 part of any prefix or complete name. The ISO C standard reserves int[0-9a-z]\*\_t and uint[0-9a-  
 17575 z]\*\_t in <stdint.h>; this is not included in the table above because it is covered by the reserved  
 17576 \_t suffix for any header.

17577 Additional symbolic constants with the prefix \_CS\_, \_PC\_, and \_SC\_ may be defined by the  
 17578 inclusion of <unistd.h>, but as these are already reserved for the implementation, they are not  
 17579 included in the table above. Extensions with these prefixes should be compatible with use by  
 17580 *confstr()*, *pathconf()*, and *sysconf()*, respectively.

17581 Implementations may also add symbols to the <complex.h> header with the following complete  
 17582 names or the same names suffixed with 'f' or 'l':

17583	cerf	cerfc	cexp2
17584	cexpm1	clog10	clog1p
17585	clog2	clgamma	ctgamma

17586 If any header in the following table is included, macros with the prefixes or suffixes shown may  
 17587 be defined. After the last inclusion of a given header, an application may use identifiers with the  
 17588 corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the  
 17589 corresponding macro.

Header	Prefix	Suffix
17590 <endian.h>		_ENDIAN
17591 <errno.h>	E[0-9], E[A-Z]	
17592 <fcntl.h>	F_, O_	
17593 <fenv.h>	FE_[A-Z]	
17594 <inttypes.h>	PRI[Xa-z], SCN[Xa-z]	
17595 <locale.h>	LC_[A-Z]	
17596 <math.h>	FP_[A-Z]	
17597 <netinet/in.h>	IMPLINK_, IN_, IP_, IPPORT_, SOCK_,	
17598	IN6_	
17599 IP6 <signal.h>	SIG_, SIG[A-Z],	
17600	SV_	
17601 XSI <stdatomic.h>	ATOMIC_[A-Z]	
17602		
17603 CX <stdio.h>	SEEK_	
17604 XSI <sys/resource.h>	RLIM_	
17605 XSI <sys/socket.h>	CMSG_	
17606 <sys/stat.h>	S_	
17607 XSI <sys/uio.h>	IOV_	
17608 <termios.h>	I, O, V (See below.)	
17609 <time.h>	TIME_[A-Z]	
17610 <unistd.h>	SEEK_	

17611 The following are used to reserve complete names for the **<stdint.h>** header:

```

17612     INT[0-9A-Za-z-]*_MIN
17613     INT[0-9A-Za-z-]*_MAX
17614     INT[0-9A-Za-z-]*_C
17615     UINT[0-9A-Za-z-]*_MIN
17616     UINT[0-9A-Za-z-]*_MAX
17617     UINT[0-9A-Za-z-]*_C
  
```

17618 **Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the  
 17619 portable character set. The notation [Xa-z] indicates the character 'x' or any lowercase letter  
 17620 in the portable character set. The notation [0-9A-Za-z-]\* indicates zero or more occurrences of  
 17621 any of the following: a digit, an uppercase or lowercase letter in the portable character set, or an  
 17622 <underscore>.

17623 XSI The following reserved names are used as exact matches for **<termios.h>**:

17624 CBAUD	EXTB	VDSUSP
17625 DEFECHO	FLUSHO	VLNEXT
17626 ECHOCTL	LOBLK	VREPRINT
17627 ECHOK	PENDIN	VSTATUS
17628 ECHOPRT	SWTCH	VWERASE
17629 EXTA	VDISCARD	

17630 When the feature test macro `_STDC_WANT_LIB_EXT1__` is defined with the value 1 (see  
 17631 [Section 2.2.1](#), on page 496), implementations may add symbols to the headers shown in the  
 17632 following table provided the identifiers for those symbols have one of the corresponding

17633 complete names in the table.

Header	Complete Name
<stdio.h>	fopen_s, fprintf_s, freopen_s, fscanf_s, gets_s, printf_s, scanf_s, snprintf_s, sprintf_s, sscanf_s, tmpfile_s, tmpnam_s, vfprintf_s, vfscanf_s, vprintf_s, vscanf_s, vsnprintf_s, vsprintf_s, vsscanf_s
<stdlib.h>	abort_handler_s, bsearch_s, getenv_s, ignore_handler_s, mbstowcs_s, qsort_s, set_constraint_handler_s, wcstombs_s, wctomb_s
<time.h>	asctime_s, ctime_s, gmtime_s, localtime_s
<wchar.h>	fwprintf_s, fwscanf_s, mbsrtowcs_s, snwprintf_s, swprintf_s, swscanf_s, vfwprintf_s, vfwscanf_s, vsnwprintf_s, vswprintf_s, vswscanf_s, vwprintf_s, vwscanf_s, wcrctomb_s, wmemcpy_s, wmemmove_s, wprintf_s, wscanf_s

17646 When the feature test macro `__STDC_WANT_LIB_EXT1__` is defined with the value 1 (see  
 17647 [Section 2.2.1](#), on page 496), if any header in the following table is included, macros with the  
 17648 complete names shown may be defined.

Header	Complete Name
<stdint.h>	RSIZE_MAX
<stdio.h>	L_tmpnam_s, TMP_MAX_S

17652 **Note:** The above two tables only include those symbols from Annex K of the ISO C standard that are  
 17653 not already allowed to be visible by entries in earlier tables in this section.

- 17654 The following identifiers are reserved regardless of the inclusion of headers:
- 17655 1. With the exception of identifiers beginning with the prefix `_POSIX_` and those identifiers  
17656 which are lexically identical to keywords defined by the ISO C standard (for example  
17657 `_Bool`), all identifiers that begin with an `<underscore>` and either an uppercase letter or  
17658 another `<underscore>` are always reserved for any use by the implementation.
- 17659 2. All identifiers that begin with an `<underscore>` are always reserved for use as identifiers  
17660 with file scope in both the ordinary identifier and tag name spaces.
- 17661 3. All identifiers in the table below are reserved for use as identifiers with external linkage.  
17662 Some of these identifiers do not appear in this volume of POSIX.1-2024, but are reserved for  
17663 future use by the ISO C standard.
- 17664 4. All functions and external identifiers defined in XBD [Chapter 14](#) (on page 221) are reserved  
17665 for use as identifiers with external linkage.
- 17666 5. All the identifiers defined in this volume of POSIX.1-2024 that have external linkage and  
17667 *errno* are always reserved for use as identifiers with external linkage.
- 17668 **Note:** The notation `[a-z]` indicates any lowercase letter in the portable character set. The notation `'*'`  
17669 indicates any combination of digits, letters in the portable character set, or `<underscore>`.
- 17670 No other identifiers are reserved.

17671	_Exit	atomic_init	cerfcf
17672	abort	atomic_is_lock_free	cerfcf
17673	abs	atomic_load	cerff
17674	acos	atomic_load_explicit	cerfl
17675	acosh	atomic_signal_fence	cexpm1
17676	acoshf	atomic_store	cexpm1f
17677	acoshl	atomic_store_explicit	cexpm1l
17678	acoshl	atomic_thread_fence	cexp
17679	acosl	bsearch	cexp2
17680	aligned_alloc	btowc	cexp2f
17681	asctime	c16rtomb	cexp2l
17682	asin	c32rtomb	cexpf
17683	asinh	cabs	cexpl
17684	asinhf	cabsf	cimag
17685	asinhf	cabsl	cimagf
17686	asinhf	cacos	cimagl
17687	asinhf	cacosf	clearerr
17688	at_quick_exit	cacosh	clgamma
17689	atan	cacoshf	clgammaf
17690	atan2	cacoshl	clgammal
17691	atan2f	cacosl	clock
17692	atan2l	call_once	clog
17693	atanf	calloc	clog10
17694	atanh	carg	clog10f
17695	atanhf	cargf	clog10l
17696	atanhl	cargl	clog1p
17697	atanl	casin	clog1pf
17698	atexit	casinf	clog1pl
17699	atof	casinh	clog2
17700	atoi	casinhf	clog2f
17701	atol	casinhf	clog2l
17702	atoll	casinl	clogf
17703	atomic_compare_exchange_strong	catan	clogl
17704	atomic_compare_exchange_strong_explicit	catanf	cnd_broadcast
17705	atomic_compare_exchange_weak	catanh	cnd_destroy
17706	atomic_compare_exchange_weak_explicit	catanhf	cnd_init
17707	atomic_exchange	catanhf	cnd_signal
17708	atomic_exchange_explicit	catanhf	cnd_signal
17709	atomic_fetch_add	catanhl	cnd_timedwait
17710	atomic_fetch_add_explicit	catanl	cnd_wait
17711	atomic_fetch_and	cbrt	conj
17712	atomic_fetch_and_explicit	cbrtf	conjf
17713	atomic_fetch_or	cbrtl	conjl
17714	atomic_fetch_or_explicit	ccos	copysign
17715	atomic_fetch_sub	ccosf	copysignf
17716	atomic_fetch_sub_explicit	ccosh	copysignl
17717	atomic_fetch_xor	ccoshf	cos
17718	atomic_fetch_xor_explicit	ccoshl	cosf
17719	atomic_flag_clear	ccosl	cosh
17720	atomic_flag_clear_explicit	ceil	coshf
17721	atomic_flag_test_and_set	ceilf	coshl
17722	atomic_flag_test_and_set_explicit	ceilf	cosl
		ceilf	cpow



17723	cpowf	fdiml	fwprintf	longjmp
17724	cpowl	feclearexcept	fwrite	lrint
17725	cproj	fegetenv	fwscanf	lrintf
17726	cprojf	fegetexceptflag	getc	lrintl
17727	cprojl	fegetround	getchar	lround
17728	creal	feholdexcept	getenv	lroundf
17729	crealf	feof	getwc	lroundl
17730	creall	feraiseexcept	getwchar	malloc
17731	csin	ferror	gmtime	math_errhandling
17732	csinf	fesetenv	hypot	mblen
17733	csinh	fesetexceptflag	hypotf	mbrlen
17734	csinhf	fesetround	hypotl	mbrtoc16
17735	csinhl	fetestexcept	ilogb	mbrtoc32
17736	csinl	feupdateenv	ilogbf	mbrtowc
17737	csqrt	fflush	ilogbl	mbsinit
17738	csqrtf	fgetc	imaxabs	mbsrtowcs
17739	csqrtl	fgetpos	imaxdiv	mbstowcs
17740	ctan	fgets	is[a-z]*	mbtowc
17741	ctanf	fgetwc	kill_dependency	mem[a-z]*
17742	ctanh	fgetws	labs	mktime
17743	ctanhf	floor	ldexp	modf
17744	ctanhl	floorf	ldexpf	modff
17745	ctanl	floorl	ldexpl	modfl
17746	ctgamma	fma	ldiv	mtx_destroy
17747	ctgammaf	fmaf	lgamma	mtx_init
17748	ctgammal	fmal	lgammaf	mtx_lock
17749	ctime	fmax	lgammal	mtx_timedlock
17750	difftime	fmaxf	llabs	mtx_trylock
17751	div	fmaxl	lldiv	mtx_unlock
17752	erf	fmin	llrint	nan
17753	erfc	fminf	llrintf	nanf
17754	erfcf	fminl	llrintl	nanl
17755	erfcl	fmod	llround	nearbyint
17756	erff	fmodf	llroundf	nearbyintf
17757	erfl	fmodl	llroundl	nearbyintl
17758	errno	fopen	localeconv	nextafter
17759	exit	fprintf	localtime	nextafterf
17760	exp	fputc	log	nextafterl
17761	exp2	fputs	log10	nexttoward
17762	exp2f	fputwc	log10f	nexttowardf
17763	exp2l	fputws	log10l	nexttowardl
17764	expf	fread	log1p	perror
17765	expl	free	log1pf	pow
17766	expm1	freopen	log1pl	powf
17767	expm1f	frexp	log2	powl
17768	expm1l	frexpf	log2f	printf
17769	fabs	frexpl	log2l	putc
17770	fabsf	fscanf	logb	putchar
17771	fabsl	fseek	logbf	puts
17772	fclose	fsetpos	logbl	putwc
17773	fdim	ftell	logf	putwchar
17774	fdimf	fwide	logl	qsort

17775	quick_exit	setjmp	tgamma	vfprintf
17776	raise	setlocale	tgammaf	vfscanf
17777	rand	setvbuf	tgammal	vwprintf
17778	realloc	signal	thrd_create	vwscanf
17779	remainder	sin	thrd_current	vprintf
17780	remainderf	sinf	thrd_detach	vscanf
17781	remainderl	sinh	thrd_equal	vsnprintf
17782	remove	sinhf	thrd_exit	vsprintf
17783	remquo	sinhl	thrd_join	vsscanf
17784	remquof	sinl	thrd_sleep	vswprintf
17785	remquol	snprintf	thrd_yield	vswscanf
17786	rename	sprintf	time	vwprintf
17787	rewind	sqrt	timespec_get	vwscanf
17788	rint	sqrtf	tmpfile	wrtomb
17789	rintf	sqrtl	tmpnam	wcs[a-z]*
17790	rintl	srand	to[a-z]*	wctob
17791	round	sscanf	trunc	wctomb
17792	roundf	str[a-z]*	truncf	wctrans
17793	roundl	swprintf	truncl	wctype
17794	scalbln	swscanf	tss_create	wmemchr
17795	scalblnf	system	tss_delete	wmemcmp
17796	scalblnl	tan	tss_get	wmemcpy
17797	scalbn	tanf	tss_set	wmemmove
17798	scalbnf	tanh	ungetc	wmemset
17799	scalbnl	tanhf	ungetwc	wprintf
17800	scanf	tanhl	va_copy	wscanf
17801	setbuf	tanl	va_end	

17802       **Note:**       The notation [a–z] indicates any lowercase letter in the portable character set. The notation '\*'  
17803                   indicates any sequence of zero or more characters that are valid in identifiers with external  
17804                   linkage.

17805       Applications shall not declare or define identifiers with the same name as an identifier reserved  
17806       in the same context. Since macro names are replaced whenever found, independent of scope and  
17807       name space, macro names matching any of the reserved identifier names shall not be defined by  
17808       an application if any associated header is included.

17809       Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`,  
17810       headers may be included in any order, and each may be included more than once in a given  
17811       scope, with no difference in effect from that of being included only once.

17812       If used, the application shall ensure that a header is included outside of any external declaration  
17813       or definition, and it shall be first included before the first reference to any type or macro it  
17814       defines, or to any function or object it declares. However, if an identifier is declared or defined in  
17815       more than one header, the second and subsequent associated headers may be included after the  
17816       initial reference to the identifier. Prior to the inclusion of a header, or when any macro defined in  
17817       the header is expanded, the application shall not define any macros with names lexically  
17818       identical to symbols defined by that header.

17819 **2.3 Error Numbers**

17820 Most functions can provide an error number. The means by which each function provides its  
17821 error numbers is specified in its description.

17822 Some functions provide the error number in a variable accessed through the symbol *errno*,  
17823 defined by including the `<errno.h>` header. The value of *errno* should only be examined when it  
17824 is indicated to be valid by a function's return value. No function in this volume of POSIX.1-2024  
17825 shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by  
17826 function calls or assignments to *errno* by other threads.

17827 Some functions return an error number directly as the function value. These functions return a  
17828 value of zero to indicate success.

17829 If more than one error occurs in processing a function call, any one of the possible errors may be  
17830 returned, as the order of detection is undefined.

17831 Implementations may support additional errors not included in this list, may generate errors  
17832 included in this list under circumstances other than those described here, or may contain  
17833 extensions or limitations that prevent some errors from occurring.

17834 The ERRORS section on each reference page specifies which error conditions shall be detected  
17835 by all implementations ("shall fail") and which may be optionally detected by an  
17836 implementation ("may fail"). If no error condition is detected, the action requested shall be  
17837 successful. If an error condition is detected, the action requested may have been partially  
17838 performed, unless otherwise stated.

17839 Implementations may generate error numbers listed here under circumstances other than those  
17840 described, if and only if all those error conditions can always be treated identically to the error  
17841 conditions as described in this volume of POSIX.1-2024. Implementations shall not generate a  
17842 different error number from one required by this volume of POSIX.1-2024 for an error condition  
17843 described in this volume of POSIX.1-2024, but may generate additional errors unless explicitly  
17844 disallowed for a particular function.

17845 Each implementation shall document, in the conformance document, situations in which each of  
17846 the optional conditions defined in POSIX.1-2024 is detected. The conformance document may  
17847 also contain statements that one or more of the optional error conditions are not detected.

17848 Certain threads-related functions are not allowed to return an error code of [EINTR]. Where this  
17849 applies it is stated in the ERRORS section on the individual function pages.

17850 The following macro names identify the possible error numbers, in the context of the functions  
17851 specifically defined in this volume of POSIX.1-2024; these general descriptions are more  
17852 precisely defined in the ERRORS sections of the functions that return them. Only these macro  
17853 names should be used in programs, since the actual value of the error number is unspecified. All  
17854 values listed in this section shall be unique, except as noted below. The values for all these  
17855 macros shall be found in the `<errno.h>` header defined in the Base Definitions volume of  
17856 POSIX.1-2024. The actual values are unspecified by this volume of POSIX.1-2024.

17857 [E2BIG]

17858 Argument list too long. The sum of the number of bytes used by the new process image's  
17859 argument list and environment list is greater than the system-imposed limit of {ARG\_MAX}  
17860 bytes.

17861 or:

17862 Lack of space in an output buffer.

17863 or:

17864	Argument is greater than the system-imposed maximum.
17865	[EACCES]
17866	Permission denied. An attempt was made to access a file in a way forbidden by its file
17867	access permissions.
17868	[EADDRINUSE]
17869	Address in use. The specified address is in use.
17870	[EADDRNOTAVAIL]
17871	Address not available. The specified address is not available from the local system.
17872	[EAFNOSUPPORT]
17873	Address family not supported. The implementation does not support the specified address
17874	family, or the specified address is not a valid address for the address family of the specified
17875	socket.
17876	[EAGAIN]
17877	Resource temporarily unavailable. This is a temporary condition and later calls to the same
17878	routine may complete normally.
17879	[EALREADY]
17880	Connection already in progress. A connection request is already in progress for the specified
17881	socket.
17882	[EBADF]
17883	Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
17884	read (write) request is made to a file that is only open for writing (reading).
17885	[EBADMSG]
17886	Bad Message. The implementation has detected a corrupted message.
17887	[EBUSY]
17888	Resource busy. An attempt was made to make use of a system resource that is not currently
17889	available, as it is being used by another process in a manner that would have conflicted
17890	with the request being made by this process.
17891	[ECANCELED]
17892	Operation canceled. The associated asynchronous operation was canceled before
17893	completion.
17894	[ECHILD]
17895	No child process. A <i>wait()</i> , <i>waitid()</i> , or <i>waitpid()</i> function was executed by a process that
17896	had no existing or unwaited-for child process.
17897	[ECONNABORTED]
17898	Connection aborted. The connection has been aborted.
17899	[ECONNREFUSED]
17900	Connection refused. An attempt to connect to a socket was refused because there was no
17901	process listening or because the queue of connection requests was full and the underlying
17902	protocol does not support retransmissions.
17903	[ECONNRESET]
17904	Connection reset. The connection was forcibly closed by the peer.
17905	[EDEADLK]
17906	Resource deadlock would occur. An attempt was made to lock a system resource that would
17907	have resulted in a deadlock situation.

17908	[EDESTADDRREQ]
17909	Destination address required. No bind address was established.
17910	[EDOM]
17911	Domain error. An input argument is outside the defined domain of the mathematical
17912	function (defined in the ISO C standard).
17913	[EDQUOT]
17914	Reserved.
17915	[EEXIST]
17916	File exists. An existing file was mentioned in an inappropriate context; for example, as a
17917	new link name in the <i>link()</i> function.
17918	[EFAULT]
17919	Bad address. The system detected an invalid address in attempting to use an argument of a
17920	call. The reliable detection of this error cannot be guaranteed, and when not detected may
17921	result in the generation of a signal, indicating an address violation, which is sent to the
17922	process.
17923	[EFBIG]
17924	File too large. The size of a file would exceed the implementation's maximum file size, the
17925	file size limit of the process, or the offset maximum established in the corresponding open
17926	file description.
17927	[EHOSTUNREACH]
17928	Host is unreachable. The destination host cannot be reached (probably because the host is
17929	down or a remote router cannot reach it).
17930	[EIDRM]
17931	Identifier removed. Returned during XSI interprocess communication if an identifier has
17932	been removed from the system.
17933	[EILSEQ]
17934	Illegal byte sequence. A wide-character code has been detected that does not correspond to
17935	a valid character, or a byte sequence does not form a valid wide-character code (defined in
17936	the ISO C standard).
17937	[EINPROGRESS]
17938	Operation in progress. This code is used to indicate that an asynchronous operation has not
17939	yet completed.
17940	or:
17941	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
17942	immediately established.
17943	[EINTR]
17944	Interrupted function call. An asynchronous signal was caught by the process during the
17945	execution of an interruptible function. If the signal handler performs a normal return, the
17946	interrupted function call may return this condition (see the Base Definitions volume of
17947	POSIX.1-2024, <signal.h>).
17948	[EINVAL]
17949	Invalid argument. Some invalid argument was supplied; for example, specifying an
17950	undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
17951	[EIO]
17952	Input/output error. Some physical input or output error has occurred. This error may be
17953	reported on a subsequent operation on the same file descriptor. Any other error-causing

17954		operation on the same file descriptor may cause the [EIO] error indication to be lost.
17955		[EISCONN]
17956		Socket is connected. The specified socket is already connected.
17957		[EISDIR]
17958		Is a directory. An attempt was made to open a directory with write mode specified.
17959		[ELOOP]
17960		Symbolic link loop. A loop exists in symbolic links encountered during pathname resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links are encountered during pathname resolution.
17961		
17962		
17963		[EMFILE]
17964		File descriptor value too large or too many open streams. An attempt was made to open a
17965	XSI	file descriptor with a value greater than or equal to {OPEN_MAX}, or greater than or equal
17966		to the soft limit RLIMIT_NOFILE for the process (if smaller than {OPEN_MAX}); or an
17967		attempt was made to open more than the maximum number of streams allowed in the
17968		process.
17969		[EMLINK]
17970		Too many hard links. An attempt was made to have the link count of a single file exceed
17971		{LINK_MAX}.
17972		[EMSGSIZE]
17973		Message too large. A message sent on a transport provider was larger than an internal
17974		message buffer or some other network limit.
17975		or:
17976		Inappropriate message buffer length.
17977		[EMULTIHOP]
17978		Reserved.
17979		[ENAMETOOLONG]
17980		Filename too long. The length of a pathname exceeds {PATH_MAX} and the
17981		implementation considers this to be an error, or a pathname component is longer than
17982		{NAME_MAX}. This error may also occur when pathname substitution, as a result of
17983		encountering a symbolic link during pathname resolution, results in a pathname string the
17984		size of which exceeds {PATH_MAX}.
17985		[ENETDOWN]
17986		Network is down. The local network interface used to reach the destination is down.
17987		[ENETRESET]
17988		The connection was aborted by the network.
17989		[ENETUNREACH]
17990		Network unreachable. No route to the network is present.
17991		[ENFILE]
17992		Too many files open in system. Too many files are currently open in the system. The system
17993		has reached its predefined limit for simultaneously open files and temporarily cannot
17994		accept requests to open another one.
17995		[ENOBUFS]
17996		No buffer space available. Insufficient buffer resources were available in the system to
17997		perform the socket operation.

17998	[ENODEV]
17999	No such device. An attempt was made to apply an inappropriate function to a device; for
18000	example, trying to read a write-only device such as a printer.
18001	[ENOENT]
18002	No such file or directory. A component of a specified pathname does not exist, or the
18003	pathname is an empty string.
18004	[ENOEXEC]
18005	Executable file format error. A request is made to execute a file that, although it has
18006	appropriate privileges, is not in the format required by the implementation for executable
18007	files.
18008	[ENOLCK]
18009	No locks available. A system-imposed limit on the number of simultaneous file and record
18010	locks has been reached and no more are currently available.
18011	[ENOLINK]
18012	Reserved.
18013	[ENOMEM]
18014	Not enough space. The new process image requires more memory than is allowed by the
18015	hardware or system-imposed memory management constraints.
18016	[ENOMSG]
18017	No message of the desired type. The message queue does not contain a message of the
18018	required type during XSI interprocess communication.
18019	[ENOPROTOOPT]
18020	Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the
18021	implementation.
18022	[ENOSPC]
18023	No space left on a device. During the <i>write()</i> function on a regular file or when extending a
18024	directory, there is no free space left on the device.
18025	[ENOSYS]
18026	Functionality not supported. An attempt was made to use optional functionality that is not
18027	supported in this implementation.
18028	[ENOTCONN]
18029	Socket not connected. The socket is not connected.
18030	[ENOTDIR]
18031	Not a directory. A component of the specified pathname exists, but it is not a directory,
18032	when a directory was expected; or an attempt was made to create a non-directory file, and
18033	the specified pathname contains at least one non- <i>&lt;/i&gt;slash<i>&gt;</i> character and ends with one or</i>
18034	more trailing <i>&lt;/i&gt;slash<i>&gt;</i> characters.</i>
18035	[ENOTEMPTY]
18036	Directory not empty. A directory other than an empty directory was supplied when an
18037	empty directory was expected.
18038	[ENOTRECOVERABLE]
18039	State not recoverable. The state protected by a robust mutex is not recoverable.
18040	[ENOTSOCK]
18041	Not a socket. The file descriptor does not refer to a socket.

18042	[ENOTSUP]
18043	Not supported. The implementation does not support the requested feature or value.
18044	[ENOTTY]
18045	Inappropriate I/O control operation. A control function has been attempted for a file or
18046	special file for which the operation is inappropriate.
18047	[ENXIO]
18048	No such device or address. Input or output on a special file refers to a device that does not
18049	exist, or makes a request beyond the capabilities of the device. It may also occur when, for
18050	example, a tape drive is not on-line.
18051	[EOPNOTSUPP]
18052	Operation not supported on socket. The type of socket (address family or protocol) does not
18053	support the requested operation. A conforming implementation may assign the same values
18054	for [EOPNOTSUPP] and [ENOTSUP].
18055	[EOVERFLOW]
18056	Value too large to be stored in data type. An operation was attempted which would
18057	generate a value that is outside the range of values that can be represented in the relevant
18058	data type or that are allowed for a given data item.
18059	[EOWNERDEAD]
18060	Previous owner died. The owner of a robust mutex terminated while holding the mutex
18061	lock.
18062	[EPERM]
18063	Operation not permitted. An attempt was made to perform an operation limited to
18064	processes with appropriate privileges or to the owner of a file or other resource.
18065	[EPIPE]
18066	Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process
18067	to read the data.
18068	[EPROTO]
18069	Protocol error. Some protocol error occurred. This error is device-specific, but is generally
18070	not related to a hardware failure.
18071	[EPROTONOSUPPORT]
18072	Protocol not supported. The protocol is not supported by the address family, or the protocol
18073	is not supported by the implementation.
18074	[EPROTOTYPE]
18075	Protocol wrong type for socket. The socket type is not supported by the protocol.
18076	[ERANGE]
18077	Result too large or too small. The result of the function is too large (overflow) or too small
18078	(underflow) to be represented in the available space (defined in the ISO C standard).
18079	[EROFS]
18080	Read-only file system. An attempt was made to modify a file or directory on a file system
18081	that is read-only.
18082	[ESOCKTNOSUPPORT]
18083	Socket type not supported. The socket type is not supported by the address family, or the
18084	socket type is not supported by the implementation.
18085	[ESPIPE]
18086	Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.



18087	[ESRCH]
18088	No such process. No process can be found corresponding to that specified by the given
18089	process ID.
18090	[ESTALE]
18091	Reserved.
18092	[ETIMEDOUT]
18093	Connection timed out. The connection to a remote machine has timed out. If the connection
18094	timed out during execution of the function that reported this error (as opposed to timing
18095	out prior to the function being called), it is unspecified whether the function has completed
18096	some or all of the documented behavior associated with a successful completion of the
18097	function.
18098	or:
18099	Operation timed out. The time limit associated with the operation was exceeded before the
18100	operation completed.
18101	[ETXTBSY]
18102	Text file busy. An attempt was made to execute a pure-procedure program that is currently
18103	open for writing, or an attempt has been made to open for writing a pure-procedure
18104	program that is being executed.
18105	[EWOULDBLOCK]
18106	Operation would block. An operation on a socket marked as non-blocking has encountered
18107	a situation such as no data available that otherwise would have caused the function to
18108	suspend execution.
18109	A conforming implementation may assign the same values for [EWOULDBLOCK] and
18110	[EAGAIN].
18111	[EXDEV]
18112	Improper hard link. Creation of a hard link to a file on another file system was attempted.

### 18113 2.3.1 Additional Error Numbers

18114 Additional implementation-defined error numbers may be defined in `<errno.h>`.

## 18115 2.4 Signal Concepts

### 18116 2.4.1 Signal Generation and Delivery

18117 A signal is said to be “generated” for (or sent to) a process or thread when the event that causes  
 18118 the signal first occurs. Examples of such events include detection of hardware faults, timer  
 18119 expiration, signals generated via the `sigevent` structure and terminal activity, as well as  
 18120 invocations of the `kill()` and `sigqueue()` functions. In some circumstances, the same event  
 18121 generates signals for multiple processes.

18122 At the time of generation, a determination shall be made whether the signal has been generated  
 18123 for the process or for a specific thread within the process. Signals which are generated by some  
 18124 action attributable to a particular thread, such as a hardware fault, shall be generated for the  
 18125 thread that caused the signal to be generated. Signals that are generated in association with a

18126 process ID or process group ID or an asynchronous event, such as terminal activity, shall be  
18127 generated for the process.

18128 Each process has an action to be taken in response to each signal defined by the system (see  
18129 [Section 2.4.3](#)). A signal is said to be “delivered” to a process when the appropriate action for the  
18130 process and signal is taken. A signal is said to be “accepted” by a process when the signal is  
18131 selected and returned by one of the *sigwait()* functions.

18132 During the time between the generation of a signal and its delivery or acceptance, the signal is  
18133 said to be “pending”. Ordinarily, this interval cannot be detected by an application. However, a  
18134 signal can be “blocked” from delivery to a thread. If the action associated with a blocked signal  
18135 is anything other than to ignore the signal, and if that signal is generated for the thread, the  
18136 signal shall remain pending until it is unblocked, it is accepted when it is selected and returned  
18137 by a call to the *sigwait()* function, or the action associated with it is set to ignore the signal.  
18138 Signals generated for the process shall be delivered to exactly one of those threads within the  
18139 process which is in a call to a *sigwait()* function selecting that signal or has not blocked delivery  
18140 of the signal. If there are no threads in a call to a *sigwait()* function selecting that signal, and if all  
18141 threads within the process block delivery of the signal, the signal shall remain pending on the  
18142 process until a thread calls a *sigwait()* function selecting that signal, a thread unblocks delivery  
18143 of the signal, or the action associated with the signal is set to ignore the signal. If the action  
18144 associated with a blocked signal is to ignore the signal and if that signal is generated for the  
18145 process, it is unspecified whether the signal is discarded immediately upon generation or  
18146 remains pending.

18147 Each thread has a “signal mask” that defines the set of signals currently blocked from delivery  
18148 to it. The signal mask for a thread shall be initialized from that of its parent or creating thread,  
18149 or from the corresponding thread in the parent process if the thread was created as the result of a  
18150 call to *fork()*. The *pthread\_sigmask()*, *sigaction()*, *sigprocmask()*, and *sigsuspend()* functions control  
18151 the manipulation of the signal mask.

18152 The determination of which action is taken in response to a signal is made at the time the signal  
18153 is delivered, allowing for any changes since the time of generation. This determination is  
18154 independent of the means by which the signal was originally generated. If a subsequent  
18155 occurrence of a pending signal is generated, it is implementation-defined as to whether the  
18156 signal is delivered or accepted more than once in circumstances other than those in which  
18157 queuing is required. The order in which multiple, simultaneously pending signals outside the  
18158 range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

18159 When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process or  
18160 thread, all pending SIGCONT signals for that process or any of the threads within that process  
18161 shall be discarded. Conversely, when SIGCONT is generated for a process or thread, all pending  
18162 stop signals for that process or any of the threads within that process shall be discarded. When  
18163 SIGCONT is generated for a process that is stopped, the process shall be continued, even if the  
18164 SIGCONT signal is ignored by the process or is blocked by all threads within the process and  
18165 there are no threads in a call to a *sigwait()* function selecting SIGCONT. If SIGCONT is blocked  
18166 by all threads within the process, there are no threads in a call to a *sigwait()* function selecting  
18167 SIGCONT, and SIGCONT is not ignored by the process, the SIGCONT signal shall remain  
18168 pending on the process until it is either unblocked by a thread or a thread calls a *sigwait()*  
18169 function selecting SIGCONT, or a stop signal is generated for the process or any of the threads  
18170 within the process.

18171 An implementation shall document any condition not specified by this volume of POSIX.1-2024  
18172 under which the implementation generates signals.

## 18173 2.4.2 Realtime Signal Generation and Delivery

18174 This section describes functionality to support realtime signal generation and delivery.

18175 Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O  
 18176 completion, interprocess message arrival, and the *sigqueue()* function, support the specification  
 18177 of an application-defined value, either explicitly as a parameter to the function or in a **sigevent**  
 18178 structure parameter. The **sigevent** structure is defined in `<signal.h>` and contains at least the  
 18179 following members:

Member Type	Member Name	Description
<b>int</b>	<i>sigev_notify</i>	Notification type.
<b>int</b>	<i>sigev_signo</i>	Signal number.
<b>union signal</b>	<i>sigev_value</i>	Signal value.
<b>void*(*) (union signal)</b>	<i>sigev_notify_function</i>	Notification function.
<b>(pthread_attr_t*)</b>	<i>sigev_notify_attributes</i>	Notification attributes.

18186 The *sigev\_notify* member specifies the notification mechanism to use when an asynchronous  
 18187 event occurs. This volume of POSIX.1-2024 defines the following values for the *sigev\_notify*  
 18188 member:

18189 SIGEV\_NONE No asynchronous notification shall be delivered when the event of  
 18190 interest occurs.

18191 SIGEV\_SIGNAL The signal specified in *sigev\_signo* shall be generated for the process when  
 18192 the event of interest occurs. If the SA\_SIGINFO flag is set for that signal  
 18193 number, then the signal shall be queued to the process and the value  
 18194 specified in *sigev\_value* shall be the *si\_value* component of the generated  
 18195 signal. If SA\_SIGINFO is not set for that signal number, it is unspecified  
 18196 whether the signal is queued and what value, if any, is sent.

18197 SIGEV\_THREAD A notification function shall be called to perform notification.

18198 An implementation may define additional notification mechanisms.

18199 The *sigev\_signo* member specifies the signal to be generated. The *sigev\_value* member is the  
 18200 application-defined value to be passed to the signal-catching function at the time of the signal  
 18201 delivery or to be returned at signal acceptance as the *si\_value* member of the **siginfo\_t** structure.

18202 The **signal** union is defined in `<signal.h>` and contains at least the following members:

Member Type	Member Name	Description
<b>int</b>	<i>sival_int</i>	Integer signal value.
<b>void*</b>	<i>sival_ptr</i>	Pointer signal value.

18206 The *sival\_int* member shall be used when the application-defined value is of type **int**; the  
 18207 *sival\_ptr* member shall be used when the application-defined value is a pointer.

18208 When a signal is generated by the *sigqueue()* function or any signal-generating function that  
 18209 supports the specification of an application-defined value, the signal shall be marked pending  
 18210 and, if the SA\_SIGINFO flag is set for that signal, the signal shall be queued to the process along  
 18211 with the application-specified signal value. Multiple occurrences of signals so generated are  
 18212 queued in FIFO order. It is unspecified whether signals so generated are queued when the  
 18213 SA\_SIGINFO flag is not set for that signal.

18214 Signals generated by the *kill()* function or other events that cause signals to occur, such as  
 18215 detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the  
 18216 implementation does not support queuing, shall have no effect on signals already queued for the

18217 same signal number.

18218 When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the  
18219 behavior shall be as if the implementation delivers the pending unblocked signal with the  
18220 lowest signal number within that range. No other ordering of signal delivery is specified.

18221 If, when a pending signal is delivered, there are additional signals queued to that signal number,  
18222 the signal shall remain pending. Otherwise, the pending indication shall be reset.

18223 Multi-threaded programs can use an alternate event notification mechanism. When a  
18224 notification is processed, and the *sigev\_notify* member of the **sigevent** structure has the value  
18225 SIGEV\_THREAD, the function *sigev\_notify\_function* is called with parameter *sigev\_value*.

18226 The function shall be executed in a newly created thread as if it were the *start\_routine* for a call to  
18227 *pthread\_create()* with the thread attributes specified by *sigev\_notify\_attributes*. If  
18228 *sigev\_notify\_attributes* is NULL, the behavior shall be as if the thread were created with the  
18229 *detachstate* attribute set to PTHREAD\_CREATE\_DETACHED. Supplying an attributes structure  
18230 with a *detachstate* attribute of PTHREAD\_CREATE\_JOINABLE results in undefined behavior. It  
18231 is implementation-defined whether the signal mask of this thread has all signals except SIGKILL  
18232 and SIGSTOP blocked, or is the same as the mask that was in effect for the thread which  
18233 installed the **sigevent** notification handler at the time of the call that installed the handler.

### 18234 2.4.3 Signal Actions

18235 There are three types of action that can be associated with a signal: SIG\_DFL, SIG\_IGN, or a  
18236 pointer to a function. Initially, all signals shall be set to SIG\_DFL or SIG\_IGN prior to entry of  
18237 the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows.

#### 18238 SIG\_DFL

18239 Signal-specific default action.

18240 The default actions for the signals defined in this volume of POSIX.1-2024 are specified under  
18241 **<signal.h>**. The default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX  
18242 shall be to terminate the process abnormally.

18243 If the default action is to terminate the process abnormally, the process is terminated as if by a  
18244 call to *\_exit()*, except that the status made available to *wait()*, *waitid()*, and *waitpid()* indicates  
18245 abnormal termination by the signal. If the default action is to terminate the process abnormally  
18246 with additional actions, implementation-defined abnormal termination actions, such as creation  
18247 of a core image, may also occur.

18248 If the default action is to stop the process, the execution of that process is temporarily  
18249 suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process,  
18250 unless the parent process has set the SA\_NOCLDSTOP flag. While a process is stopped, any  
18251 additional signals that are sent to the process shall not be delivered until the process is  
18252 continued, except SIGKILL which always terminates the receiving process. A process that is a  
18253 member of an orphaned process group shall not be allowed to stop in response to the SIGTSTP,  
18254 SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a  
18255 process, the signal shall be discarded.

18256 If the default action is to ignore the signal, delivery of the signal shall have no effect on the  
18257 process.

18258 Setting a signal action to SIG\_DFL for a signal that is pending, and whose default action is to  
18259 ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded,  
18260 whether or not it is blocked. Any queued values pending shall be discarded and the resources

18261 used to queue them shall be released and returned to the system for other use.  
 18262 The default action for SIGCONT is to resume execution at the point where the process was  
 18263 stopped, after first handling any pending unblocked signals.

18264 XSI When a stopped process is continued, a SIGCHLD signal may be generated for its parent  
 18265 process, unless the parent process has set the SA\_NOCLDSTOP flag.

## 18266 SIG\_IGN

18267 Ignore signal.

18268 Delivery of the signal shall have no effect on the process. The behavior of a process is undefined  
 18269 after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by *kill()*,  
 18270 *sigqueue()*, or *raise()*.

18271 The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG\_IGN.

18272 Setting a signal action to SIG\_IGN for a signal that is pending shall cause the pending signal to  
 18273 be discarded, whether or not it is blocked.

18274 If a process sets the action for the SIGCHLD signal to SIG\_IGN, the behavior is unspecified,  
 18275 XSI except as specified under “Consequences of Process Termination” in the description of the  
 18276 *\_Exit()* function (see XSH *\_Exit()*, on page 568).

18277 Any queued values pending shall be discarded and the resources used to queue them shall be  
 18278 released and made available to queue other signals.

## 18279 Pointer to a Function

18280 Catch signal.

18281 On delivery of the signal, the receiving process is to execute the signal-catching function at the  
 18282 specified address. After returning from the signal-catching function, the receiving process shall  
 18283 resume execution at the point at which it was interrupted.

18284 If the SA\_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as  
 18285 a C-language function call as follows:

```
18286 void func(int signo);
```

18287 If the SA\_SIGINFO flag for the signal is set, the signal-catching function shall be entered as a C-  
 18288 language function call as follows:

```
18289 void func(int signo, siginfo_t *info, void *context);
```

18290 where *func* is the specified signal-catching function, *signo* is the signal number of the signal  
 18291 being delivered, and *info* is a pointer to a **siginfo\_t** structure defined in **<signal.h>** containing at  
 18292 least the following members:

18293	Member Type	Member Name	Description
18294	<b>int</b>	<i>si_signo</i>	Signal number.
18295	<b>int</b>	<i>si_code</i>	Cause of the signal.
18296	<b>pid_t</b>	<i>si_pid</i>	Sending process ID.
18297	<b>uid_t</b>	<i>si_uid</i>	Real user ID of sending process.
18298	<b>void *</b>	<i>si_addr</i>	Address of faulting instruction.
18299	<b>int</b>	<i>si_status</i>	Exit value or signal.
18300	<b>union signal</b>	<i>si_value</i>	Signal value.

18301 The *si\_signo* member shall contain the signal number. This shall be the same as the *signo*  
 18302 parameter. The *si\_code* member shall contain a code identifying the cause of the signal. The

18303 following non-signal-specific values are defined for *si\_code*:

18304 SI\_USER The signal was sent by the *kill()* function. The implementation may set *si\_code*  
18305 to SI\_USER if the signal was sent by the *raise()* or *abort()* functions or any  
18306 similar functions provided as implementation extensions.

18307 SI\_QUEUE The signal was sent by the *sigqueue()* function.

18308 SI\_TIMER The signal was generated by the expiration of a timer set by *timer\_settime()*.

18309 SI\_ASYNCIO The signal was generated by the completion of an asynchronous I/O request.

18310 MSG SI\_MESGQ The signal was generated by the arrival of a message on an empty message  
18311 queue.

18312 Signal-specific values for *si\_code* are also defined, as described in XBD <signal.h>.

18313 If the signal was not generated by one of the functions or events listed above, *si\_code* shall be set  
18314 either to one of the signal-specific values described in XBD <signal.h>, or to an implementation-  
18315 defined value that is not equal to any of the values defined above.

18316 XSI If *si\_code* is SI\_USER or SI\_QUEUE, or any value less than or equal to 0, then the signal was  
18317 generated by a process and *si\_pid* and *si\_uid* shall be set to the process ID and the real user ID of  
18318 the sender, respectively.

18319 In addition, *si\_addr*, *si\_pid*, *si\_status*, and *si\_uid* shall be set for certain signal-specific values of  
18320 *si\_code*, as described in XBD <signal.h>.

18321 If *si\_code* is one of SI\_QUEUE, SI\_TIMER, SI\_ASYNCIO, or SI\_MESGQ, then *si\_value* shall  
18322 contain the application-specified signal value. Otherwise, the contents of *si\_value* are undefined.

18323 The behavior of a process is undefined after it returns normally from a signal-catching function  
18324 for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not generated by *kill()*, *sigqueue()*,  
18325 or *raise()*.

18326 The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

18327 If a process establishes a signal-catching function for the SIGCHLD signal while it has a  
18328 terminated child process for which it has not waited, it is unspecified whether a SIGCHLD  
18329 signal is generated to indicate that child process.

18330 If the process is multi-threaded, or if the process is single-threaded and a signal handler is  
18331 executed other than as the result of:

- 18332 • The process calling *abort()*, *raise()*, *kill()*, *pthread\_kill()*, or *sigqueue()* to generate a signal  
18333 that is not blocked
- 18334 • A pending signal being unblocked and being delivered before the call that unblocked it  
18335 returns

18336 the behavior is undefined if:

- 18337 • The signal handler refers to any object other than *errno* with static or thread storage  
18338 duration that is not a lock-free atomic object, and not a non-modifiable object (for example,  
18339 string literals, objects that were defined with a const-qualified type, and objects in memory  
18340 that is mapped read-only), other than by assigning a value to an object declared as **volatile**  
18341 **sig\_atomic\_t**, unless the previous modification (if any) to the object happens before the  
18342 signal handler is called and the return from the signal handler happens before the next  
18343 modification (if any) to the object.

- 18344 • The signal handler calls any function or function-like macro defined in this standard other  
18345 than one of the functions and macros specified below as being `async-signal-safe`.

18346 The following table defines a set of functions and function-like macros that shall be `async-signal-`  
18347 `safe`. Therefore, applications can call them, without restriction, from `signal-catching` functions.  
18348 Note that, although there is no restriction on the calls themselves, for certain functions there are  
18349 restrictions on subsequent behavior after the function is called from a `signal-catching` function  
18350 (see `longjmp()`).

18351	<code>_Exit()</code>	<code>fstat()</code>	<code>mkdir()</code>	<code>setpgid()</code>
18352	<code>_Fork()</code>	<code>fstatat()</code>	<code>mkdirat()</code>	<code>setregid()</code>
18353	<code>_exit()</code>	<code>fsync()</code>	<code>mkfifo()</code>	<code>setresgid()</code>
18354	<code>abort()</code>	<code>ftruncate()</code>	<code>mkfifoat()</code>	<code>setresuid()</code>
18355	<code>accept()</code>	<code>futimens()</code>	<code>mknod()</code>	<code>setreuid()</code>
18356	<code>accept4()</code>	<code>getegid()</code>	<code>mknodat()</code>	<code>setsid()</code>
18357	<code>access()</code>	<code>geteuid()</code>	<code>ntohl()</code>	<code>setsockopt()</code>
18358	<code>aio_error()</code>	<code>getgid()</code>	<code>ntohs()</code>	<code>setuid()</code>
18359	<code>aio_return()</code>	<code>getgroups()</code>	<code>open()</code>	<code>shutdown()</code>
18360	<code>aio_suspend()</code>	<code>getpeername()</code>	<code>openat()</code>	<code>sig2str()</code>
18361	<code>alarm()</code>	<code>getpgrp()</code>	<code>pause()</code>	<code>sigaction()</code>
18362	<code>be16toh()</code>	<code>getpid()</code>	<code>pipe()</code>	<code>sigaddset()</code>
18363	<code>be32toh()</code>	<code>getppid()</code>	<code>pipe2()</code>	<code>sigdelset()</code>
18364	<code>be64toh()</code>	<code>getresgid()</code>	<code>poll()</code>	<code>sigemptyset()</code>
18365	<code>bind()</code>	<code>getresuid()</code>	<code>posix_close()</code>	<code>sigfillset()</code>
18366	<code>cfgetispeed()</code>	<code>getsockname()</code>	<code>ppoll()</code>	<code>sigismember()</code>
18367	<code>cfgetospeed()</code>	<code>getsockopt()</code>	<code>pread()</code>	<code>siglongjmp()</code>
18368	<code>cfsetispeed()</code>	<code>getuid()</code>	<code>pselect()</code>	<code>signal()</code>
18369	<code>cfsetospeed()</code>	<code>htobe16()</code>	<code>pthread_kill()</code>	<code>sigpending()</code>
18370	<code>chdir()</code>	<code>htobe32()</code>	<code>pthread_self()</code>	<code>sigprocmask()</code>
18371	<code>chmod()</code>	<code>htobe64()</code>	<code>pthread_setcancelstate()</code>	<code>sigqueue()</code>
18372	<code>chown()</code>	<code>htole16()</code>	<code>pthread_sigmask()</code>	<code>sigsuspend()</code>
18373	<code>clock_gettime()</code>	<code>htole32()</code>	<code>pwrite()</code>	<code>sleep()</code>
18374	<code>close()</code>	<code>htole64()</code>	<code>quick_exit()</code>	<code>socketmark()</code>
18375	<code>connect()</code>	<code>htonl()</code>	<code>raise()</code>	<code>socket()</code>
18376	<code>creat()</code>	<code>htons()</code>	<code>read()</code>	<code>socketpair()</code>
18377	<code>dup()</code>	<code>kill()</code>	<code>readv()</code>	<code>stat()</code>
18378	<code>dup2()</code>	<code>killpg()</code>	<code>readlink()</code>	<code>stpncpy()</code>
18379	<code>dup3()</code>	<code>le16toh()</code>	<code>readlinkat()</code>	<code>stpncpy()</code>
18380	<code>execl()</code>	<code>le32toh()</code>	<code>recv()</code>	<code>strcat()</code>
18381	<code>execle()</code>	<code>le64toh()</code>	<code>recvfrom()</code>	<code>strchr()</code>
18382	<code>execv()</code>	<code>link()</code>	<code>recvmsg()</code>	<code>strcmp()</code>
18383	<code>execve()</code>	<code>linkat()</code>	<code>rename()</code>	<code>strcpy()</code>
18384	<code>faccessat()</code>	<code>listen()</code>	<code>renameat()</code>	<code>strcspn()</code>
18385	<code>fchdir()</code>	<code>longjmp()</code>	<code>rmdir()</code>	<code>strlcat()</code>
18386	<code>fchmod()</code>	<code>lseek()</code>	<code>select()</code>	<code>strncpy()</code>
18387	<code>fchmodat()</code>	<code>lstat()</code>	<code>sem_post()</code>	<code>strlen()</code>
18388	<code>fchown()</code>	<code>memccpy()</code>	<code>send()</code>	<code>strncat()</code>
18389	<code>fchownat()</code>	<code>memchr()</code>	<code>sendmsg()</code>	<code>strncmp()</code>
18390	<code>fcntl()</code>	<code>memcmp()</code>	<code>sendto()</code>	<code>strncpy()</code>
18391	<code>fdatasync()</code>	<code>memcpy()</code>	<code>setegid()</code>	<code>strnlen()</code>
18392	<code>fexecve()</code>	<code>memmove()</code>	<code>seteuid()</code>	<code>strpbrk()</code>
18393	<code>ffs()</code>	<code>memset()</code>	<code>setgid()</code>	<code>strrchr()</code>

18394	<i>strspn()</i>	<i>time()</i>	<i>wait()</i>	<i>wcsncpy()</i>
18395	<i>strstr()</i>	<i>timer_getoverrun()</i>	<i>waitid()</i>	<i>wcsnlen()</i>
18396	<i>strtok_r()</i>	<i>timer_gettime()</i>	<i>waitpid()</i>	<i>wcspbrk()</i>
18397	<i>symlink()</i>	<i>timer_settime()</i>	<i>wcpcpy()</i>	<i>wcsrchr()</i>
18398	<i>symlinkat()</i>	<i>times()</i>	<i>wcpncpy()</i>	<i>wcsspn()</i>
18399	<i>tcdrain()</i>	<i>umask()</i>	<i>wcscat()</i>	<i>wcsstr()</i>
18400	<i>tcflow()</i>	<i>uname()</i>	<i>wcschr()</i>	<i>wcstok()</i>
18401	<i>tcflush()</i>	<i>unlink()</i>	<i>wcscmp()</i>	<i>wmemchr()</i>
18402	<i>tcgetattr()</i>	<i>unlinkat()</i>	<i>wscpy()</i>	<i>wmemcmp()</i>
18403	<i>tcgetpgrp()</i>	<i>utimensat()</i>	<i>wscspn()</i>	<i>wmemcpy()</i>
18404	<i>tcgetwinsize()</i>	<i>utimes()</i>	<i>wcslcat()</i>	<i>wmemmove()</i>
18405	<i>tcsendbreak()</i>	<i>va_arg()</i>	<i>wcslcpy()</i>	<i>wmemset()</i>
18406	<i>tcsetattr()</i>	<i>va_copy()</i>	<i>wcslen()</i>	<i>write()</i>
18407	<i>tcsetpgrp()</i>	<i>va_end()</i>	<i>wcsncat()</i>	<i>writev()</i>
18408	<i>tcsetwinsize()</i>	<i>va_start()</i>	<i>wcsncmp()</i>	

18409 In addition, the functions in `<stdatomic.h>` other than *atomic\_init()* shall be async-signal-safe  
 18410 when the atomic arguments are lock-free, and the *atomic\_is\_lock\_free()* function shall be async-  
 18411 signal-safe when called with an atomic argument.

18412 All other functions (including generic functions) and function-like macros may be unsafe with  
 18413 respect to signals. It is implementation-defined which additional interfaces, if any, are also  
 18414 async-signal-safe. In the presence of signals, all functions defined by this volume of  
 18415 POSIX.1-2024 shall behave as defined when called from or interrupted by a signal-catching  
 18416 function, with the exception that when a signal interrupts an unsafe function or function-like  
 18417 macro, or equivalent (such as the processing equivalent to *exit()* performed after a return from  
 18418 the initial call to *main()*), and the signal-catching function calls an unsafe function or function-  
 18419 like macro, the behavior is undefined. Additional exceptions are specified in the descriptions of  
 18420 individual functions such as *longjmp()*.

18421 Operations which obtain the value of *errno* and operations which assign a value to *errno* shall be  
 18422 async-signal-safe, provided that the signal-catching function saves the value of *errno* upon entry  
 18423 and restores it before it returns.

18424 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or  
 18425 continue, the entire process shall be terminated, stopped, or continued, respectively.

#### 18426 2.4.4 Signal Effects on Other Functions

18427 Signals affect the behavior of certain functions defined by this volume of POSIX.1-2024 if  
 18428 delivered to a process while it is executing such a function. If the action of the signal is to  
 18429 terminate the process, the process shall be terminated and the function shall not return. If the  
 18430 action of the signal is to stop the process, the process shall stop until continued or terminated.  
 18431 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the  
 18432 original function shall continue at the point the process was stopped. If the action of the signal is  
 18433 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case  
 18434 the original function is said to be “interrupted” by the signal. If the signal-catching function  
 18435 executes a **return** statement, the behavior of the interrupted function shall be as described  
 18436 individually for that function, except as noted for unsafe functions. After returning from a  
 18437 signal-catching function, the value of *errno* is unspecified if the signal-catching function or any  
 18438 function it called assigned a value to *errno* and the signal-catching function did not save and  
 18439 restore the original value of *errno*. Signals that are ignored shall not affect the behavior of any  
 18440 function; signals that are blocked shall not affect the behavior of any function until they are  
 18441 unblocked and then delivered, except as specified for the *sigpending()* and *sigwait()* functions.



## 18442 2.5 Standard I/O Streams

18443 CX A stream is associated with an external file (which may be a physical device) or memory buffer  
 18444 CX by “opening” a file or buffer. This may involve “creating” a new file. Creating an existing file  
 18445 causes its former contents to be discarded if necessary. If a file can support positioning requests  
 18446 (such as a disk file, as opposed to a terminal), then a “file position indicator” associated with the  
 18447 stream is positioned at the start (byte number 0) of the file, unless the file is opened with append  
 18448 mode, in which case it is implementation-defined whether the file position indicator is initially  
 18449 positioned at the beginning or end of the file. The file position indicator is maintained by  
 18450 subsequent reads, writes, and positioning requests, to facilitate an orderly progression through  
 18451 the file.

18452 The wide-character input functions shall read characters from the stream and convert them to  
 18453 wide characters as if they were read by successive calls to the *fgetwc()* function. Each conversion  
 18454 shall occur as if by a call to the *mbrtowc()* function, with the conversion state described by the  
 18455 stream’s own **mbstate\_t** object (see Section 2.5.2, on page 524). The byte input functions shall  
 18456 read characters from the stream as if by successive calls to the *fgetc()* function.

18457 The wide-character output functions shall convert wide characters to characters and write them  
 18458 to the stream as if they were written by successive calls to the *fputwc()* function. Each conversion  
 18459 shall occur as if by a call to the *wcrtomb()* function, with the conversion state described by the  
 18460 stream’s own **mbstate\_t** object (see Section 2.5.2, on page 524). The byte output functions shall  
 18461 write characters to the stream as if by successive calls to the *fputc()* function.

18462 The *perror()*, *psiginfo()*, and *psignal()* functions shall behave as described above for the byte  
 18463 output functions if the stream is already byte-oriented, and shall behave as described above for  
 18464 the wide-character output functions if the stream is already wide-oriented. If the stream has no  
 18465 orientation, they shall behave as described for the byte output functions except that they shall  
 18466 not change the orientation of the stream.

18467 Functions other than *perror()*, *psiginfo()*, and *psignal()* that write to streams but are neither wide-  
 18468 character output nor byte output functions (*getopt()* and *wordexp()*), shall behave as described  
 18469 above for the byte output functions, except that if the stream has no orientation, it is unspecified  
 18470 whether they set the stream to byte orientation or leave it with no orientation.

18471 When a stream is “unbuffered”, bytes are intended to appear from the source or at the  
 18472 destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a  
 18473 block. When a stream is “fully buffered”, bytes are intended to be transmitted as a block when a  
 18474 buffer is filled. When a stream is “line buffered”, bytes are intended to be transmitted as a block  
 18475 when a <newline> byte is encountered. Furthermore, bytes are intended to be transmitted as a  
 18476 block when a buffer is filled, when input is requested on an unbuffered stream, or when input is  
 18477 requested on a line-buffered stream that requires the transmission of bytes. Support for these  
 18478 characteristics is implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

18479 A file may be disassociated from a controlling stream by “closing” the file. Output streams are  
 18480 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from  
 18481 the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed  
 18482 (including the standard streams).

18483 A file may be subsequently reopened, by the same or another program execution, and its  
 18484 contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function  
 18485 returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all  
 18486 output streams are flushed) before program termination. Other paths to program termination,  
 18487 such as calling *abort()*, need not close all files properly.

18488 The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE**  
 18489 object need not necessarily serve in place of the original.

18490 At program start-up, three streams shall be predefined and already open: *stdin* (standard input,  
 18491 for conventional input) for reading, *stdout* (standard output, for conventional output) for  
 18492 writing, and *stderr* (standard error, for diagnostic output) for writing. When opened, *stderr* shall  
 18493 CX not be fully buffered; *stdin* and *stdout* shall be fully buffered if and only if the file descriptor  
 18494 associated with the stream is determined not to be associated with an interactive device.

18495 Each stream shall have an associated lock that is used to prevent data races when multiple  
 18496 threads of execution access a stream, and to restrict the interleaving of stream operations  
 18497 performed by multiple threads. Only one thread can hold this lock at a time. The lock shall be  
 18498 reentrant: a single thread can hold the lock multiple times at a given time. All functions that  
 18499 CX read, write, position, or query the position of a stream, except those with names ending  
 18500 CX *\_unlocked*, shall lock the stream as if by a call to *flockfile()* before accessing it and release the  
 18501 CX lock as if by a call to *funlockfile()* when the access is complete.

18502 CX If the lock is not immediately available, the function shall wait for it to become available, except  
 18503 in the following circumstances. If the stream is line buffered and is open for writing or for  
 18504 update, and the reason the function is attempting to lock the stream is because it is going to  
 18505 request input on another stream that is unbuffered, or is line buffered and requires the  
 18506 transmission of characters from the host environment (see above), then the function shall  
 18507 attempt to determine whether a deadlock situation exists. If a deadlock situation is found to  
 18508 exist, the function shall fail. If the function is able to establish that a deadlock situation does not  
 18509 exist, it shall wait for the lock to become available. If the function does not establish whether or  
 18510 not a deadlock situation exists, it shall continue as if it had already locked the stream, found its  
 18511 buffer to be empty, and released the lock.

18512 CX A stream associated with a memory buffer shall have the same operations for text files that a  
 18513 stream associated with an external file would have. In addition, the stream orientation shall be  
 18514 determined in exactly the same fashion.

18515 Input and output operations on a stream associated with a memory buffer by a call to  
 18516 *fmemopen()* shall be constrained by the implementation to take place within the bounds of the  
 18517 memory buffer. In the case of a stream opened by *open\_memstream()* or *open\_wmemstream()*, the  
 18518 memory area shall grow dynamically to accommodate write operations as necessary. For output,  
 18519 if the stream is fully buffered or line buffered, data shall be moved from the stream's internal  
 18520 buffer, or a buffer provided by *setvbuf()*, to the memory buffer during a flush or close operation.  
 18521 For input, it is unspecified whether a buffer provided by *setvbuf()* is used or whether read  
 18522 operations read directly from the memory buffer provided to or allocated by *fmemopen()*.

18523 When a standard I/O stream has an associated memory buffer (whether allocated internally,  
 18524 supplied to *setvbuf()*, or supplied to *fmemopen()*), the behavior is undefined if that buffer  
 18525 overlaps with the destination buffer passed to a call that reads from the stream or with the  
 18526 source buffer passed to a call that writes to the stream.

## 18527 2.5.1 Interaction of File Descriptors and Standard I/O Streams

18528 CX This section describes the interaction of file descriptors and standard I/O streams. The  
 18529 functionality described in this section is an extension to the ISO C standard (and the rest of this  
 18530 section is not further CX shaded).

18531 An open file description may be accessed through a file descriptor, which is created using  
 18532 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as  
 18533 *fopen()* or *popen()*. Either a file descriptor or a stream is called a "handle" on the open file  
 18534 description to which it refers; an open file description may have several handles.

18535 Handles can be created or destroyed by explicit user action, without affecting the underlying  
 18536 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,

18537 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

18538 A file descriptor that is never used in an operation that could affect the file offset (for example,  
18539 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to  
18540 one (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include  
18541 the file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is  
18542 not used directly by the application to affect the file offset. The *read()* and *write()* functions  
18543 implicitly affect the file offset; *lseek()* explicitly affects it.

18544 The result of function calls involving any one handle (the “active handle”) is defined elsewhere  
18545 in this volume of POSIX.1-2024, but if two or more handles are used, and any one of them is a  
18546 stream, the application shall ensure that their actions are coordinated as described below. If this  
18547 is not done, the result is undefined.

18548 A handle which is a stream is considered to be closed when either an *fclose()*, or *freopen()* with  
18549 non-null filename, is executed on it (for *freopen()* with a null filename, it is implementation-  
18550 defined whether a new handle is created or the existing one reused), or when the process  
18551 owning that stream terminates with *exit()*, *abort()*, or due to a signal. Several functions close file  
18552 descriptors, including *close()*, *dup2()*, *\_exit()*, the *exec* functions when *FD\_CLOEXEC* is set on a  
18553 file descriptor, *fork()* when *FD\_CLOFORK* is set on a file descriptor, and *posix\_spawn()* when  
18554 either *FD\_CLOEXEC* or *FD\_CLOFORK* is set.

18555 For a handle to become the active handle, the application shall ensure that the actions below are  
18556 performed between the last use of the handle (the current active handle) and the first use of the  
18557 second handle (the future active handle). The second handle then becomes the active handle. All  
18558 activity by the application affecting the file offset on the first handle shall be suspended until it  
18559 again becomes the active file handle. (If a stream function has as an underlying function one that  
18560 affects the file offset, the stream function shall be considered to affect the file offset.)

18561 The handles need not be in the same process for these rules to apply.

18562 Note that after a *fork()*, two handles exist where one existed before. The application shall ensure  
18563 that, if both handles can ever be accessed, they are both in a state where the other could become  
18564 the active handle first. The application shall prepare for a *fork()* exactly as if it were a change of  
18565 active handle. (If the only action performed by one of the processes is one of the *exec* functions or  
18566 *\_exit()* (not *exit()*), the handle is never accessed in that process.)

18567 For the first handle, the first applicable condition below applies. After the actions required  
18568 below are taken, if the handle is still open, the application can close it.

- 18569 • If it is a file descriptor, no action is required.
- 18570 • If the only further action to be performed on any handle to this open file descriptor is to  
18571 close it, no action need be taken.
- 18572 • If it is a stream which is unbuffered, no action need be taken.
- 18573 • If it is a stream which is line buffered, and the last byte written to the stream was a  
18574 <newline> (that is, as if a:

```
18575         putchar('\n')
```

18576 was the most recent operation on that stream), no action need be taken.

- 18577 • If it is a stream which is open for writing or appending (but not also open for reading), the  
18578 application shall either perform an *fflush()*, or the stream shall be closed.

18579 • If the stream is open for reading and it is at the end of the file (*feof()* is true), no action need  
18580 be taken.

18581 • If the stream is open with a mode that allows reading and the underlying open file  
18582 description refers to a device that is capable of seeking, the application shall either perform  
18583 an *fflush()*, or the stream shall be closed.

18584 For the second handle:

18585 • If any previous active handle has been used by a function that explicitly changed the file  
18586 offset, except as required above for the first handle, the application shall perform an *lseek()*  
18587 or *fseek()* (as appropriate to the type of handle) to an appropriate location.

18588 If the active handle ceases to be accessible before the requirements on the first handle, above,  
18589 have been met, the state of the open file description becomes undefined. This might occur  
18590 during functions such as a *fork()* or *\_exit()*.

18591 The *exec* functions make inaccessible all streams that are open at the time they are called,  
18592 independent of which streams or file descriptors may be available to the new process image.

18593 When these rules are followed, regardless of the sequence of handles used, no data shall be lost  
18594 or duplicated when writing, and all data shall be written in order, except as requested by seeks.  
18595 It is implementation-defined whether, and under what conditions, all input is seen exactly once.

18596 Each function that operates on a stream is said to have zero or more “underlying functions”.  
18597 This means that the stream function shares certain traits with the underlying functions, but does  
18598 not require that there be any relation between the implementations of the stream function and its  
18599 underlying functions.

## 18600 2.5.2 Stream Orientation and Encoding Rules

18601 The definition of a stream includes an “orientation”. After a stream is associated with an  
18602 external file, but before any operations are performed on it, the stream is without orientation.  
18603 Once a wide-character input/output function has been applied to a stream without orientation,  
18604 the stream shall become “wide-oriented”. Similarly, once a byte input/output function has been  
18605 applied to a stream without orientation, the stream shall become “byte-oriented”. Only a call to  
18606 the *freopen()* function or the *fwide()* function can otherwise alter the orientation of a stream.

18607 A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard*  
18608 *input*, *standard output*, and *standard error* shall be unoriented at program start-up.

18609 Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character  
18610 input/output functions cannot be applied to a byte-oriented stream. The remaining stream  
18611 operations shall not affect and shall not be affected by a stream’s orientation, except for the  
18612 following additional restriction:

18613 • For wide-oriented streams, after a successful call to a file-positioning function that leaves  
18614 the file position indicator prior to the end-of-file, a wide-character output function can  
18615 overwrite a partial character; any file contents beyond the byte(s) written are henceforth  
18616 undefined.

18617 CX Each wide-oriented stream that was not opened with *open\_wmemstream()* has an associated  
18618 **mbstate\_t** object that stores the current parse state of the stream. A successful call to *fgetpos()*  
18619 shall store a representation of the value of this **mbstate\_t** object as part of the value of the **fpos\_t**  
18620 object. A later successful call to *fsetpos()* using the same stored **fpos\_t** value shall restore the  
18621 value of the associated **mbstate\_t** object as well as the position within the controlled stream.

18622 Implementations that support multiple encoding rules associate an encoding rule with the

18623 stream. The encoding rule shall be determined by the setting of the `LC_CTYPE` category in the  
 18624 current locale at the time when the stream becomes wide-oriented. As with the stream's  
 18625 orientation, the encoding rule associated with a stream cannot be changed once it has been set,  
 18626 except by a successful call to `freopen()` which clears the encoding rule and resets the orientation  
 18627 to unoriented.

18628 Although wide-oriented streams are conceptually sequences of wide characters, the external file  
 18629 CX associated with a wide-oriented stream that was not opened with `open_wmemstream()` is a  
 18630 sequence of (possibly multi-byte) characters generalized as follows:

- 18631 • Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte  
 18632 encodings valid for use internal to the program).
- 18633 • A file need not begin nor end in the initial shift state.

18634 Moreover, the encodings used for characters may differ among files. Both the nature and choice  
 18635 of such encodings are implementation-defined.

18636 CX On streams that were not opened with `open_wmemstream()`, the wide-character input functions  
 18637 read characters from the stream and convert them to wide characters as if they were read by  
 18638 successive calls to the `fgetwc()` function. Each conversion shall occur as if by a call to the  
 18639 CX `mbrtowc()` function, with the conversion state described by the stream's own `mbstate_t` object,  
 18640 except the encoding rule associated with the stream is used instead of the encoding rule implied  
 18641 by the `LC_CTYPE` category of the current locale.

18642 CX On streams that were not opened with `open_wmemstream()`, the wide-character output functions  
 18643 convert wide characters to (possibly multi-byte) characters and write them to the stream as if  
 18644 they were written by successive calls to the `fputwc()` function. Each conversion shall occur as if  
 18645 by a call to the `wcrtomb()` function, with the conversion state described by the stream's own  
 18646 CX `mbstate_t` object, except the encoding rule associated with the stream is used instead of the  
 18647 encoding rule implied by the `LC_CTYPE` category of the current locale.

18648 An "encoding error" shall occur if the character sequence presented to the underlying `mbrtowc()`  
 18649 function does not form a valid (generalized) character, or if the code value passed to the  
 18650 underlying `wcrtomb()` function does not correspond to a valid (generalized) character. The wide-  
 18651 character input/output functions and the byte input/output functions store the value of the  
 18652 macro `[EILSEQ]` in `errno` if and only if an encoding error occurs.

## 18653 2.6 File Descriptor Allocation

18654 All functions that open one or more file descriptors shall, unless specified otherwise, atomically  
 18655 allocate the lowest numbered available (that is, not already open in the calling process) file  
 18656 descriptor at the time of each allocation. Where a single function allocates two file descriptors  
 18657 (for example, `pipe()` or `socketpair()`), the allocations may be independent and therefore  
 18658 applications should not expect them to have adjacent values or depend on which has the higher  
 18659 value.

## 18660 2.7 XSI Interprocess Communication

18661 XSI This section describes extensions to support interprocess communication. The functionality  
18662 described in this section shall be provided on implementations that support the XSI option (and  
18663 the rest of this section is not further shaded).

18664 The following message passing, semaphore, and shared memory services form an XSI  
18665 interprocess communication facility. Certain aspects of their operation are common, and are  
18666 defined as follows.

18667 IPC Functions		
18668 <i>msgctl()</i>	<i>semctl()</i>	<i>shmat()</i>
18669 <i>msgget()</i>	<i>semget()</i>	<i>shmctl()</i>
18670 <i>msgrcv()</i>	<i>semop()</i>	<i>shmdt()</i>
18671 <i>msgsnd()</i>		<i>shmget()</i>

18672 Another interprocess communication facility is provided by functions in the Realtime Option  
18673 Group; see Section 2.8 (on page 527).

### 18674 2.7.1 IPC General Description

18675 Each individual shared memory segment, message queue, and semaphore set shall be identified  
18676 by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a semaphore  
18677 identifier, *semid*, and a message queue identifier, *msqid*. The identifiers shall be returned by calls  
18678 to *shmget()*, *semget()*, and *msgget()*, respectively.

18679 Associated with each identifier is a data structure which contains data related to the operations  
18680 which may be or may have been performed; see the Base Definitions volume of POSIX.1-2024,  
18681 <sys/shm.h>, <sys/sem.h>, and <sys/msg.h> for their descriptions.

18682 Each of the data structures contains both ownership information and an **ipc\_perm** structure (see  
18683 the Base Definitions volume of POSIX.1-2024, <sys/ipc.h>) which are used in conjunction to  
18684 determine whether or not read/write (read/alter for semaphores) permissions should be  
18685 granted to processes using the IPC facilities. The *mode* member of the **ipc\_perm** structure acts as  
18686 a bit field which determines the permissions.

18687 The values of the bits are given below in octal notation along with the symbolic constants  
18688 defined in <sys/stat.h> that can be used to represent them.

18689 Octal Value	<sys/stat.h> Symbolic Constant	18690 Meaning
18691 0400	S_IRUSR	Read by user.
18692 0200	S_IWUSR	Write (for shared memory & message queues) or 18693 alter (for semaphores) by user.
18694 0040	S_IRGRP	Read by group.
18695 0020	S_IWGRP	Write or alter by group.
18696 0004	S_IROTH	Read by others.
18697 0002	S_IWOTH	Write or alter by others.

18698 The name of the **ipc\_perm** structure is *shm\_perm*, *sem\_perm*, or *msg\_perm*, depending on which  
18699 service is being used. In each case, read and write/alter permissions shall be granted to a  
18700 process if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as  
18701 appropriate):

- 18702 • The process has appropriate privileges.
- 18703 • The effective user ID of the process matches *xxx\_perm.cuid* or *xxx\_perm.uid* in the data
- 18704 structure associated with the IPC identifier, and the appropriate bit of the *user* field in
- 18705 *xxx\_perm.mode* is set.
- 18706 • The effective user ID of the process does not match *xxx\_perm.cuid* or *xxx\_perm.uid* but the
- 18707 effective group ID of the process matches *xxx\_perm.cgid* or *xxx\_perm.gid* in the data
- 18708 structure associated with the IPC identifier, and the appropriate bit of the *group* field in
- 18709 *xxx\_perm.mode* is set.
- 18710 • The effective user ID of the process does not match *xxx\_perm.cuid* or *xxx\_perm.uid* and the
- 18711 effective group ID of the process does not match *xxx\_perm.cgid* or *xxx\_perm.gid* in the data
- 18712 structure associated with the IPC identifier, but the appropriate bit of the *other* field in
- 18713 *xxx\_perm.mode* is set.
- 18714 Otherwise, the permission shall be denied.
- 18715 In addition to the **ipc\_perm** structure, each associated data structure includes several **time\_t**
- 18716 fields for recording timestamps of particular operations. When an operation is described as
- 18717 setting a timestamp to the current time, that particular timestamp member of the associated data
- 18718 structure shall be set to the largest **time\_t** value which is not greater than the current time.

## 18719 2.8 Realtime

18720 This section defines functions to support the source portability of applications with realtime  
 18721 requirements. The presence of some of these functions is dependent on support for  
 18722 implementation options described in the text.

18723 The specific functional areas included in this section and their scope include the following. Full  
 18724 definitions of these terms can be found in XBD [Chapter 3](#) (on page 31).

- 18725 • Semaphores
- 18726 • Process Memory Locking
- 18727 • Memory Mapped Files and Shared Memory Objects
- 18728 • Priority Scheduling
- 18729 • Realtime Signal Extension
- 18730 • Timers
- 18731 • Interprocess Communication
- 18732 • Synchronized Input and Output
- 18733 • Asynchronous Input and Output

18734 All the realtime functions defined in this volume of POSIX.1-2024 are portable, although some of  
 18735 the numeric parameters used by an implementation may have hardware dependencies.

18736 **2.8.1 Realtime Signals**18737 See [Section 2.4.2](#) (on page 515).18738 **2.8.2 Asynchronous I/O**

18739 An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O  
 18740 functions. It is defined in the Base Definitions volume of POSIX.1-2024, **<aio.h>** and has at least  
 18741 the following members:

Member Type	Member Name	Description
<b>int</b>	<i>aio_fildes</i>	File descriptor.
<b>off_t</b>	<i>aio_offset</i>	File offset.
<b>volatile void*</b>	<i>aio_buf</i>	Location of buffer.
<b>size_t</b>	<i>aio_nbytes</i>	Length of transfer.
<b>int</b>	<i>aio_reqprio</i>	Request priority offset.
<b>struct sigevent</b>	<i>aio_sigevent</i>	Signal number and value.
<b>int</b>	<i>aio_lio_opcode</i>	Operation to be performed.

18750 The *aio\_fildes* element is the file descriptor on which the asynchronous operation is performed.

18751 If `O_APPEND` is not set for the file descriptor *aio\_fildes* and if *aio\_fildes* is associated with a  
 18752 device that is capable of seeking, then the requested operation takes place at the absolute  
 18753 position in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to the  
 18754 operation with an *offset* argument equal to *aio\_offset* and a *whence* argument equal to `SEEK_SET`.  
 18755 If `O_APPEND` is set for the file descriptor, or if *aio\_fildes* is associated with a device that is  
 18756 incapable of seeking, write operations append to the file in the same order as the calls were  
 18757 made, with the following exception: under implementation-defined circumstances, such as  
 18758 operation on a multi-processor or when requests of differing priorities are submitted at the same  
 18759 time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming  
 18760 application to determine whether this relaxation applies, all strictly conforming applications  
 18761 which rely on ordering of output shall be written in such a way that they operate correctly if the  
 18762 relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the value  
 18763 of the file offset for the file is unspecified. The *aio\_nbytes* and *aio\_buf* elements are the same as the  
 18764 *nbyte* and *buf* arguments defined by *read()* and *write()*, respectively.

18765 If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then  
 18766 asynchronous I/O is queued in priority order, with the priority of each asynchronous operation  
 18767 based on the current scheduling priority of the calling process. The *aio\_reqprio* member can be  
 18768 used to lower (but not raise) the asynchronous I/O operation priority and is within the range  
 18769 zero through `{AIO_PRIO_DELTA_MAX}`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and  
 18770 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O  
 18771 requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and  
 18772 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted  
 18773 by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is  
 18774 unspecified. The priority of an asynchronous request is computed as (process scheduling  
 18775 priority) minus *aio\_reqprio*. The priority assigned to each asynchronous I/O request is an  
 18776 indication of the desired order of execution of the request relative to other asynchronous I/O  
 18777 requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same  
 18778 priority to a character special file are processed by the underlying device in FIFO order; the  
 18779 order of processing of requests of the same priority issued to files that are not character special  
 18780 files is unspecified. Numerically higher priority values indicate requests of higher priority. The  
 18781 value of *aio\_reqprio* has no effect on process scheduling priority. When prioritized asynchronous  
 18782 I/O requests to the same file are blocked waiting for a resource required for that I/O operation,



18783 the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests  
 18784 are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is  
 18785 implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall  
 18786 define for which files I/O prioritization is supported.

18787 The `aiio_sigevent` determines how the calling process shall be notified upon I/O completion, as  
 18788 specified in Section 2.4.1 (on page 513). If `aiio_sigevent.sigev_notify` is `SIGEV_NONE`, then no  
 18789 signal shall be posted upon I/O completion, but the error status for the operation and the return  
 18790 status for the operation shall be set appropriately.

18791 The `aiio_lio_opcode` field is used only by the `lio_listio()` call. The `lio_listio()` call allows multiple  
 18792 asynchronous I/O operations to be submitted at a single time. The function takes as an  
 18793 argument an array of pointers to `aiocb` structures. Each `aiocb` structure indicates the operation to  
 18794 be performed (read or write) via the `aiio_lio_opcode` field.

18795 The address of the `aiocb` structure is used as a handle for retrieving the error status and return  
 18796 status of the asynchronous operation while it is in progress.

18797 The `aiocb` structure and the data buffers associated with the asynchronous I/O operation are  
 18798 being used by the system for asynchronous I/O while, and only while, the error status of the  
 18799 asynchronous operation is equal to `[EINPROGRESS]`. Applications shall not modify the `aiocb`  
 18800 structure while the structure is being used by the system for asynchronous I/O.

18801 The return status of the asynchronous operation is the number of bytes transferred by the I/O  
 18802 operation. If the error status is set to indicate an error completion, then the return status is set to  
 18803 the return value that the corresponding `read()`, `write()`, or `fsync()` call would have returned.  
 18804 When the error status is not equal to `[EINPROGRESS]`, the return status shall reflect the return  
 18805 status of the corresponding synchronous operation.

## 18806 2.8.3 Memory Management

### 18807 2.8.3.1 Memory Locking

18808 MLR Range memory locking operations are defined in terms of pages. Implementations may restrict  
 18809 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes,  
 18810 is the value of the configurable system variable `{PAGESIZE}`. If an implementation has no  
 18811 restrictions on size or alignment, it may specify a 1-byte page size.

18812 ML|MLR Memory locking guarantees the residence of portions of the address space. It is implementation-  
 18813 defined whether locking memory guarantees fixed translation between virtual addresses (as  
 18814 seen by the process) and physical addresses. Per-process memory locks are not inherited across a  
 18815 `fork()`, and all memory locks owned by a process are unlocked upon `exec` or process termination.  
 18816 Unmapping of an address range removes any memory locks established on that address range  
 18817 by this process.

### 18818 2.8.3.2 Memory Mapped Files

18819 Range memory mapping operations are defined in terms of pages. Implementations may  
 18820 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,  
 18821 in bytes, is the value of the configurable system variable `{PAGESIZE}`. If an implementation has  
 18822 no restrictions on size or alignment, it may specify a 1-byte page size.

18823 Memory mapped files provide a mechanism that allows a process to access files by directly

18824 incorporating file data into its address space. Once a file is mapped into a process address space,  
 18825 the data can be manipulated as memory. If more than one process maps a file, its contents are  
 18826 shared among them. If the mappings allow shared write access, then data written into the  
 18827 memory object through the address space of one process appears in the address spaces of all  
 18828 processes that similarly map the same portion of the memory object.

18829 SHM Shared memory objects are named regions of storage that may be independent of the file system  
 18830 and can be mapped into the address space of one or more processes to allow them to share the  
 18831 associated memory.

18832 SHM An *unlink()* of a file or *shm\_unlink()* of a shared memory object, while causing the removal of  
 18833 the name, does not unmap any mappings established for the object. Once the name has been  
 18834 removed, the contents of the memory object are preserved as long as it is referenced. The  
 18835 memory object remains referenced as long as a process has the memory object open or has some  
 18836 area of the memory object mapped.

### 18837 2.8.3.3 Memory Protection

18838 When an object is mapped, various application accesses to the mapped region may result in  
 18839 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and  
 18840 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 18841 • A mapping may be restricted to disallow some types of access.
- 18842 • Write attempts to memory that was mapped without write access, or any access to  
 18843 memory mapped PROT\_NONE, shall result in a SIGSEGV signal.
- 18844 • References to unmapped addresses shall result in a SIGSEGV signal.
- 18845 • Reference to whole pages within the mapping, but beyond the current length of the object,  
 18846 shall result in a SIGBUS signal.
- 18847 • The size of the object is unaffected by access beyond the end of the object (even if a  
 18848 SIGBUS is not generated).

### 18849 2.8.3.4 Typed Memory Objects

18850 TYM The functionality described in this section shall be provided on implementations that support  
 18851 the Typed Memory Objects option (and the rest of this section is not further shaded for this  
 18852 option).

18853 Implementations may support the Typed Memory Objects option independently of support for  
 18854 memory mapped files or shared memory objects. Typed memory objects are implementation-  
 18855 configurable named storage pools accessible from one or more processors in a system, each via  
 18856 one or more ports, such as backplane buses, LANs, I/O channels, and so on. Each valid  
 18857 combination of a storage pool and a port is identified through a name that is defined at system  
 18858 configuration time, in an implementation-defined manner; the name may be independent of the  
 18859 file system. Using this name, a typed memory object can be opened and mapped into process  
 18860 address space. For a given storage pool and port, it is necessary to support both dynamic  
 18861 allocation from the pool as well as mapping at an application-supplied offset within the pool;  
 18862 when dynamic allocation has been performed, subsequent deallocation shall be supported.  
 18863 Lastly, accessing typed memory objects from different ports requires a method for obtaining the  
 18864 offset and length of contiguous storage of a region of typed memory (dynamically allocated or  
 18865 not); this allows typed memory to be shared among processes and/or processors while being  
 18866 accessed from the desired port.

**18867 2.8.4 Process Scheduling**

18868 PS The functionality described in this section shall be provided on implementations that support  
18869 the Process Scheduling option (and the rest of this section is not further shaded for this option).

**18870 Scheduling Policies**

18871 The scheduling semantics described in this volume of POSIX.1-2024 are defined in terms of a  
18872 conceptual model that contains a set of thread lists. No implementation structures are  
18873 necessarily implied by the use of this conceptual model. It is assumed that no time elapses  
18874 during operations described using this model, and therefore no simultaneous operations are  
18875 possible. This model discusses only processor scheduling for runnable threads, but it should be  
18876 noted that greatly enhanced predictability of realtime applications results if the sequencing of  
18877 other resources takes processor scheduling policy into account.

18878 There is, conceptually, one thread list for each priority. A runnable thread shall be on the thread  
18879 list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty  
18880 thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose  
18881 of a scheduling policy is to define the allowable operations on this set of lists (for example,  
18882 moving threads between and within lists).

18883 The POSIX model treats a "process" as an aggregation of system resources, including one or  
18884 more threads that may be scheduled by the operating system on the processor(s) it controls.  
18885 Although a process has its own set of scheduling attributes, these have an indirect effect (if any)  
18886 on the scheduling behavior of individual threads as described below.

18887 Each thread shall be controlled by an associated scheduling policy and priority. These  
18888 parameters may be specified by explicit application execution of the *pthread\_setschedparam()*  
18889 function. Additionally, the scheduling parameters of a thread (but not its scheduling policy) may  
18890 be changed by application execution of the *pthread\_setschedprio()* function.

18891 Each process shall be controlled by an associated scheduling policy and priority. These  
18892 parameters may be specified by explicit application execution of the *sched\_setscheduler()* or  
18893 *sched\_setparam()* functions.

18894 The effect of the process scheduling attributes on individual threads in the process is dependent  
18895 on the scheduling contention scope of the threads (see [Section 2.9.4](#), on page 540):

- 18896 • For threads with system scheduling contention scope, the process scheduling attributes  
18897 shall have no effect on the scheduling attributes or behavior either of the thread or an  
18898 underlying kernel scheduling entity dedicated to that thread.
- 18899 • For threads with process scheduling contention scope, the process scheduling attributes  
18900 shall have no effect on the scheduling attributes of the thread. However, any underlying  
18901 kernel scheduling entity used by these threads shall at all times behave as specified by the  
18902 scheduling attributes of the containing process, and this behavior may affect the  
18903 scheduling behavior of the process contention scope threads. For example, a process  
18904 contention scope thread with scheduling policy *SCHED\_FIFO* and the system maximum  
18905 priority *H* (the value returned by *sched\_get\_priority\_max(SCHED\_FIFO)*) in a process with  
18906 scheduling policy *SCHED\_RR* and system minimum priority *L* (the value returned by  
18907 *sched\_get\_priority\_min(SCHED\_RR)*) shall be subject to timeslicing and to preemption by  
18908 any thread with an effective priority higher than *L*.

18909 Associated with each policy is a priority range. Each policy definition shall specify the minimum  
18910 priority range for that policy. The priority ranges for each policy may but need not overlap the  
18911 priority ranges of other policies.

18912 A conforming implementation shall select the thread that is defined as being at the head of the

18913 highest priority non-empty thread list to become a running thread, regardless of its associated  
18914 policy. This thread is then removed from its thread list.

18915 Four scheduling policies are specifically required. Other implementation-defined scheduling  
18916 policies may be defined. The following symbols are defined in the Base Definitions volume of  
18917 POSIX.1-2024, <**sched.h**>:

18918 SCHED\_FIFO First in, first out (FIFO) scheduling policy.

18919 SCHED\_RR Round robin scheduling policy.

18920 SS SCHED\_SPORADIC Sporadic server scheduling policy.

18921 SCHED\_OTHER Another scheduling policy.

18922 The values of these symbols shall be distinct.

### 18923 SCHED\_FIFO

18924 Conforming implementations shall include a scheduling policy called the FIFO scheduling  
18925 policy.

18926 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its  
18927 threads have been on the list without being executed; generally, the head of the list is the thread  
18928 that has been on the list the longest time, and the tail is the thread that has been on the list the  
18929 shortest time.

18930 Under the SCHED\_FIFO policy, the modification of the definitional thread lists is as follows:

- 18931 1. When a running thread becomes a preempted thread, it becomes the head of the thread  
18932 list for its priority.
- 18933 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list  
18934 for its priority.
- 18935 3. When a running thread calls the *sched\_setscheduler()* function, the process specified in the  
18936 function call is modified to the specified policy and the priority specified by the *param*  
18937 argument.
- 18938 4. When a running thread calls the *sched\_setparam()* function, the priority of the process  
18939 specified in the function call is modified to the priority specified by the *param* argument.
- 18940 5. When a running thread calls the *pthread\_setschedparam()* function, the thread specified in  
18941 the function call is modified to the specified policy and the priority specified by the *param*  
18942 argument.
- 18943 6. When a running thread calls the *pthread\_setschedprio()* function, the thread specified in the  
18944 function call is modified to the priority specified by the *prio* argument.
- 18945 7. If a thread whose policy or priority has been modified other than by *pthread\_setschedprio()*  
18946 is a running thread or is runnable, it then becomes the tail of the thread list for its new  
18947 priority.
- 18948 8. If a thread whose priority has been modified by *pthread\_setschedprio()* is a running thread  
18949 or is runnable, the effect on its position in the thread list depends on the direction of the  
18950 modification, as follows:
  - 18951 a. If the priority is raised, the thread becomes the tail of the thread list.
  - 18952 b. If the priority is unchanged, the thread does not change position in the thread list.

- 18953 c. If the priority is lowered, the thread becomes the head of the thread list.
- 18954 9. When a running thread issues the *sched\_yield()* or *thrd\_yield()* function, the thread
- 18955 becomes the tail of the thread list for its priority.
- 18956 10. At no other time is the position of a thread with this scheduling policy within the thread
- 18957 lists affected.

18958 While a thread is executing at a temporarily elevated priority as a consequence of owning a

18959 mutex initialized with the PTHREAD\_PRIO\_INHERIT or PTHREAD\_PRIO\_PROTECT protocol

18960 (see *pthread\_mutexattr\_getprotocol()*), the effects of the above requirements on thread priority

18961 shall apply only to the thread's normal priority, not to its elevated priority, and those of the

18962 above requirements that describe the thread being placed on any thread list as a result of a

18963 priority change shall not apply. Likewise, when such a thread reverts to its normal priority as a

18964 consequence of unlocking such a mutex, those of the above requirements that describe the

18965 thread being placed on any thread list as a result of a priority change shall not apply.

18966 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_max()*

18967 and *sched\_get\_priority\_min()* functions when SCHED\_FIFO is provided as the parameter.

18968 Conforming implementations shall provide a priority range of at least 32 priorities for this

18969 policy.

## 18970 SCHED\_RR

18971 Conforming implementations shall include a scheduling policy called the "round robin"

18972 scheduling policy. This policy shall be identical to the SCHED\_FIFO policy with the additional

18973 condition that when the implementation detects that a running thread has been executing as a

18974 running thread for a time period of the length returned by the *sched\_rr\_get\_interval()* function or

18975 longer, the thread shall become the tail of its thread list and the head of that thread list shall be

18976 removed and made a running thread.

18977 The effect of this policy is to ensure that if there are multiple SCHED\_RR threads at the same

18978 priority, one of them does not monopolize the processor. An application should not rely only on

18979 the use of SCHED\_RR to ensure application progress among multiple threads if the application

18980 includes threads using the SCHED\_FIFO policy at the same or higher priority levels or

18981 SCHED\_RR threads at a higher priority level.

18982 A thread under this policy that is preempted and subsequently resumes execution as a running

18983 thread completes the unexpired portion of its round robin interval time period.

18984 For this policy, valid priorities shall be within the range returned by the *sched\_get\_priority\_max()*

18985 and *sched\_get\_priority\_min()* functions when SCHED\_RR is provided as the parameter.

18986 Conforming implementations shall provide a priority range of at least 32 priorities for this

18987 policy.

## 18988 SCHED\_SPORADIC

18989 SS|TSP The functionality described in this section shall be provided on implementations that support

18990 the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not

18991 further shaded for these options).

18992 If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the

18993 implementation shall include a scheduling policy identified by the value SCHED\_SPORADIC.

18994 The sporadic server policy is based primarily on two time parameters: the replenishment period

18995 and the available execution capacity. The replenishment period is given by the

18996 *sched\_ss\_repl\_period* member of the **sched\_param** structure. The available execution capacity is

18997 initialized to the value given by the *sched\_ss\_init\_budget* member of the same parameter. The

18998 sporadic server policy is identical to the SCHED\_FIFO policy with some additional conditions  
 18999 that cause the thread's assigned priority to be switched between the values specified by the  
 19000 *sched\_priority* and *sched\_ss\_low\_priority* members of the **sched\_param** structure.

19001 The priority assigned to a thread using the sporadic server scheduling policy is determined in  
 19002 the following manner: if the available execution capacity is greater than zero and the number of  
 19003 pending replenishment operations is strictly less than *sched\_ss\_max\_repl*, the thread is assigned  
 19004 the priority specified by *sched\_priority*; otherwise, the assigned priority shall be  
 19005 *sched\_ss\_low\_priority*. If the value of *sched\_priority* is less than or equal to the value of  
 19006 *sched\_ss\_low\_priority*, the results are undefined. When active, the thread shall belong to the  
 19007 thread list corresponding to its assigned priority level, according to the mentioned priority  
 19008 assignment. The modification of the available execution capacity and, consequently of the  
 19009 assigned priority, is done as follows:

- 19010 1. When the thread at the head of the *sched\_priority* list becomes a running thread, its  
 19011 execution time shall be limited to at most its available execution capacity, plus the  
 19012 resolution of the execution time clock used for this scheduling policy. This resolution shall  
 19013 be implementation-defined.
- 19014 2. Each time the thread is inserted at the tail of the list associated with *sched\_priority*—  
 19015 because as a blocked thread it became runnable with priority *sched\_priority* or because a  
 19016 replenishment operation was performed—the time at which this operation is done is  
 19017 posted as the *activation\_time*.
- 19018 3. When the running thread with assigned priority equal to *sched\_priority* becomes a  
 19019 preempted thread, it becomes the head of the thread list for its priority, and the execution  
 19020 time consumed is subtracted from the available execution capacity. If the available  
 19021 execution capacity would become negative by this operation, it shall be set to zero.
- 19022 4. When the running thread with assigned priority equal to *sched\_priority* becomes a blocked  
 19023 thread, the execution time consumed is subtracted from the available execution capacity,  
 19024 and a replenishment operation is scheduled, as described in 6 and 7. If the available  
 19025 execution capacity would become negative by this operation, it shall be set to zero.
- 19026 5. When the running thread with assigned priority equal to *sched\_priority* reaches the limit  
 19027 imposed on its execution time, it becomes the tail of the thread list for  
 19028 *sched\_ss\_low\_priority*, the execution time consumed is subtracted from the available  
 19029 execution capacity (which becomes zero), and a replenishment operation is scheduled, as  
 19030 described in 6 and 7.
- 19031 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be  
 19032 replenished, *replenish\_amount*, is set equal to the execution time consumed by the thread  
 19033 since the *activation\_time*. The replenishment is scheduled to occur at *activation\_time* plus  
 19034 *sched\_ss\_repl\_period*. If the scheduled time obtained is before the current time, the  
 19035 replenishment operation is carried out immediately. Several replenishment operations  
 19036 can be pending at the same time, each of which shall be serviced at its respective  
 19037 scheduled time. With the above rules, the number of replenishment operations  
 19038 simultaneously pending for a given thread that is scheduled under the sporadic server  
 19039 policy shall not be greater than *sched\_ss\_max\_repl*.
- 19040 7. A replenishment operation consists of adding the corresponding *replenish\_amount* to the  
 19041 available execution capacity at the scheduled time. If, as a consequence of this operation,  
 19042 the execution capacity would become larger than *sched\_ss\_initial\_budget*, it shall be  
 19043 rounded down to a value equal to *sched\_ss\_initial\_budget*. Additionally, if the thread was  
 19044 runnable or running, and had assigned priority equal to *sched\_ss\_low\_priority*, then it  
 19045 becomes the tail of the thread list for *sched\_priority*.

19046 Execution time is defined in XBD [Section 3.90](#) (on page 44).  
 19047 For this policy, changing the value of a CPU-time clock via `clock_settime()` shall have no effect on  
 19048 its behavior.  
 19049 For this policy, valid priorities shall be within the range returned by the `sched_get_priority_min()`  
 19050 and `sched_get_priority_max()` functions when `SCHED_SPORADIC` is provided as the parameter.  
 19051 Conforming implementations shall provide a priority range of at least 32 distinct priorities for  
 19052 this policy.  
 19053 If the scheduling policy of the target process is either `SCHED_FIFO` or `SCHED_RR`, the  
 19054 `sched_ss_low_priority`, `sched_ss_repl_period`, and `sched_ss_init` budget members of the `param`  
 19055 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process  
 19056 is not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC`, the effects of these members are  
 19057 implementation-defined; this case includes the `SCHED_OTHER` policy.

19058 **SCHED\_OTHER**

19059 Conforming implementations shall include one scheduling policy identified as `SCHED_OTHER`  
 19060 (which may execute identically with either the FIFO or round robin scheduling policy). The  
 19061 effect of scheduling threads with the `SCHED_OTHER` policy in a system in which other threads  
 19062 SS are executing under `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is implementation-  
 19063 defined.  
 19064 This policy is defined to allow strictly conforming applications to be able to indicate in a  
 19065 portable manner that they no longer need a realtime scheduling policy.  
 19066 For threads executing under this policy, the implementation shall use only priorities within the  
 19067 range returned by the `sched_get_priority_max()` and `sched_get_priority_min()` functions when  
 19068 `SCHED_OTHER` is provided as the parameter.

19069 **2.8.5 Clocks and Timers**

19070 The `<time.h>` header defines the types and manifest constants used by the timing facility.

19071 **Time Value Specification Structures**

19072 Many of the timing facility functions accept or return time value specifications. A time value  
 19073 structure `timespec` specifies a single time value and includes at least the following members:

Member Type	Member Name	Description
<code>time_t</code>	<code>tv_sec</code>	Seconds.
<code>long</code>	<code>tv_nsec</code>	Nanoseconds.

19077 The `tv_nsec` member is only valid if greater than or equal to zero, and less than the number of  
 19078 nanoseconds in a second (1 000 million). The time interval described by this structure is  $(tv\_sec * 10^9 + tv\_nsec)$  nanoseconds.  
 19079

19080 A time value structure `itimerspec` specifies an initial timer value and a repetition interval for use  
 19081 by the per-process timer functions. This structure includes at least the following members:

Member Type	Member Name	Description
<code>struct timespec</code>	<code>it_interval</code>	Timer period.
<code>struct timespec</code>	<code>it_value</code>	Timer expiration.

19085 If the value described by `it_value` is non-zero, it indicates the time to or time of the next timer  
 19086 expiration (for relative and absolute timer values, respectively). If the value described by `it_value`

19087 is zero, the timer shall be disarmed.

19088 If the value described by *it\_interval* is non-zero, it specifies an interval which shall be used in  
 19089 reloading the timer when it expires; that is, a periodic timer is specified. If the value described  
 19090 by *it\_interval* is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is  
 19091 specified.

### 19092 **Timer Event Notification Control Block**

19093 Per-process timers may be created that notify the process of timer expirations by queuing a  
 19094 realtime extended signal. The **sigevent** structure, defined in the Base Definitions volume of  
 19095 POSIX.1-2024, **<signal.h>**, is used in creating such a timer. The **sigevent** structure contains the  
 19096 signal number and an application-specific data value which shall be used when notifying the  
 19097 calling process of timer expiration events.

### 19098 **Manifest Constants**

19099 The following constants are defined in the Base Definitions volume of POSIX.1-2024, **<time.h>**:

19100 **CLOCK\_REALTIME** The identifier for the system-wide realtime clock.

19101 **TIMER\_ABSTIME** Flag indicating time is absolute with respect to the clock associated  
 19102 with a timer.

19103 **CLOCK\_MONOTONIC** The identifier for the system-wide monotonic clock, which is defined  
 19104 as a clock whose value cannot be set via *clock\_settime()* and which  
 19105 cannot have backward clock jumps. The maximum possible clock  
 19106 jump is implementation-defined.

19107 The maximum allowable resolution for **CLOCK\_REALTIME** and **CLOCK\_MONOTONIC** clocks  
 19108 and all time services based on these clocks is represented by **{\_POSIX\_CLOCKRES\_MIN}** and  
 19109 shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of  
 19110 resolution for these clocks to provide finer granularity time bases. The actual resolution  
 19111 supported by an implementation for a specific clock is obtained using the *clock\_getres()* function.  
 19112 If the actual resolution supported for a time service based on one of these clocks differs from the  
 19113 resolution supported for that clock, the implementation shall document this difference.

19114 The minimum allowable maximum value for **CLOCK\_REALTIME** and **CLOCK\_MONOTONIC**  
 19115 clocks and all absolute time services based on them is the same as that defined by the ISO C  
 19116 standard for the **time\_t** type. If the maximum value supported by a time service based on one of  
 19117 these clocks differs from the maximum value supported by that clock, the implementation shall  
 19118 document this difference.

### 19119 **Execution Time Monitoring**

19120 CPT If **\_POSIX\_CPUTIME** is defined, process CPU-time clocks shall be supported in addition to the  
 19121 clocks described in [Manifest Constants](#).

19122 TCT If **\_POSIX\_THREAD\_CPUTIME** is defined, thread CPU-time clocks shall be supported.

19123 CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in XBD [Section 3.90](#) (on page  
 19124 44). The mechanism used to measure execution time is described in XBD [Section 4.14](#) (on page  
 19125 99).

19126 CPT If **\_POSIX\_CPUTIME** is defined, the following constant of the type **clockid\_t** is defined in  
 19127 **<time.h>**:



19128 **CLOCK\_PROCESS\_CPUTIME\_ID**  
 19129 When this value of the type **clockid\_t** is used in a *clock()* or *timer\*()* function call, it is  
 19130 interpreted as the identifier of the CPU-time clock associated with the process making the  
 19131 function call.

19132 TCT If **\_POSIX\_THREAD\_CPUTIME** is defined, the following constant of the type **clockid\_t** is  
 19133 defined in **<time.h>**:

19134 **CLOCK\_THREAD\_CPUTIME\_ID**  
 19135 When this value of the type **clockid\_t** is used in a *clock()* or *timer\*()* function call, it is  
 19136 interpreted as the identifier of the CPU-time clock associated with the thread making the  
 19137 function call.

## 19138 2.9 Threads

19139 This section defines functionality to support multiple flows of control, called “threads”, within a  
 19140 process. For the definition of threads, see XBD [Section 3.388](#) (on page 88).

19141 The specific functional areas covered by threads and their scope include:

- 19142 • Thread management: the creation, control, and termination of multiple flows of control in  
 19143 the same process under the assumption of a common shared address space
- 19144 • Synchronization primitives optimized for tightly coupled operation of multiple control  
 19145 flows in a common, shared address space

### 19146 2.9.1 Thread-Safety

19147 All functions defined by this volume of POSIX.1-2024 shall be thread-safe, except that the  
 19148 following functions<sup>8</sup> need not be thread-safe.

---

19149 8. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

19150	<i>asctime()</i>	<i>endutxent()</i>	<i>getservent()</i>	<i>putenv()</i>
19151	<i>atomic_init()</i>	<i>getdate()</i>	<i>getutxent()</i>	<i>pututxline()</i>
19152	<i>catgets()</i>	<i>getgrent()</i>	<i>getutxid()</i>	<i>rand()</i>
19153	<i>crypt()</i>	<i>getgrgid()</i>	<i>getutxline()</i>	<i>setenv()</i>
19154	<i>ctime()</i>	<i>getgrnam()</i>	<i>gmtime()</i>	<i>setgrent()</i>
19155	<i>dbm_clearerr()</i>	<i>gethostent()</i>	<i>hcreate()</i>	<i>setkey()</i>
19156	<i>dbm_close()</i>	<i>getlogin()</i>	<i>hdestroy()</i>	<i>setlocale()</i>
19157	<i>dbm_delete()</i>	<i>getnetbyaddr()</i>	<i>hsearch()</i>	<i>setpwent()</i>
19158	<i>dbm_error()</i>	<i>getnetbyname()</i>	<i>inet_ntoa()</i>	<i>setutxent()</i>
19159	<i>dbm_fetch()</i>	<i>getnetent()</i>	<i>l64a()</i>	<i>srand()</i>
19160	<i>dbm_firstkey()</i>	<i>getopt()</i>	<i>localeconv()</i>	<i>strerror()</i>
19161	<i>dbm_nextkey()</i>	<i>getprotobyname()</i>	<i>localtime()</i>	<i>strsignal()</i>
19162	<i>dbm_open()</i>	<i>getprotobynumber()</i>	<i>lrand48()</i>	<i>strtok()</i>
19163	<i>dbm_store()</i>	<i>getprotoent()</i>	<i>mblen()</i>	<i>ttyname()</i>
19164	<i>dlerror()</i>	<i>getpwent()</i>	<i>mbtowc()</i>	<i>unsetenv()</i>
19165	<i>drand48()</i>	<i>getpwnam()</i>	<i>mrnd48()</i>	<i>wctomb()</i>
19166	<i>encrypt()</i>	<i>getpwuid()</i>	<i>nftw()</i>	
19167	<i>endgrent()</i>	<i>getservbyname()</i>	<i>nl_langinfo()</i>	
19168	<i>endpwent()</i>	<i>getservbyport()</i>	<i>ptsname()</i>	

19169 The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a null pointer argument.  
 19170 The *c16rtomb()*, *c32rtomb()*, *mbrlen()*, *mbrtoc16()*, *mbrtoc32()*, *mbrtowc()*, *mbsnrtoombs()*,  
 19171 *mbsrtombs()*, *wcrtomb()*, *wcsnrtoombs()*, and *wcsrtombs()* functions need not be thread-safe if  
 19172 passed a null *ps* argument. The *lgamma()*, *lgammaf()*, and *lgammal()* functions shall be thread-  
 19173 safe **except that they need not avoid data races when storing a value in the *signgam* variable.**  
 19174 The *getc\_unlocked()*, *getchar\_unlocked()*, *putc\_unlocked()*, and *putchar\_unlocked()* functions need  
 19175 not be thread-safe unless the invoking thread owns the (**FILE \***) object accessed by the call, as is  
 19176 the case after a successful call to the *flockfile()* or *ftrylockfile()* functions. The *readdir()* function  
 19177 need not be thread-safe if concurrent calls are made for the same directory stream.

19178 Some functions that are not required to be thread-safe are nevertheless required to avoid data  
 19179 races with either all or some other functions, as specified on their individual reference pages.

19180 Implementations shall provide internal synchronization as necessary in order to satisfy thread-  
 19181 safety requirements.

19182 Since multi-threaded applications are not allowed to use the *environ* variable to access or modify  
 19183 any environment variable while any other thread is concurrently modifying any environment  
 19184 variable, the *getenv()* and *secure\_getenv()* functions and any function dependent on any  
 19185 environment variable are not thread-safe if another thread modifies the environment; see XSH  
 19186 *exec* (on page 866).

## 19187 2.9.2 Thread IDs

19188 Although implementations may have thread IDs that are unique in a system, applications  
 19189 should only assume that thread IDs are usable and unique within a single process. The effect of  
 19190 calling any of the functions defined in this volume of POSIX.1-2024 and passing as an argument  
 19191 the thread ID of a thread from another process is unspecified. The lifetime of a thread ID ends  
 19192 after the later of thread termination (see [Section 3.392](#), on page 89) and the point when the  
 19193 thread is no longer joinable (see [Section 3.183](#), on page 58). A conforming implementation is free  
 19194 to reuse a thread ID after its lifetime has ended. If an application attempts to use a thread ID  
 19195 whose lifetime has ended, the behavior is undefined.

19196 If a thread is detached, its thread ID is invalid for use as an argument in a call to  
 19197 *pthread\_detach()*, *pthread\_join()*, *thrd\_detach()*, or *thrd\_join()*.

19198 **2.9.3 Thread Mutexes**

19199 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same  
 19200 processing resources from eventually making forward progress in its execution. Eligibility for  
 19201 processing resources is determined by the scheduling policy.

19202 A thread shall become the owner of a mutex, *m*, of type **pthread\_mutex\_t** when one of the  
 19203 following occurs:

- 19204 • It calls *pthread\_mutex\_clocklock()*, *pthread\_mutex\_lock()*, *pthread\_mutex\_timedlock()*, or  
 19205 *pthread\_mutex\_trylock()* with *m* as the *mutex* argument and the call returns zero or  
 19206 [EOWNERDEAD].
- 19207 • It calls *pthread\_mutex\_setprioceiling()* with *m* as the *mutex* argument and the call returns  
 19208 [EOWNERDEAD].
- 19209 • It calls *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* with *m* as  
 19210 the *mutex* argument and the call returns zero or certain error numbers (see  
 19211 *pthread\_cond\_clockwait()*).

19212 The thread shall remain the owner of *m* until one of the following occurs:

- 19213 • It executes *pthread\_mutex\_unlock()* with *m* as the *mutex* argument
- 19214 • It blocks in a call to *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or  
 19215 *pthread\_cond\_wait()* with *m* as the *mutex* argument.

19216 A thread shall become the owner of a mutex, *m*, of type **mtx\_t** when one of the following occurs:

- 19217 • It calls *mtx\_lock()* with *m* as the *mtx* argument and the call returns `thrd_success`.
- 19218 • It calls *mtx\_trylock()* with *m* as the *mtx* argument and the call returns `thrd_success`.
- 19219 • It calls *mtx\_timedlock()* with *m* as the *mtx* argument and the call returns `thrd_success`.
- 19220 • It calls *cond\_wait()* with *m* as the *mtx* argument and the call returns `thrd_success`.
- 19221 • It calls *cond\_timedwait()* with *m* as the *mtx* argument and the call returns `thrd_success` or  
 19222 `thrd_timedout`.

19223 The thread shall remain the owner of *m* until one of the following occurs:

- 19224 • It executes *mtx\_unlock()* with *m* as the *mtx* argument.
- 19225 • It blocks in a call to *cond\_wait()* with *m* as the *mtx* argument.
- 19226 • It blocks in a call to *cond\_timedwait()* with *m* as the *mtx* argument.

19227 The implementation shall behave as if at all times there is at most one owner of any mutex.

19228 A thread that becomes the owner of a mutex is said to have “acquired” the mutex and the mutex  
 19229 is said to have become “locked”; when a thread gives up ownership of a mutex it is said to have  
 19230 “released” the mutex and the mutex is said to have become “unlocked”.

19231 A problem can occur if a process terminates while one of its threads holds a mutex lock.  
 19232 Depending on the mutex type, it might be possible for another thread to unlock the mutex and  
 19233 recover the state of the mutex. However, it is difficult to perform this recovery reliably.

19234 Robust mutexes provide a means to enable the implementation to notify other threads in the  
 19235 event of a process terminating while one of its threads holds a lock on a mutex of type  
 19236 **pthread\_mutex\_t**. The next thread that acquires the mutex is notified about the termination by  
 19237 the return value [EOWNERDEAD] from the locking function. The notified thread can then  
 19238 attempt to recover the state protected by the mutex, and if successful mark the state protected by  
 19239 the mutex as consistent by a call to *pthread\_mutex\_consistent()*. If the notified thread is unable to

19240 recover the state, it can declare the state as not recoverable by a call to `pthread_mutex_unlock()`  
19241 without a prior call to `pthread_mutex_consistent()`.

19242 Whether or not the state protected by a mutex can be recovered is dependent solely on the  
19243 application using robust mutexes. The robust mutex support provided in the implementation  
19244 provides notification only that a mutex owner has terminated while holding a lock, or that the  
19245 state of the mutex is not recoverable.

## 19246 2.9.4 Thread Scheduling

19247 TPS The functionality described in this section shall be provided on implementations that support  
19248 the Thread Execution Scheduling option (and the rest of this section is not further shaded for  
19249 this option).

### 19250 Thread Scheduling Attributes

19251 In support of the scheduling function, threads have attributes which are accessed through the  
19252 **pthread\_attr\_t** thread creation attributes object.

19253 The *contentionscope* attribute defines the scheduling contention scope of the thread to be either  
19254 PTHREAD\_SCOPE\_PROCESS or PTHREAD\_SCOPE\_SYSTEM.

19255 The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling  
19256 attributes of the creating thread or to have its scheduling values set according to the other  
19257 scheduling attributes in the **pthread\_attr\_t** object.

19258 The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute  
19259 defines the scheduling parameters for the thread. The interaction of threads having different  
19260 policies within a process is described as part of the definition of those policies.

19261 If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one  
19262 of the priority-based policies defined under this option, the *schedparam* attribute contains the  
19263 scheduling priority of the thread. A conforming implementation ensures that the priority value  
19264 in *schedparam* is in the range associated with the scheduling policy when the thread attributes  
19265 object is used to create a thread, or when the scheduling attributes of a thread are dynamically  
19266 modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

19267 TSP If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four  
19268 new members that are used for the sporadic server scheduling policy. These members are  
19269 *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and *sched\_ss\_max\_repl*. The  
19270 meaning of these attributes is the same as in the definitions that appear under [Section 2.8.4](#) (on  
19271 page 531).

19272 When a process is created, its single thread has a scheduling policy and associated attributes  
19273 equal to the policy and attributes of the process. The default scheduling contention scope value  
19274 is implementation-defined. The default values of other scheduling attributes are  
19275 implementation-defined.

## 19276 Thread Scheduling Contention Scope

19277 The scheduling contention scope of a thread defines the set of threads with which the thread  
19278 competes for use of the processing resources. The scheduling operation selects at most one  
19279 thread to execute on each processor at any point in time and the thread's scheduling attributes  
19280 (for example, *priority*), whether under process scheduling contention scope or system scheduling  
19281 contention scope, are the parameters used to determine the scheduling decision.

19282 The scheduling contention scope, in the context of scheduling a mixed scope environment,  
19283 affects threads as follows:

- 19284 • A thread created with `PTHREAD_SCOPE_SYSTEM` scheduling contention scope contends  
19285 for resources with all other threads in the same scheduling allocation domain relative to  
19286 their system scheduling attributes. The system scheduling attributes of a thread created  
19287 with `PTHREAD_SCOPE_SYSTEM` scheduling contention scope are the scheduling  
19288 attributes with which the thread was created. The system scheduling attributes of a thread  
19289 created with `PTHREAD_SCOPE_PROCESS` scheduling contention scope are the  
19290 implementation-defined mapping into system attribute space of the scheduling attributes  
19291 with which the thread was created.
- 19292 • Threads created with `PTHREAD_SCOPE_PROCESS` scheduling contention scope contend  
19293 directly with other threads within their process that were created with  
19294 `PTHREAD_SCOPE_PROCESS` scheduling contention scope. The contention is resolved  
19295 based on the threads' scheduling attributes and policies. It is unspecified how such threads  
19296 are scheduled relative to threads in other processes or threads with  
19297 `PTHREAD_SCOPE_SYSTEM` scheduling contention scope.
- 19298 • Conforming implementations shall support the `PTHREAD_SCOPE_PROCESS` scheduling  
19299 contention scope, the `PTHREAD_SCOPE_SYSTEM` scheduling contention scope, or both.

## 19300 Scheduling Allocation Domain

19301 Implementations shall support scheduling allocation domains containing one or more  
19302 processors. It should be noted that the presence of multiple processors does not automatically  
19303 indicate a scheduling allocation domain size greater than one. Conforming implementations on  
19304 multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation  
19305 domains, and could define these scheduling allocation domains on a per-thread, per-process, or  
19306 per-system basis, depending on the types of applications intended to be supported by the  
19307 implementation. The scheduling allocation domain is independent of scheduling contention  
19308 scope, as the scheduling contention scope merely defines the set of threads with which a thread  
19309 contends for processor resources, while scheduling allocation domain defines the set of  
19310 processors for which it contends. The semantics of how this contention is resolved among  
19311 threads for processors is determined by the scheduling policies of the threads.

19312 The choice of scheduling allocation domain size and the level of application control over  
19313 scheduling allocation domains is implementation-defined. Conforming implementations may  
19314 change the size of scheduling allocation domains and the binding of threads to scheduling  
19315 allocation domains at any time.

19316 For application threads with scheduling allocation domains of size equal to one, the scheduling  
19317 rules defined for `SCHED_FIFO` and `SCHED_RR` shall be used; see [Scheduling Policies](#) (on page  
19318 531). All threads with system scheduling contention scope, regardless of the processes in which  
19319 they reside, compete for the processor according to their priorities. Threads with process  
19320 scheduling contention scope compete only with other threads with process scheduling  
19321 contention scope within their process.

19322 For application threads with scheduling allocation domains of size greater than one, the rules

19323 TSP defined for SCHED\_FIFO, SCHED\_RR, and SCHED\_SPORADIC shall be used in an  
 19324 implementation-defined manner. Each thread with system scheduling contention scope  
 19325 competes for the processors in its scheduling allocation domain in an implementation-defined  
 19326 manner according to its priority. Threads with process scheduling contention scope are  
 19327 scheduled relative to other threads within the same scheduling contention scope in the process.

19328 TSP If \_POSIX\_THREAD\_SPORADIC\_SERVER is defined, the rules defined for SCHED\_SPORADIC  
 19329 in [Scheduling Policies](#) (on page 531) shall be used in an implementation-defined manner for  
 19330 application threads whose scheduling allocation domain size is greater than one.

### 19331 Scheduling Documentation

19332 If \_POSIX\_PRIORITY\_SCHEDULING is defined, then any scheduling policies beyond  
 19333 TSP SCHED\_OTHER, SCHED\_FIFO, SCHED\_RR, and SCHED\_SPORADIC, as well as the effects of  
 19334 the scheduling policies indicated by these other values, and the attributes required in order to  
 19335 support such a policy, are implementation-defined. Furthermore, the implementation shall  
 19336 document the effect of all processor scheduling allocation domain values supported for these  
 19337 policies.

## 19338 2.9.5 Thread Cancellation

19339 The thread cancellation mechanism allows a thread to terminate the execution of any other  
 19340 thread in the process, except for threads created using *thrd\_create()*, in a controlled manner. The  
 19341 target thread (that is, the one that is being canceled) is allowed to hold cancellation requests  
 19342 pending in a number of ways and to perform application-specific cleanup processing when the  
 19343 notice of cancellation is acted upon.

19344 Cancellation is controlled by the cancellation control functions. Each thread maintains its own  
 19345 cancelability state. Cancellation may only occur at cancellation points or when the thread is  
 19346 asynchronously cancelable.

19347 The thread cancellation mechanism described in this section depends upon programs having set  
 19348 *deferred* cancelability state, which is specified as the default. Applications shall also carefully  
 19349 follow static lexical scoping rules in their execution behavior. For example, use of *setjmp()*,  
 19350 *return*, *goto*, and so on, to leave user-defined cancellation scopes without doing the necessary  
 19351 scope pop operation results in undefined behavior.

19352 Use of asynchronous cancelability while holding resources which potentially need to be released  
 19353 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated  
 19354 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

### 19355 2.9.5.1 Cancelability States

19356 The cancelability state of a thread determines the action taken upon receipt of a cancellation  
 19357 request. The thread may control cancellation in a number of ways.

19358 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 19359 1. Cancelability-Enable: When cancelability is PTHREAD\_CANCEL\_DISABLE (as defined  
 19360 in the Base Definitions volume of POSIX.1-2024, [<pthread.h>](#)), cancellation requests  
 19361 against the target thread are held pending. By default, cancelability is set to  
 19362 PTHREAD\_CANCEL\_ENABLE (as defined in [<pthread.h>](#)).
- 19363 2. Cancelability Type: When cancelability is enabled and the cancelability type is  
 19364 PTHREAD\_CANCEL\_ASYNCHRONOUS (as defined in [<pthread.h>](#)), new or pending  
 19365 cancellation requests may be acted upon at any time. When cancelability is enabled and

19366 the cancelability type is PTHREAD\_CANCEL\_DEFERRED (as defined in `<pthread.h>`),  
 19367 cancellation requests are held pending until a cancellation point (see below) is reached. If  
 19368 cancelability is disabled, the setting of the cancelability type has no immediate effect as all  
 19369 cancellation requests are held pending; however, once cancelability is enabled again the  
 19370 new type is in effect. The cancelability type is PTHREAD\_CANCEL\_DEFERRED in all  
 19371 newly created threads including the thread in which `main()` was first invoked.

#### 19372 2.9.5.2 Cancellation Points

19373 Cancellation points shall occur when a thread is executing the following functions:

19374	<code>accept()</code>	<code>msync()</code>	<code>recvfrom()</code>
19375	<code>accept4()</code>	<code>nanosleep()</code>	<code>recvmsg()</code>
19376	<code>aio_suspend()</code>	<code>open()</code>	<code>select()</code>
19377	<code>clock_nanosleep()</code>	<code>openat()</code>	<code>send()</code>
19378	<code>close()</code>	<code>pause()</code>	<code>sendmsg()</code>
19379	<code>cond_timedwait()</code>	<code>poll()</code>	<code>sendto()</code>
19380	<code>cond_wait()</code>	<code>posix_close()</code>	<code>sigsuspend()</code>
19381	<code>connect()</code>	<code>ppoll()</code>	<code>sigtimedwait()</code>
19382	<code>creat()</code>	<code>pread()</code>	<code>sigwait()</code>
19383	<code>fcntl()†</code>	<code>pselect()</code>	<code>sigwaitinfo()</code>
19384	<code>fdatasync()</code>	<code>pthread_cond_clockwait()</code>	<code>sleep()</code>
19385	<code>fsync()</code>	<code>pthread_cond_timedwait()</code>	<code>tcdrain()</code>
19386	<code>lockf()††</code>	<code>pthread_cond_wait()</code>	<code>thrd_join()</code>
19387	<code>mq_receive()</code>	<code>pthread_join()</code>	<code>thrd_sleep()</code>
19388	<code>mq_send()</code>	<code>pthread_testcancel()</code>	<code>wait()</code>
19389	<code>mq_timedreceive()</code>	<code>pwrite()</code>	<code>waitid()</code>
19390	<code>mq_timedsend()</code>	<code>read()</code>	<code>waitpid()</code>
19391	<code>msgrcv()</code>	<code>readv()</code>	<code>write()</code>
19392	<code>msgsnd()</code>	<code>recv()</code>	<code>writew()</code>

19393 A cancellation point may also occur when a thread is executing the following functions:

---

19394 † When the `cmd` argument is `F_SETLKW` or `F_OFD_SETLKW`.

19395 †† When the `function` argument is `F_LOCK`.

19396	<i>access()</i>	<i>endservent()</i>	<i>freopen()</i>
19397	<i>bindtextdomain()</i>	<i>faccessat()</i>	<i>fscanf()</i>
19398	<i>catclose()</i>	<i>fchmod()</i>	<i>fseek()</i>
19399	<i>catopen()</i>	<i>fchmodat()</i>	<i>fseeko()</i>
19400	<i>chmod()</i>	<i>fchown()</i>	<i>fsetpos()</i>
19401	<i>chown()</i>	<i>fchownat()</i>	<i>fstat()</i>
19402	<i>closedir()</i>	<i>fclose()</i>	<i>fstatat()</i>
19403	<i>closelog()</i>	<i>fcntl()†††</i>	<i>ftell()</i>
19404	<i>ctermid()</i>	<i>fflush()</i>	<i>ftello()</i>
19405	<i>dcgettext()</i>	<i>fgetc()</i>	<i>futimens()</i>
19406	<i>dcgettext_l()</i>	<i>fgetpos()</i>	<i>fwprintf()</i>
19407	<i>dcngettext()</i>	<i>fgets()</i>	<i>fwrite()</i>
19408	<i>dcngettext_l()</i>	<i>fgetwc()</i>	<i>fwscanf()</i>
19409	<i>dgettext()</i>	<i>fgetws()</i>	<i>getaddrinfo()</i>
19410	<i>dgettext_l()</i>	<i>fntmsg()</i>	<i>getc()</i>
19411	<i>dlclose()</i>	<i>fopen()</i>	<i>getc_unlocked()</i>
19412	<i>dlopen()</i>	<i>fpathconf()</i>	<i>getchar()</i>
19413	<i>dngettext()</i>	<i>fprintf()</i>	<i>getchar_unlocked()</i>
19414	<i>dngettext_l()</i>	<i>fputc()</i>	<i>getcwd()</i>
19415	<i>dprintf()</i>	<i>fputs()</i>	<i>getdelim()</i>
19416	<i>endhostent()</i>	<i>fputwc()</i>	<i>getgrgid_r()</i>
19417	<i>endnetent()</i>	<i>fputws()</i>	<i>getgrnam_r()</i>
19418	<i>endprotoent()</i>	<i>fread()</i>	<i>gethostid()</i>

---

19419 ††† For any value of the *cmd* argument.



19420	<i>gethostname()</i>	<i>posix_fadvise()</i>	<i>sem_clockwait()</i>
19421	<i>getline()</i>	<i>posix_fallocate()</i>	<i>sem_timedwait()</i>
19422	<i>getlogin_r()</i>	<i>posix_getdents()</i>	<i>sem_wait()</i>
19423	<i>getnameinfo()</i>	<i>posix_madvise()</i>	<i>semop()</i>
19424	<i>getpwnam_r()</i>	<i>posix_openpt()</i>	<i>sethostent()</i>
19425	<i>getpwuid_r()</i>	<i>posix_spawn()</i>	<i>setnetent()</i>
19426	<i>gettext()</i>	<i>posix_spawnnp()</i>	<i>setprotoent()</i>
19427	<i>gettext_l()</i>	<i>posix_typed_mem_open()</i>	<i>setservent()</i>
19428	<i>getwc()</i>	<i>printf()</i>	<i>stat()</i>
19429	<i>getwchar()</i>	<i>psiginfo()</i>	<i>strerror_l()</i>
19430	<i>glob()</i>	<i>psignal()</i>	<i>strerror_r()</i>
19431	<i>iconv_close()</i>	<i>pthread_rwlock_clockrdlock()</i>	<i>strftime()</i>
19432	<i>iconv_open()</i>	<i>pthread_rwlock_clockwrlock()</i>	<i>strftime_l()</i>
19433	<i>link()</i>	<i>pthread_rwlock_rdlock()</i>	<i>symlink()</i>
19434	<i>linkat()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>symlinkat()</i>
19435	<i>lio_listio()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>sync()</i>
19436	<i>localtime_r()</i>	<i>pthread_rwlock_wrlock()</i>	<i>syslog()</i>
19437	<i>lockf()</i>	<i>ptsname()</i>	<i>tmpfile()</i>
19438	<i>lseek()</i>	<i>ptsname_r()</i>	<i>tmpnam()</i>
19439	<i>lstat()</i>	<i>putc()</i>	<i>ttyname_r()</i>
19440	<i>mkdir()</i>	<i>putc_unlocked()</i>	<i>tzset()</i>
19441	<i>mkdirat()</i>	<i>putchar()</i>	<i>ungetc()</i>
19442	<i>mkdtemp()</i>	<i>putchar_unlocked()</i>	<i>ungetwc()</i>
19443	<i>mkfifo()</i>	<i>puts()</i>	<i>unlink()</i>
19444	<i>mkfifoat()</i>	<i>putwc()</i>	<i>unlinkat()</i>
19445	<i>mknod()</i>	<i>putwchar()</i>	<i>utimensat()</i>
19446	<i>mknodat()</i>	<i>readdir_r()</i>	<i>utimes()</i>
19447	<i>mkstemp()</i>	<i>readlink()</i>	<i>vdprintf()</i>
19448	<i>mktime()</i>	<i>readlinkat()</i>	<i>vfprintf()</i>
19449	<i>ngettext()</i>	<i>remove()</i>	<i>vfwprintf()</i>
19450	<i>ngettext_l()</i>	<i>rename()</i>	<i>vprintf()</i>
19451	<i>opendir()</i>	<i>renameat()</i>	<i>vwprintf()</i>
19452	<i>openlog()</i>	<i>rewind()</i>	<i>wcsftime()</i>
19453	<i>pathconf()</i>	<i>rewinddir()</i>	<i>wordexp()</i>
19454	<i>perror()</i>	<i>scandir()</i>	<i>wprintf()</i>
19455	<i>popen()</i>	<i>scanf()</i>	<i>wscanf()</i>
19456	<i>posix_devctl()</i>	<i>seekdir()</i>	

19457 In addition, a cancellation point may occur when a thread is executing any function that this  
 19458 standard does not require to be thread-safe but the implementation documents as being thread-  
 19459 safe. If a thread is cancelled while executing a non-thread-safe function, the behavior is  
 19460 undefined.

19461 An implementation shall not introduce cancellation points into any other functions specified in  
 19462 this volume of POSIX.1-2024.

19463 The side-effects of acting upon a cancellation request while suspended during a call of a function  
 19464 are the same as the side-effects that may be seen in a single-threaded program when a call to a  
 19465 function is interrupted by a signal and the given function returns [EINTR]. Any such side-  
 19466 effects occur before any cancellation cleanup handlers are called. For functions that are explicitly  
 19467 required not to return when interrupted (for example, *pclose()*), if a thread is canceled while  
 19468 executing the function, the behavior is undefined.

19469 Whenever a thread has cancelability enabled and a cancellation request has been made with that  
 19470 thread as the target, and the thread then calls any function that is a cancellation point (such as

19471 *pthread\_testcancel()* or *read()*, the cancellation request shall be acted upon before the function  
 19472 returns. If a thread has cancelability enabled and a cancellation request is made with the thread  
 19473 as a target while the thread is suspended at a cancellation point, the thread shall be awakened  
 19474 and the cancellation request shall be acted upon. It is unspecified whether the cancellation  
 19475 request is acted upon or whether the cancellation request remains pending and the thread  
 19476 resumes normal execution if the thread is suspended at a cancellation point and either:

- 19477 • The event for which it is waiting occurs
  - 19478 • A specified timeout expires
- 19479 before the cancellation request is acted upon.

### 19480 2.9.5.3 Thread Cancellation Cleanup Handlers

19481 Each thread that was not created using *thrd\_create()* maintains a list of cancellation cleanup  
 19482 handlers. The programmer uses the *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions  
 19483 to place routines on and remove routines from this list.

19484 When a cancellation request is acted upon, or when a thread calls *pthread\_exit()*, the thread first  
 19485 disables cancellation by setting its cancelability state to `PTHREAD_CANCEL_DISABLE` and its  
 19486 cancelability type to `PTHREAD_CANCEL_DEFERRED`. The cancelability state shall remain set  
 19487 to `PTHREAD_CANCEL_DISABLE` until the thread has terminated. The behavior is undefined if  
 19488 a cancellation cleanup handler or thread-specific data destructor routine changes the  
 19489 cancelability state to `PTHREAD_CANCEL_ENABLE`.

19490 The routines in the thread's list of cancellation cleanup handlers shall be invoked one by one in  
 19491 LIFO sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked  
 19492 (First Out). When the cancellation cleanup handler for a scope is invoked, the storage for that  
 19493 scope remains valid. If the last cancellation cleanup handler returns, thread-specific data  
 19494 destructors (if any) associated with thread-specific data keys for which the thread has non-  
 19495 NULL values shall be run, in unspecified order, as described for *pthread\_key\_create()* and  
 19496 *tss\_create()*.

19497 After all cancellation cleanup handlers and thread-specific data destructors have returned,  
 19498 thread execution is terminated. If the thread has terminated because of a call to *pthread\_exit()*,  
 19499 the *value\_ptr* argument is made available to any threads joining with the target. If the thread has  
 19500 terminated by acting on a cancellation request, a status of `PTHREAD_CANCELED` is made  
 19501 available to any threads joining with the target. The symbolic constant `PTHREAD_CANCELED`  
 19502 expands to a constant expression of type `(void *)` whose value matches no pointer to an object in  
 19503 memory nor the value `NULL`.

19504 A side-effect of acting upon a cancellation request while in a condition variable wait is that the  
 19505 mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread  
 19506 is no longer considered to be waiting for the condition and the thread shall not have consumed  
 19507 any pending condition signals on the condition.

19508 A cancellation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

### 19509 2.9.5.4 Async-Cancel Safety

19510 The *pthread\_cancel()*, *pthread\_setcancelstate()*, and *pthread\_setcanceltype()* functions are defined to  
 19511 be async-cancel safe.

19512 No other functions in this volume of POSIX.1-2024 are required to be async-cancel-safe.

19513 If a thread has asynchronous cancellation enabled and is cancelled during execution of a

19514 function that is not async-cancel-safe, the behavior is undefined.

19515 If a thread has deferred cancellation enabled, a signal-catching function is called in that thread  
 19516 during execution of a function that is not async-cancel-safe, and the signal-catching function  
 19517 calls any function that is a cancellation point while a cancellation is pending for the thread,  
 19518 without first disabling cancellation, the behavior is undefined.

## 19519 2.9.6 Thread Read-Write Locks

19520 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous  
 19521 read-only access to data while allowing only one thread to have exclusive write access at any  
 19522 given time. They are typically used to protect data that is read more frequently than it is  
 19523 changed.

19524 One or more readers acquire read access to the resource by performing a read lock operation on  
 19525 the associated read-write lock. A writer acquires exclusive write access by performing a write  
 19526 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and  
 19527 any other writers.

19528 A thread that has blocked on a read-write lock (for example, has not yet returned from a  
 19529 *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_wrlock()* call) shall not prevent any unblocked thread  
 19530 that is eligible to use the same processing resources from eventually making forward progress in  
 19531 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

19532 Read-write locks can be used to synchronize threads in the current process and other processes if  
 19533 they are allocated in memory that is writable and shared among the cooperating processes and  
 19534 have been initialized for this behavior.

## 19535 2.9.7 Thread Interactions with File Operations

19536 All of the following functions shall be atomic with respect to each other in the effects specified in  
 19537 POSIX.1-2024 when they operate on files in the file hierarchy:

19538	<i>chmod()</i>	<i>fchownat()</i>	<i>link()</i>	<i>readlinkat()</i>	<i>truncate()</i>
19539	<i>chown()</i>	<i>fstat()</i>	<i>linkat()</i>	<i>rename()</i>	<i>unlink()</i>
19540	<i>creat()</i>	<i>fstatat()</i>	<i>lstat()</i>	<i>renameat()</i>	<i>unlinkat()</i>
19541	<i>fchmod()</i>	<i>ftruncate()</i>	<i>open()</i>	<i>stat()</i>	<i>utimensat()</i>
19542	<i>fchmodat()</i>	<i>futimens()</i>	<i>openat()</i>	<i>symlink()</i>	<i>utimes()</i>
19543	<i>fchown()</i>	<i>lchown()</i>	<i>readlink()</i>	<i>symlinkat()</i>	

19544 If two threads each call one of these functions, each call shall either see all of the specified effects  
 19545 of the other call, or none of them.

19546 Except where specified otherwise, all of the following functions shall be atomic with respect to  
 19547 each other in the effects specified in POSIX.1-2024 when they operate on file descriptors that are  
 19548 open, or being opened, to files in the file hierarchy:

19549	<i>close()</i>	<i>fstat()</i>	<i>lseek()</i>	<i>read()</i>	<i>writev()</i>
19550	<i>dup2()</i>	<i>fstatat()</i>	<i>open()</i>	<i>readv()</i>	
19551	<i>dup3()</i>	<i>ftruncate()</i>	<i>openat()</i>	<i>pwrite()</i>	
19552	<i>fcntl()</i>	<i>futimens()</i>	<i>pread()</i>	<i>write()</i>	

19553 If two threads each call one of these functions, each call shall either see all of the specified effects  
 19554 of the other call, or none of them. The requirement on the *close()* function shall also apply  
 19555 whenever a file descriptor is successfully closed, however caused (for example, as a consequence

19556 of calling *close()*, calling *dup2()*, or of process termination).

## 19557 2.9.8 Use of Application-Managed Thread Stacks

19558 An “application-managed thread stack” is a region of memory allocated by the application—for  
 19559 example, memory returned by the *malloc()* or *mmap()* functions—and designated as a stack  
 19560 through the act of passing the address and size of the stack, respectively, as the *stackaddr* and  
 19561 *stacksize* arguments to *pthread\_attr\_setstack()*. Application-managed stacks allow the application  
 19562 to precisely control the placement and size of a stack.

19563 The application grants to the implementation permanent ownership of and control over the  
 19564 application-managed stack when the attributes object in which the *stack* or *stackaddr* attribute has  
 19565 been set is used, either by presenting that attribute’s object as the *attr* argument in a call to  
 19566 *pthread\_create()* that completes successfully, or by storing a pointer to the attributes object in the  
 19567 *sigev\_notify\_attributes* member of a **struct sigevent** and passing that **struct sigevent** to a function  
 19568 accepting such argument that completes successfully. The application may thereafter utilize the  
 19569 memory within the stack only within the normal context of stack usage within or properly  
 19570 synchronized with a thread that has been scheduled by the implementation with stack pointer  
 19571 value(s) that are within the range of that stack. In particular, the region of memory cannot be  
 19572 freed, nor can it be later specified as the stack for another thread.

19573 When specifying an attributes object with an application-managed stack through the  
 19574 *sigev\_notify\_attributes* member of a **struct sigevent**, the results are undefined if the requested  
 19575 signal is generated multiple times (as for a repeating timer).

19576 Until an attributes object in which the *stack* or *stackaddr* attribute has been set is used, the  
 19577 application retains ownership of and control over the memory allocated to the stack. It may free  
 19578 or reuse the memory as long as it either deletes the attributes object, or before using the  
 19579 attributes object replaces the stack by making an additional call to *pthread\_attr\_setstack()*, that  
 19580 was used originally to designate the stack. There is no mechanism to retract the reference to an  
 19581 application-managed stack by an existing attributes object.

19582 Once an attributes object with an application-managed stack has been used, that attributes object  
 19583 cannot be used again by a subsequent call to *pthread\_create()* or any function accepting a **struct**  
 19584 **sigevent** with *sigev\_notify\_attributes* containing a pointer to the attributes object, without  
 19585 designating an unused application-managed stack by making an additional call to  
 19586 *pthread\_attr\_setstack()*.

## 19587 2.9.9 Synchronization Object Copies and Alternative Mappings

19588 TSH For barriers, condition variables, mutexes, and read-write locks, if the process-shared attribute  
 19589 is set to `PTHREAD_PROCESS_PRIVATE`, only the synchronization object at the address used to  
 19590 initialize it can be used for performing synchronization. The effect of referring to another  
 19591 TSH mapping of the same object when locking, unlocking, or destroying the object is undefined. If  
 19592 the process-shared attribute is set to `PTHREAD_PROCESS_SHARED`, only the synchronization  
 19593 object itself can be used for performing synchronization; however, it need not be referenced at  
 19594 the address used to initialize it (that is, another mapping of the same object can be used). The  
 19595 effect of referring to a copy of the object when locking, unlocking, or destroying it is undefined.

19596 For spin locks, the above requirements shall apply as if spin locks have a process-shared  
 19597 attribute that is set from the *pshared* argument to *pthread\_spin\_init()*. For semaphores, the above  
 19598 requirements shall apply as if semaphores have a process-shared attribute that is set to  
 19599 `PTHREAD_PROCESS_PRIVATE` if the *pshared* argument to *sem\_init()* is zero and set to  
 19600 `PTHREAD_PROCESS_SHARED` if *pshared* is non-zero.

19601 For ISO C functions declared in `<threads.h>`, the above requirements shall apply as if condition  
19602 variables of type `cond_t` and mutexes of type `mtx_t` have a process-shared attribute that is set to  
19603 `PTHREAD_PROCESS_PRIVATE`.

## 19604 2.10 Sockets

19605 A socket is an endpoint for communication using the facilities described in this section. A socket  
19606 is created with a specific socket type, described in [Section 2.10.6](#) (on page 550), and is associated  
19607 with a specific protocol, detailed in [Section 2.10.3](#). A socket is accessed via a file descriptor  
19608 obtained when the socket is created.

### 19609 2.10.1 Address Families

19610 All network protocols are associated with a specific address family. An address family provides  
19611 basic services to the protocol implementation to allow it to function within a specific network  
19612 environment. These services may include packet fragmentation and reassembly, routing,  
19613 addressing, and basic transport. An address family is normally comprised of a number of  
19614 protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not  
19615 required that an address family support all socket types. An address family may contain  
19616 multiple protocols supporting the same socket abstraction.

19617 [Section 2.10.17](#) (on page 557), [Section 2.10.19](#) (on page 558), and [Section 2.10.20](#) (on page 558),  
19618 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based  
19619 on IPv4, and for Internet protocols based on IPv6.

### 19620 2.10.2 Addressing

19621 An address family defines the format of a socket address. All network addresses are described  
19622 using a general structure, called a `sockaddr`, as defined in the Base Definitions volume of  
19623 POSIX.1-2024, `<sys/socket.h>`. However, each address family imposes finer and more specific  
19624 structure, generally defining a structure with fields specific to the address family. The field  
19625 `sa_family` in the `sockaddr` structure contains the address family identifier, specifying the format  
19626 of the `sa_data` area. The size of the `sa_data` area is unspecified.

### 19627 2.10.3 Protocols

19628 A protocol supports one of the socket abstractions detailed in [Section 2.10.6](#) (on page 550).  
19629 Selecting a protocol involves specifying the address family, socket type, and protocol number to  
19630 the `socket()` function. Certain semantics of the basic socket abstractions are protocol-specific. All  
19631 protocols are expected to support the basic model for their particular socket type, but may, in  
19632 addition, provide non-standard facilities or extensions to a mechanism.

**19633 2.10.4 Routing**

19634 Sockets provides packet routing facilities. A routing information database is maintained, which  
19635 is used in selecting the appropriate network interface when transmitting packets.

**19636 2.10.5 Interfaces**

19637 Each network interface in a system corresponds to a path through which messages can be sent  
19638 and received. A network interface usually has a hardware device associated with it, though  
19639 certain interfaces such as the loopback interface, do not.

**19640 2.10.6 Socket Types**

19641 A socket is created with a specific type, which defines the communication semantics and which  
19642 allows the selection of an appropriate communication protocol. Four types are defined:  
19643 RS SOCK\_DGRAM, SOCK\_RAW, SOCK\_SEQPACKET, and SOCK\_STREAM. Implementations  
19644 may specify additional socket types.

19645 The SOCK\_STREAM socket type provides reliable, sequenced, full-duplex octet streams  
19646 between the socket and a peer to which the socket is connected. A socket of type  
19647 SOCK\_STREAM needs to be in a connected state before any data can be sent or received. Record  
19648 boundaries are not maintained; data sent on a stream socket using output operations of one size  
19649 can be received using input operations of smaller or larger sizes without loss of data. Data may  
19650 be buffered; successful return from an output function does not imply that the data has been  
19651 delivered to the peer or even transmitted from the local system. If data cannot be successfully  
19652 transmitted within a given time then the connection is considered broken, and subsequent  
19653 operations shall fail. A SIGPIPE signal is raised if a thread attempts to send data on a broken  
19654 stream (one that is no longer connected), except that the signal is suppressed if the  
19655 MSG\_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Support for an out-of-  
19656 band data transmission facility is protocol-specific.

19657 The SOCK\_SEQPACKET socket type is similar to the SOCK\_STREAM type, and is also  
19658 connection-oriented. The only difference between these types is that record boundaries are  
19659 maintained using the SOCK\_SEQPACKET type. A record can be sent using one or more output  
19660 operations and received using one or more input operations, but a single operation never  
19661 transfers parts of more than one record. Record boundaries are visible to the receiver via the  
19662 MSG\_EOR flag in the received message flags returned by the *recvmsg()* function. It is protocol-  
19663 specific whether a maximum record size is imposed.

19664 The SOCK\_DGRAM socket type supports connectionless data transfer which is not necessarily  
19665 acknowledged or reliable. Datagrams can be sent to the address specified (possibly multicast or  
19666 broadcast) in each output operation, and incoming datagrams can be received from multiple  
19667 sources. The source address of each datagram is available when receiving the datagram. An  
19668 application can also pre-specify a peer address, in which case calls to output functions that do  
19669 not specify a peer address shall send to the pre-specified peer. If a peer has been specified, only  
19670 datagrams from that peer shall be received. A datagram shall be sent in a single output  
19671 operation, and needs to be received in a single input operation. The maximum size of a  
19672 datagram is protocol-specific; with some protocols, the limit is implementation-defined. Output  
19673 datagrams may be buffered within the system; thus, a successful return from an output function  
19674 does not guarantee that a datagram is actually sent or received. However, implementations  
19675 should attempt to detect any errors possible before the return of an output function, reporting  
19676 any error by an unsuccessful return value.

19677 RS The SOCK\_RAW socket type is similar to the SOCK\_DGRAM type. It differs in that it is  
19678 normally used with communication providers that underlie those used for the other socket  
19679 types. For this reason, the creation of a socket with type SOCK\_RAW shall require appropriate  
19680 privileges. The format of datagrams sent and received with this socket type generally include  
19681 specific protocol headers, and the formats are protocol-specific and implementation-defined.

### 19682 2.10.7 Socket I/O Mode

19683 The I/O mode of a socket is described by the O\_NONBLOCK file status flag which pertains to  
19684 the open file description for the socket. This flag is initially off when a socket is created, but may  
19685 be set and cleared by the use of the F\_SETFL command of the *fcntl()* function.

19686 When the O\_NONBLOCK flag is set, certain functions that would normally block until they are  
19687 complete shall return immediately.

19688 The *bind()* function initiates an address assignment and shall return without blocking when  
19689 O\_NONBLOCK is set; if the socket address cannot be assigned immediately, *bind()* shall return  
19690 the [EINPROGRESS] error to indicate that the assignment was initiated successfully, but that it  
19691 has not yet completed.

19692 The *connect()* function initiates a connection and shall return without blocking when  
19693 O\_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection  
19694 was initiated successfully, but that it has not yet completed.

19695 Data transfer operations (the *read()*, *write()*, *send()*, and *recv()* functions) shall complete  
19696 immediately, transfer only as much as is available, and then return without blocking, or return  
19697 an error indicating that no transfer could be made without blocking.

### 19698 2.10.8 Socket Owner

19699 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or  
19700 process group ID using the F\_SETOWN command of the *fcntl()* function.

### 19701 2.10.9 Socket Queue Limits

19702 The transmit and receive queue sizes for a socket are set when the socket is created. The default  
19703 sizes used are both protocol-specific and implementation-defined. The sizes may be changed  
19704 using the *setsockopt()* function.

### 19705 2.10.10 Pending Error

19706 Errors may occur asynchronously, and be reported to the socket in response to input from the  
19707 network protocol. The socket stores the pending error to be reported to a user of the socket at the  
19708 next opportunity. The error is returned in response to a subsequent *send()*, *recv()*, or *getsockopt()*  
19709 operation on the socket, and the pending error is then cleared.

### 19710 2.10.11 Socket Receive Queue

19711 A socket has a receive queue that buffers data when it is received by the system until it is  
19712 removed by a receive call. Depending on the type of the socket and the communication provider,  
19713 the receive queue may also contain ancillary data such as the addressing and other protocol data  
19714 associated with the normal data in the queue, and may contain out-of-band or expedited data.  
19715 The limit on the queue size includes any normal, out-of-band data, datagram source addresses,  
19716 and ancillary data in the queue. The description in this section applies to all sockets, even  
19717 though some elements cannot be present in some instances.

19718 The contents of a receive buffer are logically structured as a series of data segments with  
19719 associated ancillary data and other information. A data segment may contain normal data or  
19720 out-of-band data, but never both. A data segment may complete a record if the protocol  
19721 supports records (always true for types SOCK\_SEQPACKET and SOCK\_DGRAM). A record  
19722 may be stored as more than one segment; the complete record might never be present in the  
19723 receive buffer at one time, as a portion might already have been returned to the application, and  
19724 another portion might not yet have been received from the communications provider. A data  
19725 segment may contain ancillary protocol data, which is logically associated with the segment.  
19726 Ancillary data is received as if it were queued along with the first normal data octet in the  
19727 segment (if any). A segment may contain ancillary data only, with no normal or out-of-band  
19728 data. For the purposes of this section, a datagram is considered to be a data segment that  
19729 terminates a record, and that includes a source address as a special type of ancillary data. Data  
19730 segments are placed into the queue as data is delivered to the socket by the protocol. Normal  
19731 data segments are placed at the end of the queue as they are delivered. If a new segment  
19732 contains the same type of data as the preceding segment and includes no ancillary data, and if  
19733 the preceding segment does not terminate a record, the segments are logically merged into a  
19734 single segment.

19735 The receive queue is logically terminated if an end-of-file indication has been received or a  
19736 connection has been terminated. A segment shall be considered to be terminated if another  
19737 segment follows it in the queue, if the segment completes a record, or if an end-of-file or other  
19738 connection termination has been reported. The last segment in the receive queue shall also be  
19739 considered to be terminated while the socket has a pending error to be reported.

19740 A receive operation shall never return data or ancillary data from more than one segment.

### 19741 2.10.12 Socket Out-of-Band Data State

19742 The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed  
19743 in the socket receive queue, either at the end of the queue or before all normal data in the queue.  
19744 In this case, out-of-band data is returned to an application program by a normal receive call.  
19745 Out-of-band data may also be queued separately rather than being placed in the socket receive  
19746 queue, in which case it shall be returned only in response to a receive call that requests out-of-  
19747 band data. It is protocol-specific whether an out-of-band data mark is placed in the receive  
19748 queue to demarcate data preceding the out-of-band data and following the out-of-band data. An  
19749 out-of-band data mark is logically an empty data segment that cannot be merged with other  
19750 segments in the queue. An out-of-band data mark is never returned in response to an input  
19751 operation. The *socketmark()* function can be used to test whether an out-of-band data mark is the  
19752 first element in the queue. If an out-of-band data mark is the first element in the queue when an  
19753 input function is called without the MSG\_PEEK option, the mark is removed from the queue  
19754 and the following data (if any) is processed as if the mark had not been present.



### 19755 2.10.13 Connection Indication Queue

19756 Sockets that are used to accept incoming connections maintain a queue of outstanding  
19757 connection indications. This queue is a list of connections that are awaiting acceptance by the  
19758 application; see *listen()*.

### 19759 2.10.14 Signals

19760 One category of event at the socket interface is the generation of signals. These signals report  
19761 protocol events or process errors relating to the state of the socket. The generation or delivery of  
19762 a signal does not change the state of the socket, although the generation of the signal may have  
19763 been caused by a state change.

19764 The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no  
19765 longer able to send (one that is no longer connected), except that the signal is suppressed if the  
19766 MSG\_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Regardless of whether  
19767 the generation of the signal is suppressed, the send operation shall fail with the [EPIPE] error.

19768 If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified  
19769 of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the  
19770 status of the socket is specified in [Section 2.10.17](#) (on page 557), [Section 2.10.19](#) (on page 558),  
19771 and [Section 2.10.20](#) (on page 558). Depending on the protocol, the expedited data may or may  
19772 not have arrived at the time of signal generation.

### 19773 2.10.15 Asynchronous Errors

19774 If any of the following conditions occur asynchronously for a socket, the corresponding value  
19775 listed below shall become the pending error for the socket:

19776 [ECONNABORTED]

19777 The connection was aborted locally.

19778 [ECONNREFUSED]

19779 For a connection-mode socket attempting a non-blocking connection, the attempt to connect  
19780 was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram  
19781 was forcefully rejected.

19782 [ECONNRESET]

19783 The peer has aborted the connection.

19784 [EHOSTUNREACH]

19785 The destination host is not reachable.

19786 [EMSGSIZE]

19787 For a connectionless-mode socket, the size of a previously sent datagram prevented  
19788 delivery.

19789 [ENETDOWN]

19790 The local network connection is not operational.

19791 [ENETRESET]

19792 The connection was aborted by the network.

19793 [ENETUNREACH]

19794 The destination network is not reachable.

19795 **2.10.16 Use of Options**

19796 There are a number of socket options which either specialize the behavior of a socket or provide  
 19797 useful information. These options may be set at different protocol levels and are always present  
 19798 at the uppermost “socket” level.

19799 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions  
 19800 allow an application program to customize the behavior and characteristics of a socket to  
 19801 provide the desired effect.

19802 All of the options usable with *setsockopt()* have defaults. For each option where a default value is  
 19803 listed as implementation-defined, the implementation also controls whether a socket created by  
 19804 *accept()* or *accept4()* starts with the option reset to the original default value, or inherited as the  
 19805 value previously customized on the original listening socket. The type and meaning of these  
 19806 values is defined by the protocol level to which they apply. Instead of using the default values,  
 19807 an application program may choose to customize one or more of the options. However, in the  
 19808 bulk of cases, the default values are sufficient for the application.

19809 Some of the options are used to enable or disable certain behavior within the protocol modules  
 19810 (for example, turn on debugging) while others may be used to set protocol-specific information  
 19811 (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is  
 19812 introduced, its effect on the underlying protocol modules is described.

19813 [Table 2-1](#) shows the value for the socket level.

19814 **Table 2-1** Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

19817 [Table 2-2](#) (on page 555) lists those options present at the socket level; that is, when the *level*  
 19818 parameter of the *getsockopt()* or *setsockopt()* function is SOL\_SOCKET, the types of the option  
 19819 value parameters associated with each option, and a brief synopsis of the meaning of the option  
 19820 value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with  
 19821 *setsockopt()* on all types of socket. Options at other protocol levels vary in format and name.

19822

Table 2-2 Socket-Level Options

Option	Parameter Type	Parameter Meaning
SO_ACCEPTCONN	int	Non-zero indicates that socket listening is enabled ( <i>getsockopt()</i> only).
SO_BROADCAST	int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DOMAIN	int	Identify socket domain ( <i>getsockopt()</i> only).
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket ( <i>getsockopt()</i> only).
SO_KEEPAIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsend data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_PROTOCOL	int	Identify socket protocol ( <i>getsockopt()</i> only).
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type ( <i>getsockopt()</i> only).

19855 The SO\_ACCEPTCONN option is used only on *getsockopt()*. When this option is specified,  
 19856 *getsockopt()* shall report whether socket listening is enabled for the socket. A value of zero shall  
 19857 indicate that socket listening is disabled; non-zero that it is enabled. SO\_ACCEPTCONN has no  
 19858 default value.

19859 The SO\_BROADCAST option requests permission to send broadcast datagrams on the socket.  
 19860 Support for SO\_BROADCAST is protocol-specific. The default for SO\_BROADCAST is that the  
 19861 ability to send broadcast datagrams on a socket is disabled.

19862 The SO\_DEBUG option enables debugging in the underlying protocol modules. This can be  
 19863 useful for tracing the behavior of the underlying protocol modules during normal system  
 19864 operation. The semantics of the debug reports are implementation-defined. The default value for  
 19865 SO\_DEBUG is for debugging to be turned off.

19866 The SO\_DOMAIN option is used only on *getsockopt()*. When this option is specified,  
 19867 *getsockopt()* shall return the domain of the socket (for example, AF\_INET6). SO\_DOMAIN has  
 19868 no default value.

19869 The SO\_DONTROUTE option requests that outgoing messages bypass the standard routing

19870 facilities. The destination needs to be on a directly-connected network, and messages are  
19871 directed to the appropriate network interface according to the destination address. It is protocol-  
19872 specific whether this option has any effect and how the outgoing network interface is chosen.  
19873 Support for this option with each protocol is implementation-defined.

19874 The `SO_ERROR` option is used only on `getsockopt()`. When this option is specified, `getsockopt()`  
19875 shall return any pending error on the socket and clear the error status. It shall return a value of 0  
19876 if there is no pending error. `SO_ERROR` may be used to check for asynchronous errors on  
19877 connected connectionless-mode sockets or for other types of asynchronous errors. `SO_ERROR`  
19878 has no default value.

19879 The `SO_KEEPALIVE` option enables the periodic transmission of messages on a connected  
19880 socket. The behavior of this option is protocol-specific. On a connection-mode socket for which a  
19881 connection has been established, if `SO_KEEPALIVE` is enabled and the connected socket fails to  
19882 respond to the keep-alive messages, the connection shall be broken. The default value for  
19883 `SO_KEEPALIVE` is zero, specifying that this capability is turned off.

19884 The `SO_LINGER` option controls the action of the interface when unsent messages are queued  
19885 on a socket and a `close()` is performed. The details of this option are protocol-specific. If  
19886 `SO_LINGER` is enabled, the system shall block the calling thread during `close()` until it can  
19887 transmit the data or until the end of the interval indicated by the `l_linger` member, whichever  
19888 comes first. If `SO_LINGER` is not specified, and `close()` is issued, the system handles the call in a  
19889 way that allows the calling thread to continue as quickly as possible. The default value for  
19890 `SO_LINGER` is zero, or off, for the `l_onoff` element of the option value and zero seconds for the  
19891 linger time specified by the `l_linger` element.

19892 The `SO_OOBINLINE` option is valid only on protocols that support out-of-band data. The  
19893 `SO_OOBINLINE` option requests that out-of-band data be placed in the normal data input  
19894 queue as received; it is then accessible using the `read()` or `recv()` functions without the  
19895 `MSG_OOB` flag set. The default for `SO_OOBINLINE` is off; that is, for out-of-band data not to be  
19896 placed in the normal data input queue.

19897 The `SO_PROTOCOL` option is used only on `getsockopt()`. When this option is specified,  
19898 `getsockopt()` shall return the socket protocol (for example, `IPPROTO_TCP`). `SO_PROTOCOL` has  
19899 no default value.

19900 The `SO_RCVBUF` option requests that the buffer space allocated for receive operations on this  
19901 socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer  
19902 size for high volume connections, or may decrease buffer size to limit the possible backlog of  
19903 incoming data. The default value for the `SO_RCVBUF` option value is implementation-defined,  
19904 and may vary by protocol.

19905 The `SO_RCVLOWAT` option sets the minimum number of bytes to process for socket input  
19906 operations. In general, receive calls block until any (non-zero) amount of data is received, then  
19907 return the smaller of the amount available or the amount requested. The default value for  
19908 `SO_RCVLOWAT` is 1, and does not affect the general case. If `SO_RCVLOWAT` is set to a larger  
19909 value, blocking receive calls normally wait until they have received the smaller of the low water  
19910 mark value or the requested amount. Receive calls may still return less than the low water mark  
19911 if an error occurs, a signal is caught, or the type of data next in the receive queue is different  
19912 from that returned (for example, out-of-band data). As mentioned previously, the default value  
19913 for `SO_RCVLOWAT` is 1 byte. It is implementation-defined whether the `SO_RCVLOWAT` option  
19914 can be set.

19915 The `SO_RCVTIMEO` option is an option to set a timeout value for input operations. It accepts a  
19916 **timeval** structure with the number of seconds and microseconds specifying the limit on how  
19917 long to wait for an input operation to complete. If a receive operation has blocked for this much  
19918 time without receiving additional data, it shall return with a partial count or `errno` shall be set to

19919 [EAGAIN] or [EWOULDBLOCK] if no data were received. The default for this option is the  
 19920 value zero, which indicates that a receive operation will not time out. It is implementation-  
 19921 defined whether the SO\_RCVTIMEO option can be set.

19922 The SO\_REUSEADDR option indicates that the rules used in validating addresses supplied in a  
 19923 *bind()* should allow reuse of local addresses. Operation of this option is protocol-specific. The  
 19924 default value for SO\_REUSEADDR is off; that is, reuse of local addresses is not permitted.

19925 The SO\_SNDBUF option requests that the buffer space allocated for send operations on this  
 19926 socket be set to the value, in bytes, of the option value. The default value for the SO\_SNDBUF  
 19927 option value is implementation-defined, and may vary by protocol.

19928 The SO\_SNDLOWAT option sets the minimum number of bytes to process for socket output  
 19929 operations. Most output operations process all of the data supplied by the call, delivering data to  
 19930 the protocol for transmission and blocking as necessary for flow control. Non-blocking output  
 19931 operations process as much data as permitted subject to flow control without blocking, but  
 19932 process no data if flow control does not allow the smaller of the send low water mark value or  
 19933 the entire request to be processed. A *select()* operation testing the ability to write to a socket shall  
 19934 return true only if the send low water mark could be processed. The default value for  
 19935 SO\_SNDLOWAT is implementation-defined and protocol-specific. It is implementation-defined  
 19936 whether the SO\_SNDLOWAT option can be set.

19937 The SO\_SNDTIMEO option is an option to set a timeout value for the amount of time that an  
 19938 output function shall block because flow control prevents data from being sent. As noted in  
 19939 [Table 2-2](#) (on page 555), the option value is a **timeval** structure with the number of seconds and  
 19940 microseconds specifying the limit on how long to wait for an output operation to complete. If a  
 19941 send operation has blocked for this much time, it shall return with a partial count or *errno* set to  
 19942 [EAGAIN] or [EWOULDBLOCK] if no data were sent. The default for this option is the value  
 19943 zero, which indicates that a send operation will not time out. It is implementation-defined  
 19944 whether the SO\_SNDTIMEO option can be set.

19945 The SO\_TYPE option is used only on *getsockopt()*. When this option is specified, *getsockopt()*  
 19946 shall return the type of the socket (for example, SOCK\_STREAM). This option is useful to  
 19947 servers that inherit sockets on start-up. SO\_TYPE has no default value.

## 19948 2.10.17 Use of Sockets for Local UNIX Connections

19949 Support for UNIX domain sockets is mandatory.

19950 UNIX domain sockets provide process-to-process communication in a single system.

### 19951 2.10.17.1 Headers

19952 The symbolic constant AF\_UNIX defined in the `<sys/socket.h>` header is used to identify the  
 19953 UNIX domain address family. The `<sys/un.h>` header contains other definitions used in  
 19954 connection with UNIX domain sockets. See XBD [Chapter 14](#) (on page 221).

19955 The **sockaddr\_storage** structure defined in `<sys/socket.h>` shall be large enough to  
 19956 accommodate a **sockaddr\_un** structure (see the `<sys/un.h>` header defined in XBD [Chapter 14](#),  
 19957 on page 221) and shall be aligned at an appropriate boundary so that pointers to it can be cast as  
 19958 pointers to **sockaddr\_un** structures and used to access the fields of those structures without  
 19959 alignment problems. When a **sockaddr\_storage** structure is cast as a **sockaddr\_un** structure, the  
 19960 *ss\_family* field maps onto the *sun\_family* field.

## 19961 2.10.18 Use of Sockets over Internet Protocols

19962 When a socket is created in the Internet family with a protocol value of zero, the implementation  
19963 shall use the protocol listed below for the type of socket created.

19964 SOCK\_STREAM IPPROTO\_TCP.

19965 SOCK\_DGRAM IPPROTO\_UDP.

19966 RS SOCK\_RAW IPPROTO\_RAW.

19967 SOCK\_SEQPACKET Unspecified.

19968 RS A raw interface to IP is available by creating an Internet socket of type SOCK\_RAW. The default  
19969 protocol for type SOCK\_RAW shall be identified in the IP header with the value  
19970 IPPROTO\_RAW. Applications should not use the default protocol when creating a socket with  
19971 type SOCK\_RAW, but should identify a specific protocol by value. The ICMP control protocol is  
19972 accessible from a raw socket by specifying a value of IPPROTO\_ICMP for protocol.

## 19973 2.10.19 Use of Sockets over Internet Protocols Based on IPv4

19974 Support for sockets over Internet protocols based on IPv4 is mandatory. IPv4 is described in  
19975 RFC 791.

### 19976 2.10.19.1 Headers

19977 The symbolic constant AF\_INET defined in the `<sys/socket.h>` header is used to identify the  
19978 IPv4 Internet address family. The `<netinet/in.h>` header contains other definitions used in  
19979 connection with IPv4 Internet sockets. See XBD [Chapter 14](#) (on page 221).

19980 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to  
19981 accommodate a `sockaddr_in` structure (see the `<netinet/in.h>` header defined in XBD [Chapter](#)  
19982 [14](#), on page 221) and shall be aligned at an appropriate boundary so that pointers to it can be  
19983 cast as pointers to `sockaddr_in` structures and used to access the fields of those structures  
19984 without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in`  
19985 structure, the `ss_family` field maps onto the `sin_family` field.

## 19986 2.10.20 Use of Sockets over Internet Protocols Based on IPv6

19987 IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. The  
19988 functionality described in this section shall be provided on implementations that support the  
19989 IPV6 option (and the rest of this section is not further shaded for this option).

19990 IPv6 is described in RFC 8200.

19991 To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain  
19992 circumstances, also be used in connection with IPv4; see [Section 2.10.20.2](#) (on page 559).

### 19993 2.10.20.1 Addressing

19994 IPv6 overcomes the addressing limitations of earlier versions by using 128-bit addresses instead  
19995 of 32-bit addresses. The IPv6 address architecture is described in RFC 4291.

19996 There are three kinds of IPv6 address:

- 19997 Unicast  
19998 Identifies a single interface.
- 19999 A unicast address can be global, link-local (designed for use on a single link), or site-local  
20000 (designed for systems not connected to the Internet). Link-local and site-local addresses  
20001 need not be globally unique.
- 20002 Anycast  
20003 Identifies a set of interfaces such that a packet sent to the address can be delivered to any  
20004 member of the set.
- 20005 An anycast address is similar to a unicast address; the nodes to which an anycast address is  
20006 assigned need to be explicitly configured to know that it is an anycast address.
- 20007 Multicast  
20008 Identifies a set of interfaces such that a packet sent to the address should be delivered to  
20009 every member of the set.
- 20010 An application can send multicast datagrams by simply specifying an IPv6 multicast  
20011 address in the *address* argument of *sendto()*. To receive multicast datagrams, an application  
20012 needs to join the multicast group (using *setsockopt()* with `IPV6_JOIN_GROUP`) and bind to  
20013 the socket the UDP port on which datagrams are to be received. Some applications should  
20014 also bind the multicast group address to the socket, to prevent other datagrams destined to  
20015 that port from being delivered to the socket.
- 20016 A multicast address can be global, node-local, link-local, site-local, or organization-local.
- 20017 The following special IPv6 addresses are defined:
- 20018 Unspecified  
20019 An address that is not assigned to any interface and is used to indicate the absence of an  
20020 address.
- 20021 Loopback  
20022 A unicast address that is not assigned to any interface and can be used by a node to send  
20023 packets to itself.
- 20024 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:
- 20025 IPv4-compatible addresses  
20026 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”  
20027 through IPv4.
- 20028 IPv4-mapped addresses  
20029 These are used to represent IPv4 addresses in IPv6 address format; see [Section 2.10.20.2](#).
- 20030 The unspecified address and the loopback address shall not be treated as IPv4-compatible  
20031 addresses.
- 20032 **2.10.20.2 Compatibility with IPv4**
- 20033 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,  
20034 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the  
20035 *getaddrinfo()* function when the specified host has only IPv4 addresses.
- 20036 Applications can use `AF_INET6` sockets to open TCP connections to IPv4 nodes, or send UDP  
20037 packets to IPv4 nodes, by simply encoding the destination’s IPv4 address as an IPv4-mapped  
20038 IPv6 address, and passing that address, within a `sockaddr_in6` structure, in the *connect()*,  
20039 *sendto()*, or *sendmsg()* function. When applications use `AF_INET6` sockets to accept TCP  
20040 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return

20041 the peer's address to the application in the *accept()*, *accept4()*, *recvfrom()*, *recvmsg()*, or  
 20042 *getpeername()* function using a **sockaddr\_in6** structure encoded this way. If a node has an IPv4  
 20043 address, then the implementation shall allow applications to communicate using that address  
 20044 via an AF\_INET6 socket. In such a case, the address shall be represented at the API by the  
 20045 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an AF\_INET6  
 20046 socket bound to **in6addr\_any** to receive inbound connections and packets destined to one of the  
 20047 node's IPv4 addresses.

20048 An application can use AF\_INET6 sockets to bind to a node's IPv4 address by specifying the  
 20049 address as an IPv4-mapped IPv6 address in a **sockaddr\_in6** structure in the *bind()* function. For  
 20050 an AF\_INET6 socket bound to a node's IPv4 address, the system shall return the address in the  
 20051 *getsockname()* function as an IPv4-mapped IPv6 address in a **sockaddr\_in6** structure.

### 20052 2.10.20.3 Interface Identification

20053 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;  
 20054 zero is not used. There may be gaps so that there is no current interface for a particular positive  
 20055 index. Each interface also has a unique implementation-defined name.

### 20056 2.10.20.4 Options

20057 The following options apply at the IPPROTO\_IPV6 level:

#### 20058 IPV6\_JOIN\_GROUP

20059 When set via *setsockopt()*, it joins the application to a multicast group on an interface  
 20060 (identified by its index) and addressed by a given multicast address, enabling packets sent  
 20061 to that address to be read via the socket. If the interface index is specified as zero, the  
 20062 system selects the interface (for example, by looking up the address in a routing table and  
 20063 using the resulting interface).

20064 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

20065 The parameter type of this option is a pointer to an **ipv6\_mreq** structure.

#### 20066 IPV6\_LEAVE\_GROUP

20067 When set via *setsockopt()*, it removes the application from the multicast group on an  
 20068 interface (identified by its index) and addressed by a given multicast address.

20069 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

20070 The parameter type of this option is a pointer to an **ipv6\_mreq** structure.

#### 20071 IPV6\_MULTICAST\_HOPS

20072 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the  
 20073 socket. Its possible values are the same as those of IPV6\_UNICAST\_HOPS. If the  
 20074 IPV6\_MULTICAST\_HOPS option is not set, a value of 1 is assumed. This option can be set  
 20075 via *setsockopt()* and read via *getsockopt()*.

20076 The parameter type of this option is a pointer to an **int**. (Default value: 1)

#### 20077 IPV6\_MULTICAST\_IF

20078 The index of the interface to be used for outgoing multicast packets. It can be set via  
 20079 *setsockopt()* and read via *getsockopt()*. If the interface index is specified as zero, the system  
 20080 selects the interface (for example, by looking up the address in a routing table and using the  
 20081 resulting interface).

20082 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)



20083 IPV6\_MULTICAST\_LOOP  
 20084 This option controls whether outgoing multicast packets should be delivered back to the  
 20085 local application when the sending interface is itself a member of the destination multicast  
 20086 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an  
 20087 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.

20088 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean  
 20089 value. (Default value: 1)

20090 IPV6\_UNICAST\_HOPS  
 20091 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the  
 20092 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to  
 20093 set a value less than -1 or greater than 255 shall result in an [EINVAL] error. This option can  
 20094 be set via *setsockopt()* and read via *getsockopt()*.

20095 The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

20096 IPV6\_V6ONLY  
 20097 This socket option restricts AF\_INET6 sockets to IPv6 communications only. AF\_INET6  
 20098 sockets may be used for both IPv4 and IPv6 communications. Some applications may want  
 20099 to restrict their use of an AF\_INET6 socket to IPv6 communications only. For these  
 20100 applications, the IPV6\_V6ONLY socket option is defined. When this option is turned on, the  
 20101 socket can be used to send and receive IPv6 packets only. This is an IPPROTO\_IPV6-level  
 20102 option.

20103 The parameter type of this option is a pointer to an **int** which is used as a Boolean value.  
 20104 (Default value: 0)

20105 An [EOPNOTSUPP] error shall result if IPV6\_JOIN\_GROUP or IPV6\_LEAVE\_GROUP is used  
 20106 with *getsockopt()*.

#### 20107 2.10.20.5 Headers

20108 The symbolic constant AF\_INET6 is defined in the `<sys/socket.h>` header to identify the IPv6  
 20109 Internet address family. See XBD [Chapter 14](#) (on page 221).

20110 The **sockaddr\_storage** structure defined in `<sys/socket.h>` shall be large enough to  
 20111 accommodate a **sockaddr\_in6** structure (see the `<netinet/in.h>` header defined in XBD [Chapter](#)  
 20112 [14](#), on page 221) and shall be aligned at an appropriate boundary so that pointers to it can be  
 20113 cast as pointers to **sockaddr\_in6** structures and used to access the fields of those structures  
 20114 without alignment problems. When a **sockaddr\_storage** structure is cast as a **sockaddr\_in6**  
 20115 structure, the *ss\_family* field maps onto the *sin6\_family* field.

20116 The `<netinet/in.h>`, `<arpa/inet.h>`, and `<netdb.h>` headers contain other definitions used in  
 20117 connection with IPv6 Internet sockets; see XBD [Chapter 14](#) (on page 221).

## 20118 2.11 Data Types

20119 **2.11.1 Defined Types**

20120 All of the data types used by various functions are defined by the implementation. The  
 20121 following table describes some of these types. Other types referenced in the description of a  
 20122 function, not mentioned here, can be found in the appropriate header for that function.

20123	Defined Type	Description
20124	<b>cc_t</b>	Type used for terminal special characters.
20125	<b>clock_t</b>	Integer or real-floating type used for processor times, as defined in the ISO C standard.
20126		
20127	<b>clockid_t</b>	Used for clock ID type in some timer functions.
20128	<b>dev_t</b>	Integer type used for device numbers.
20129	<b>DIR</b>	Type representing a directory stream.
20130	<b>div_t</b>	Structure type returned by the <i>div()</i> function.
20131	<b>FILE</b>	Structure containing information about a file.
20132	<b>glob_t</b>	Structure type used in pathname pattern matching.
20133	<b>fpos_t</b>	Type containing all information needed to specify uniquely every position within a file.
20134		
20135	<b>gid_t</b>	Integer type used for group IDs.
20136	<b>iconv_t</b>	Type used for conversion descriptors.
20137	<b>id_t</b>	Integer type used as a general identifier; can be used to contain at least the largest of a <b>pid_t</b> , <b>uid_t</b> , or <b>gid_t</b> .
20138		
20139	<b>ino_t</b>	Unsigned integer type used for file serial numbers.
20140	<b>key_t</b>	Arithmetic type used for XSI interprocess communication.
20141	<b>ldiv_t</b>	Structure type returned by the <i>ldiv()</i> function.
20142	<b>mode_t</b>	Integer type used for file attributes.
20143	<b>mqd_t</b>	Used for message queue descriptors.
20144	<b>nfds_t</b>	Integer type used for the number of file descriptors.
20145	<b>nlink_t</b>	Integer type used for link counts.
20146	<b>off_t</b>	Signed integer type used for file sizes.
20147	<b>pid_t</b>	Signed integer type used for process and process group IDs.
20148	<b>pthread_attr_t</b>	Used to identify a thread attribute object.
20149	<b>pthread_cond_t, cnd_t</b>	Used for condition variables.
20150	<b>pthread_condattr_t</b>	Used to identify a condition attribute object.
20151	<b>pthread_key_t, tss_t</b>	Used for thread-specific data keys.
20152	<b>pthread_mutex_t, mtx_t</b>	Used for mutexes.
20153	<b>pthread_mutexattr_t</b>	Used to identify a mutex attribute object.
20154	<b>pthread_once_t, once_flag</b>	Used for dynamic package initialization.
20155	<b>pthread_rwlock_t</b>	Used for read-write locks.
20156	<b>pthread_rwlockattr_t</b>	Used for read-write lock attributes.
20157	<b>pthread_t, thrd_t</b>	Used to identify a thread.
20158	<b>ptrdiff_t</b>	Signed integer type of the result of subtracting two pointers.
20159	<b>reclen_t</b>	Unsigned integer type used for directory entry lengths.
20160	<b>regex_t</b>	Structure type used in regular expression matching.
20161	<b>regmatch_t</b>	Structure type used in regular expression matching.
20162	<b>rlim_t</b>	Unsigned integer type used for limit values, to which objects of type <b>int</b> and <b>off_t</b> can be cast without loss of value.
20163		
20164	<b>sem_t</b>	Type used in performing semaphore operations.
20165	<b>sig_atomic_t</b>	Possibly volatile-qualified integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
20166		
20167		
20168	<b>sigset_t</b>	Integer or structure type of an object used to represent sets

Defined Type	Description
	of signals.
<b>size_t</b>	Unsigned integer type used for size of objects.
<b>speed_t</b>	Type used for terminal baud rates.
<b>ssize_t</b>	Signed integer type used for a count of bytes or an error indication.
<b>suseconds_t</b>	Signed integer type used for time in microseconds.
<b>tcflag_t</b>	Type used for terminal modes.
<b>time_t</b>	Integer type used for time in seconds, as defined in the ISO C standard.
<b>timer_t</b>	Used for timer ID returned by the <i>timer_create()</i> function.
<b>uid_t</b>	Integer type used for user IDs.
<b>va_list</b>	Type used for traversing variable argument lists.
<b>wchar_t</b>	Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales.
<b>wctype_t</b>	Scalar type which represents a character class descriptor.
<b>wint_t</b>	Integer type capable of storing any valid value of <b>wchar_t</b> or WEOF.
<b>wordexp_t</b>	Structure type used in word expansion.

### 2.11.2 The char Type

The type **char** is defined as a single byte; see XBD Chapter 3 (on page 31) (Byte and Character).

## 2.12 Status Information

Status information is data associated with a process detailing a change in the state of the process. It shall consist of:

- The state the process transitioned into (*stopped*, *continued*, or *terminated*)
- The information necessary to populate the **siginfo\_t** structure provided by *waitid()*
- If the new state is *terminated*:
  - The low-order 8 bits of the status argument that the process passed to *\_Exit()*, *\_exit()*, or *exit()*, or the low-order 8 bits of the value the process returned from *main()*
  - Note that these 8 bits are part of the complete value that is used to set the *si\_status* member of the **siginfo\_t** structure provided by *waitid()*
  - Whether the process terminated due to the receipt of a signal that was not caught and, if so, the number of the signal that caused the termination of the process
- If the new state is *stopped*:
  - The number of the signal that caused the process to stop

A process might not have any status information (such as immediately after a process has started).

Status information for a process shall be generated (made available to the parent process) when the process stops, continues, or terminates except in the following case:

- 20209
- 20210
- 20211
- 20212
- If the parent process sets the action for the SIGCHLD signal to SIG\_IGN, or if the parent sets the SA\_NOCLDWAIT flag for the SIGCHLD signal action, process termination shall not generate new status information but shall cause any existing status information for the process to be discarded.

20213

20214

If new status information is generated, and the process already had status information, the existing status information shall be discarded and replaced with the new status information.

20215

20216

20217

20218

20219

Only the process' parent process can obtain the process' status information. The parent obtains a child's status information by calling *wait()*, *waitid()*, or *waitpid()*. Except when *waitid()* is called with the WNOWAIT flag set in the *options* argument, the status information obtained by a wait function shall be consumed (discarded) by that wait function; no two calls to *wait()*, *waitid()* (without WNOWAIT), or *waitpid()* shall obtain the same status information.

20220

20221

20222


When status information becomes available to the parent process and more than one thread in the parent process is waiting for the status information (blocked in a call to *wait()*, *waitid()*, or *waitpid()* with arguments that would match the status information):

- 20223
- 20224
- If none of the matching threads is in a call to *waitid()* with the WNOWAIT flag set in the *options* argument, the thread that obtains the status information is unspecified.
- 20225
- Otherwise (at least one of the matching threads is in a call to *waitid()* with the WNOWAIT flag set), the matching thread or threads that obtain the status information is unspecified except that at least one of the matching threads shall obtain the status information and at most one of the matching threads that are not calling *waitid()* with the WNOWAIT flag set shall obtain it.
- 20226
- 20227
- 20228
- 20229

20230

Chapter 3

20231



# *System Interfaces*

20232

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

20233

20234 **NAME**

20235 CMPLX, CMPLXF, CMPLXL — make a complex value

20236 **SYNOPSIS**

```
20237 #include <complex.h>
20238 double complex    CMPLX(double x, double y);
20239 float complex    CMPLXF(float x, float y);
20240 long double complex CMPLXL(long double x, long double y);
```

20241 **DESCRIPTION**

20242 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 20243 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20244 volume of POSIX.1-2024 defers to the ISO C standard.

20245 The CMPLX macros shall expand to an expression of the specified complex type, with the real  
 20246 part having the (converted) value of  $x$  and the imaginary part having the (converted) value of  $y$ .  
 20247 The resulting expression shall be suitable for use as an initializer for an object with static or  
 20248 thread storage duration, provided both arguments are likewise suitable.

20249 **RETURN VALUE**20250 The CMPLX macros return the complex value  $x + iy$  (where  $i$  is the imaginary unit).

20251 These macros shall behave as if the implementation supported imaginary types and the  
 20252 definitions were:

```
20253 #define CMPLX(x, y) ((double complex)((double)(x) + \
20254 _Imaginary_I * (double)(y)))
20255 #define CMPLXF(x, y) ((float complex)((float)(x) + \
20256 _Imaginary_I * (float)(y)))
20257 #define CMPLXL(x, y) ((long double complex)((long double)(x) + \
20258 _Imaginary_I * (long double)(y)))
```

20259 **ERRORS**

20260 No errors are defined.

20261 **EXAMPLES**

20262 None.

20263 **APPLICATION USAGE**

20264 None.

20265 **RATIONALE**

20266 None.

20267 **FUTURE DIRECTIONS**

20268 None.

20269 **SEE ALSO**20270 XBD [<complex.h>](#)20271 **CHANGE HISTORY**

20272 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

20273 **NAME**

20274 FD\_CLR — macros for synchronous I/O multiplexing

20275 **SYNOPSIS**

```
20276     #include <sys/select.h>
20277     void FD_CLR(int fd, fd_set *fdset);
20278     int FD_ISSET(int fd, fd_set *fdset);
20279     void FD_SET(int fd, fd_set *fdset);
20280     void FD_ZERO(fd_set *fdset);
```

20281 **DESCRIPTION**20282 Refer to *pselect()*.

**20283 NAME**

20284 `_Exit, _exit` — terminate a process

**20285 SYNOPSIS**

```
20286 #include <stdlib.h>
20287 _Noreturn void _Exit(int status);
20288 #include <unistd.h>
20289 _Noreturn void _exit(int status);
```

**20290 DESCRIPTION**

20291 CX For `_Exit()`: The functionality described on this reference page is aligned with the ISO C  
20292 standard. Any conflict between the requirements described here and the ISO C standard is  
20293 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

20294 CX The value of *status* may be 0, `EXIT_SUCCESS`, `EXIT_FAILURE`, or any other value, though only  
20295 the least significant 8 bits (that is, *status* & 0377) shall be available from `wait()` and `waitpid()`; the  
20296 full value shall be available from `waitid()` and in the `siginfo_t` passed to a signal handler for  
20297 `SIGCHLD`.

20298 CX The `_Exit()` and `_exit()` functions shall be functionally equivalent.

20299 CX The `_Exit()` and `_exit()` functions shall not call functions registered with `atexit()` nor  
20300 CX `at_quick_exit()`, nor any registered signal handlers. Open streams shall not be flushed. Whether  
20301 open streams are closed (without flushing) is implementation-defined. Finally, the calling  
20302 process shall be terminated with the consequences described below.

**20303 Consequences of Process Termination**

20304 CX Process termination caused by any reason shall have the following consequences:

20305 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.  
20306 However, functionality relating to the XSI option is shaded.

20307 • All of the file descriptors, directory streams, conversion descriptors, and message catalog  
20308 descriptors open in the calling process shall be closed.

20309 XSI • If the parent process of the calling process has set its `SA_NOCLDWAIT` flag or has set the  
20310 action for the `SIGCHLD` signal to `SIG_IGN`:

20311 — The process' status information (see [Section 2.12](#), on page 563), if any, shall be  
20312 discarded.

20313 — The lifetime of the calling process shall end immediately. If `SA_NOCLDWAIT` is set,  
20314 it is implementation-defined whether a `SIGCHLD` signal is sent to the parent process.

20315 — If a thread in the parent process of the calling process is blocked in `wait()`, `waitpid()`,  
20316 or `waitid()`, and the parent process has no remaining child processes in the set of  
20317 waited-for children, the `wait()`, `waitid()`, or `waitpid()` function shall fail and set `errno`  
20318 to `[ECHILD]`.

20319 **Otherwise:**

20320 — Status information (see [Section 2.12](#), on page 563) shall be generated.

20321 — The calling process shall be transformed into a zombie process. Its status information  
20322 shall be made available to the parent process until the process' lifetime ends.



- 20323 — The process' lifetime shall end once its parent obtains the process' status information  
20324 via a currently-blocked or future call to *wait()*, *waitid()* (without *WNOWAIT*), or  
20325 *waitpid()*.
- 20326 — If one or more threads in the parent process of the calling process is blocked in a call  
20327 to *wait()*, *waitid()*, or *waitpid()* awaiting termination of the process, one (or, if any are  
20328 calling *waitid()* with *WNOWAIT*, possibly more) of these threads shall obtain the  
20329 process' status information as specified in [Section 2.12](#) (on page 563) and become  
20330 unblocked.
- 20331 — A *SIGCHLD* shall be sent to the parent process.
- 20332 • Termination of a process does not directly terminate its children. The sending of a  
20333 *SIGHUP* signal as described below indirectly terminates children in some circumstances.
  - 20334 • The parent process ID of all of the existing child processes and zombie processes of the  
20335 calling process shall be set to the process ID of an implementation-defined system process.  
20336 That is, these processes shall be inherited by a special system process.
  - 20337 XSI • Each attached shared-memory segment is detached and the value of *shm\_nattch* (see  
20338 *shmget()*) in the data structure associated with its shared memory ID shall be decremented  
20339 by 1.
  - 20340 XSI • For each semaphore for which the calling process has set a *semadj* value (see *semop()*), that  
20341 value shall be added to the *semval* of the specified semaphore.
  - 20342 • If the process is a controlling process, the *SIGHUP* signal shall be sent to each process in  
20343 the foreground process group of the controlling terminal belonging to the calling process.
  - 20344 • If the process is a controlling process, the controlling terminal associated with the session  
20345 shall be disassociated from the session, allowing it to be acquired by a new controlling  
20346 process.
  - 20347 • If the exit of the process causes a process group to become orphaned, and if any member of  
20348 the newly-orphaned process group is stopped, then a *SIGHUP* signal followed by a  
20349 *SIGCONT* signal shall be sent to each process in the newly-orphaned process group.
  - 20350 • All open named semaphores in the calling process shall be closed as if by appropriate calls  
20351 to *sem\_close()*. All open unnamed semaphores in the calling process shall be destroyed as  
20352 if by appropriate calls to *sem\_destroy()*.
  - 20353 ML • Any memory locks established by the process via calls to *mlockall()* or *mlock()* shall be  
20354 removed. If locked pages in the address space of the calling process are also mapped into  
20355 the address spaces of other processes and are locked by those processes, the locks  
20356 established by the other processes shall be unaffected by the call by this process to *\_Exit()*  
20357 or *\_exit()*.
  - 20358 • Memory mappings that were created in the process shall be unmapped before the process  
20359 is destroyed.
  - 20360 TYM • Any blocks of typed memory that were mapped in the calling process shall be unmapped,  
20361 as if *munmap()* was implicitly called to unmap them.
  - 20362 MSG • All open message queue descriptors in the calling process shall be closed as if by  
20363 appropriate calls to *mq\_close()*.
  - 20364 • Any outstanding cancelable asynchronous I/O operations may be canceled. Those  
20365 asynchronous I/O operations that are not canceled shall complete as if the *\_Exit()* or  
20366 *\_exit()* operation had not yet occurred, but any associated signal notifications shall be

20367 suppressed. The `_Exit()` or `_exit()` operation may block awaiting such I/O completion.  
20368 Whether any I/O is canceled, and which I/O may be canceled upon `_Exit()` or `_exit()`, is  
20369 implementation-defined.

- 20370 • Threads terminated by a call to `_Exit()` or `_exit()` shall not invoke their cancellation  
20371 cleanup handlers or per-thread data destructors.

## 20372 **RETURN VALUE**

20373 These functions do not return.

## 20374 **ERRORS**

20375 No errors are defined.

## 20376 **EXAMPLES**

20377 None.

## 20378 **APPLICATION USAGE**

20379 Normally applications should use `exit()` rather than `_Exit()` or `_exit()`.

20380 Exit statuses of 126 and greater are ambiguous in certain circumstances because they have  
20381 special meanings in the shell (see [Section 2.8.2](#) (on page 2499) and the EXIT STATUS section of  
20382 *sh*).

## 20383 **RATIONALE**

### 20384 **Process Termination**

20385 Early proposals drew a distinction between normal and abnormal process termination.  
20386 Abnormal termination was caused only by certain signals and resulted in implementation-  
20387 defined “actions”, as discussed below. Subsequent proposals distinguished three types of  
20388 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and  
20389 *abnormal termination with actions*. Again the distinction between the two types of abnormal  
20390 termination was that they were caused by different signals and that implementation-defined  
20391 actions would result in the latter case. Given that these actions were completely implementation-  
20392 defined, the early proposals were only saying when the actions could occur and how their  
20393 occurrence could be detected, but not what they were. This was of little or no use to conforming  
20394 applications, and thus the distinction is not made in this volume of POSIX.1-2024.

20395 The implementation-defined actions usually include, in most historical implementations, the  
20396 creation of a file named **core** in the current working directory of the process. This file contains an  
20397 image of the memory of the process, together with descriptive information about the process,  
20398 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

20399 There is a potential security problem in creating a **core** file if the process was set-user-ID and the  
20400 current user is not the owner of the program, if the process was set-group-ID and none of the  
20401 user’s groups match the group of the program, or if the user does not have permission to write  
20402 in the current directory. In this situation, an implementation either should not create a **core** file  
20403 or should make it unreadable by the user.

20404 Despite the silence of this volume of POSIX.1-2024 on this feature, applications are advised not  
20405 to create files named **core** because of potential conflicts in many implementations. Some  
20406 implementations use a name other than **core** for the file; for example, by appending the process  
20407 ID to the filename.

20408 **Terminating a Process**

20409 It is important that the consequences of process termination as described occur regardless of  
20410 whether the process called `_exit()` (perhaps indirectly through `exit()`) or instead was terminated  
20411 due to a signal or for some other reason. Note that in the specific case of `exit()` this means that  
20412 the `status` argument to `exit()` is treated in the same way as the `status` argument to `_exit()`.

20413 A language other than C may have other termination primitives than the C-language `exit()`  
20414 function, and programs written in such a language should use its native termination primitives,  
20415 but those should have as part of their function the behavior of `_exit()` as described.  
20416 Implementations in languages other than C are outside the scope of this version of this volume  
20417 of POSIX.1-2024, however.

20418 As required by the ISO C standard, using **return** from `main()` has the same behavior (other than  
20419 with respect to language scope issues) as calling `exit()` with the returned value. Reaching the end  
20420 of the `main()` function has the same behavior as calling `exit(0)`.

20421 A value of zero (or `EXIT_SUCCESS`, which is required to be zero) for the argument `status`  
20422 conventionally indicates successful termination. This corresponds to the specification for `exit()`  
20423 in the ISO C standard. The convention is followed by utilities such as `make` and various shells,  
20424 which interpret a zero status from a child process as success. For this reason, applications should  
20425 not call `exit(0)` or `_exit(0)` when they terminate unsuccessfully; for example, in signal-catching  
20426 functions.

20427 Historically, the implementation-defined process that inherits children whose parents have  
20428 terminated without waiting on them is called `init` and has a process ID of 1.

20429 The sending of a `SIGHUP` to the foreground process group when a controlling process  
20430 terminates corresponds to somewhat different historical implementations. In System V, the  
20431 kernel sends a `SIGHUP` on termination of (essentially) a controlling process. In 4.2 BSD, the  
20432 kernel does not send `SIGHUP` in a case like this, but the termination of a controlling process is  
20433 usually noticed by a system daemon, which arranges to send a `SIGHUP` to the foreground  
20434 process group with the `vhangup()` function. However, in 4.2 BSD, due to the behavior of the  
20435 shells that support job control, the controlling process is usually a shell with no other processes  
20436 in its process group. Thus, a change to make `_exit()` behave this way in such systems should not  
20437 cause problems with existing applications.

20438 The termination of a process may cause a process group to become orphaned in either of two  
20439 ways. The connection of a process group to its parent(s) outside of the group depends on both  
20440 the parents and their children. Thus, a process group may be orphaned by the termination of the  
20441 last connecting parent process outside of the group or by the termination of the last direct  
20442 descendant of the parent process(es). In either case, if the termination of a process causes a  
20443 process group to become orphaned, processes within the group are disconnected from their job  
20444 control shell, which no longer has any information on the existence of the process group.  
20445 Stopped processes within the group would languish forever. In order to avoid this problem,  
20446 newly orphaned process groups that contain stopped processes are sent a `SIGHUP` signal and a  
20447 `SIGCONT` signal to indicate that they have been disconnected from their session. The `SIGHUP`  
20448 signal causes the process group members to terminate unless they are catching or ignoring  
20449 `SIGHUP`. Under most circumstances, all of the members of the process group are stopped if any  
20450 of them are stopped.

20451 The action of sending a `SIGHUP` and a `SIGCONT` signal to members of a newly orphaned  
20452 process group is similar to the action of 4.2 BSD, which sends `SIGHUP` and `SIGCONT` to each  
20453 stopped child of an exiting process. If such children exit in response to the `SIGHUP`, any  
20454 additional descendants receive similar treatment at that time. In this volume of POSIX.1-2024,  
20455 the signals are sent to the entire process group at the same time. Also, in this volume of

20456 POSIX.1-2024, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a  
20457 process group that is not orphaned; therefore, the action taken at `_exit()` must consider processes  
20458 other than child processes.

20459 It is possible for a process group to be orphaned by a call to `setpgid()` or `setsid()`, as well as by  
20460 process termination. This volume of POSIX.1-2024 does not require sending SIGHUP and  
20461 SIGCONT in those cases, because, unlike process termination, those cases are not caused  
20462 accidentally by applications that are unaware of job control. An implementation can choose to  
20463 send SIGHUP and SIGCONT in those cases as an extension; such an extension must be  
20464 documented as required in **<signal.h>**.

20465 The ISO/IEC 9899:1999 standard added the `_Exit()` function that results in immediate program  
20466 termination without triggering signals or `atexit()`-registered functions. In POSIX.1-2024, this is  
20467 equivalent to the `_exit()` function.

## 20468 **FUTURE DIRECTIONS**

20469 None.

## 20470 **SEE ALSO**

20471 *at\_quick\_exit()*, *atexit()*, *exit()*, *mlock()*, *mlockall()*, *mq\_close()*, *munmap()*, *quick\_exit()*, *sem\_close()*,  
20472 *semop()*, *setpgid()*, *setsid()*, *shmget()*, *wait()*, *waitid()*

20473 XBD **<stdlib.h>**, **<unistd.h>**

## 20474 **CHANGE HISTORY**

20475 First released in Issue 1. Derived from Issue 1 of the SVID.

### 20476 **Issue 5**

20477 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
20478 Threads Extension.

20479 Interactions with the SA\_NOCLDWAIT flag and SIGCHLD signal are further clarified.

20480 The values of *status* from `exit()` are better described.

### 20481 **Issue 6**

20482 Extensions beyond the ISO C standard are marked.

20483 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics  
20484 for typed memory.

20485 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 20486 • The `_Exit()` function is included.
- 20487 • The DESCRIPTION is updated.

20488 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

20489 References to the `wait3()` function are removed.

20490 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the  
20491 DESCRIPTION.

### 20492 **Issue 7**

20493 Austin Group Interpretation 1003.1-2001 #031 is applied, separating these functions from the  
20494 `exit()` function.

20495 Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of  
20496 streams and closing of temporary files.

20497 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and

- 20498 Semaphores options is moved to the Base.
- 20499 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0033 [594] and XSH/TC2-2008/0034  
20500 [594,690] are applied.
- 20501 **Issue 8**
- 20502 Austin Group Defect 368 is applied, adding a requirement for unnamed semaphores to be  
20503 destroyed.
- 20504 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
20505 standard.
- 20506 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 20507 Austin Group Defect 1629 is applied, changing the APPLICATION USAGE section.

20508 **NAME**

20509 `_Fork` — create a new process

20510 **SYNOPSIS**

20511 `#include <unistd.h>`

20512 `pid_t _Fork(void);`

20513 **DESCRIPTION**

20514 Refer to *fork()*.

20515 **NAME**

20516 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

20517 **SYNOPSIS**

```
20518 XSI #include <stdlib.h>
20519 long a64l(const char *s);
20520 char *l64a(long value);
```

20521 **DESCRIPTION**

20522 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by  
 20523 which 32-bit integers can be represented by up to six characters; each character represents a digit  
 20524 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall  
 20525 be used for these operations.

20526 The characters used to represent digits are '.' (dot) for 0, '/' for 1, '0' through '9' for [2,11],  
 20527 'A' through 'Z' for [12,37], and 'a' through 'z' for [38,63].

20528 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the  
 20529 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains  
 20530 more than six characters, *a64l()* shall use the first six. If the first six characters of the string  
 20531 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The  
 20532 *a64l()* function shall scan the character string from left to right with the least significant digit on  
 20533 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than  
 20534 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null  
 20535 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

20536 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding  
 20537 radix-64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

20538 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may  
 20539 overwrite the buffer.

20540 The *l64a()* function need not be thread-safe.

20541 **RETURN VALUE**

20542 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the  
 20543 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

20544 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall  
 20545 return a pointer to an empty string.

20546 **ERRORS**

20547 No errors are defined.

20548 **EXAMPLES**

20549 None.

20550 **APPLICATION USAGE**

20551 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.

20552 **RATIONALE**

20553 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

20554 **FUTURE DIRECTIONS**

20555 None.

20556 **SEE ALSO**20557 [strtol\(\)](#)20558 XBD [<stdlib.h>](#)20559 XCU [uuencode](#)20560 **CHANGE HISTORY**

20561 First released in Issue 4, Version 2.

20562 **Issue 5**

20563 Moved from X/OPEN UNIX extension to BASE.

20564 Normative text previously in the APPLICATION USAGE section is moved to the  
20565 DESCRIPTION.20566 A note indicating that the *l64a()* function need not be reentrant is added to the DESCRIPTION.20567 **Issue 7**

20568 Austin Group Interpretation 1003.1-2001 #156 is applied.



20569 **NAME**

20570 abort — generate an abnormal process abort

20571 **SYNOPSIS**

20572 #include &lt;stdlib.h&gt;

20573 \_Noreturn void abort(void);

20574 **DESCRIPTION**

20575 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 20576 conflict between the requirements described here and the ISO C standard is unintentional. This  
 20577 volume of POSIX.1-2024 defers to the ISO C standard.

20578 The *abort()* function shall cause abnormal process termination to occur, unless a SIGABRT signal  
 20579 that it generates is caught and the signal handler does not return.

20580 CX The abnormal termination processing shall include the default actions defined for SIGABRT and  
 20581 may include an attempt to effect *fclose()* on all open streams.

20582 CX The SIGABRT signal shall be sent to the calling thread as if by means of *raise()* with the  
 20583 CX argument SIGABRT. If this signal does not terminate the process (for example, if the signal is  
 20584 caught and the handler returns), *abort()* may change the disposition of SIGABRT to SIG\_DFL  
 20585 and send the signal (in the same way) again. If a second signal is sent and it does not terminate  
 20586 the process, the behavior is unspecified, except that the *abort()* call shall not return.

20587 CX The status made available to *wait()*, *waitid()*, or *waitpid()* by *abort()* shall be that of a process  
 20588 terminated by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the  
 20589 SIGABRT signal.

20590 **RETURN VALUE**20591 The *abort()* function shall not return.20592 **ERRORS**

20593 No errors are defined.

20594 **EXAMPLES**

20595 None.

20596 **APPLICATION USAGE**

20597 Catching the signal is intended to provide the application developer with a portable means to  
 20598 abort processing, free from possible interference from any implementation-supplied functions.

20599 **RATIONALE**

20600 Historically, *abort()* has been implemented by calling other signal manipulation functions such  
 20601 as *raise()*, *sigaction()*, and *pthread\_sigmask()*. This means that its operation can be affected by  
 20602 concurrent actions in other threads. For example, if *abort()* attempts to terminate the process by  
 20603 calling *sigaction()* to change the disposition for SIGABRT to SIG\_DFL and then calling *raise()*,  
 20604 another thread could change the disposition in between those two calls, resulting in the process  
 20605 not being terminated. If this happens, the only requirement is that *abort()* does not return. An  
 20606 implementation could call those functions in a loop (which could in theory then execute  
 20607 indefinitely), or could terminate the process by calling *\_exit()* (which would ensure termination  
 20608 but result in the wrong wait status). To avoid these issues, implementations are encouraged to  
 20609 implement *abort()* in a manner such that its operation cannot be affected by concurrent actions  
 20610 in other threads. For example, it could first halt the execution of all other threads, or it could  
 20611 terminate the process using a “terminate as if by a signal” system call instead of by raising (a  
 20612 second) SIGABRT.

20613 The ISO/IEC 9899:1999 standard required (and the current standard still requires) the *abort()*  
 20614 function to be async-signal-safe. Since POSIX.1-2024 defers to the ISO C standard, this required a

20615 change to the DESCRIPTION from “shall include the effect of *fclose()*” to “may include an  
20616 attempt to effect *fclose()*.”

20617 The revised wording permits some backwards-compatibility and avoids a potential deadlock  
20618 situation.

20619 The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded  
20620 paragraph from the DESCRIPTION:

20621 “On XSI-conformant systems, in addition the abnormal termination processing shall include the  
20622 effect of *fclose()* on message catalog descriptors.”

20623 There were several reasons to remove this paragraph:

- 20624 • No special processing of open message catalogs needs to be performed prior to abnormal  
20625 process termination.
- 20626 • The main reason to specifically mention that *abort()* includes the effect of *fclose()* on open  
20627 streams is to flush output queued on the stream. Message catalogs in this context are read-  
20628 only and, therefore, do not need to be flushed.
- 20629 • The effect of *fclose()* on a message catalog descriptor is unspecified. Message catalog  
20630 descriptors are allowed, but not required to be implemented using a file descriptor, but  
20631 there is no mention in POSIX.1-2024 of a message catalog descriptor using a standard I/O  
20632 stream FILE object as would be expected by *fclose()*.

#### 20633 FUTURE DIRECTIONS

20634 A future version of this standard may require *abort()* to be implemented in a manner such that  
20635 its operation cannot be affected by concurrent actions in other threads.

#### 20636 SEE ALSO

20637 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitid()*

20638 XBD <stdlib.h>

#### 20639 CHANGE HISTORY

20640 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 20641 Issue 6

20642 Extensions beyond the ISO C standard are marked.

20643 Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

20644 The Open Group Base Resolution bwg2002-003 is applied.

20645 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the  
20646 DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

20647 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/9 is applied, changing “implementation-  
20648 defined functions” to “implementation-supplied functions” in the APPLICATION USAGE  
20649 section.

#### 20650 Issue 8

20651 Austin Group Defect 906 is applied, clarifying how the behavior of *abort()* may be affected by  
20652 concurrent actions in other threads.

20653 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
20654 standard.

20655 **NAME**

20656 abs — return an integer absolute value

20657 **SYNOPSIS**

20658 #include &lt;stdlib.h&gt;

20659 int abs(int i);

20660 **DESCRIPTION**

20661 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
20662 conflict between the requirements described here and the ISO C standard is unintentional. This  
20663 volume of POSIX.1-2024 defers to the ISO C standard.

20664 The *abs()* function shall compute the absolute value of its integer operand, *i*. If the result cannot  
20665 be represented, the behavior is undefined.

20666 **RETURN VALUE**20667 The *abs()* function shall return the absolute value of its integer operand.20668 **ERRORS**

20669 No errors are defined.

20670 **EXAMPLES**

20671 None.

20672 **APPLICATION USAGE**

20673 Since POSIX.1 requires a two's complement representation of **int**, the absolute value of the  
20674 negative integer with the largest magnitude {INT\_MIN} is not representable, thus *abs*(INT\_MIN)  
20675 is undefined.

20676 **RATIONALE**

20677 None.

20678 **FUTURE DIRECTIONS**

20679 None.

20680 **SEE ALSO**20681 *fabs()*, *labs()*

20682 XBD &lt;stdlib.h&gt;

20683 **CHANGE HISTORY**

20684 First released in Issue 1. Derived from Issue 1 of the SVID.

20685 **Issue 6**

20686 Extensions beyond the ISO C standard are marked.

20687 **Issue 8**

20688 Austin Group Defect 1108 is applied, changing the APPLICATION USAGE section.

20689 **NAME**

20690 accept, accept4 — accept a new connection on a socket

20691 **SYNOPSIS**

```
20692 #include <sys/socket.h>
20693 int accept(int socket, struct sockaddr *restrict address,
20694           socklen_t *restrict address_len);
20695 int accept4(int socket, struct sockaddr *restrict address,
20696            socklen_t *restrict address_len, int flag);
```

20697 **DESCRIPTION**

20698 The *accept()* function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket. The file descriptor shall be allocated as described in [Section 2.6](#) (on page 525).

20702 The *accept()* function takes the following arguments:

20703	<i>socket</i>	Specifies a socket that was created with <i>socket()</i> , has been bound to an address with <i>bind()</i> , and has issued a successful call to <i>listen()</i> .
20705	<i>address</i>	Either a null pointer, or a pointer to a <b>sockaddr</b> structure where the address of the connecting socket shall be returned.
20707	<i>address_len</i>	Either a null pointer, if <i>address</i> is a null pointer, or a pointer to a <b>socklen_t</b> object which on input specifies the length of the supplied <b>sockaddr</b> structure, and on output specifies the length of the address of the connecting socket.

20710 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in the object pointed to by *address\_len*.

20713 If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

20715 If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

20717 If the listen queue is empty of connection requests and `O_NONBLOCK` is not set on the file descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is empty of connection requests and `O_NONBLOCK` is set on the file descriptor for the socket, *accept()* shall fail and set *errno* to `[EAGAIN]` or `[EWOULDBLOCK]`.

20721 The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

20723 If `O_NONBLOCK` is set on the file description for *socket*, it is implementation-defined whether `O_NONBLOCK` will be set on the file description created by *accept()*. `FD_CLOEXEC` and `FD_CLOFORK` for the new file descriptor shall be clear, regardless of how they are currently set for *socket*.

20727 It is implementation-defined which socket options, if any, on the accepted socket will have a default value determined by a value previously customized by *setsockopt()* on *socket*, rather than the default value used for other new sockets.

20730 The *accept4()* function shall be equivalent to the *accept()* function, except that the state of `O_NONBLOCK` on the new file description, and `FD_CLOEXEC` and `FD_CLOFORK` on the returned file descriptor shall be determined solely by the *flag* argument, which can be constructed from a bitwise-inclusive OR of flags from the following list:

- 20734 SOCK\_CLOEXEC Atomically set the FD\_CLOEXEC flag on the new file descriptor.
- 20735 SOCK\_CLOFORK Atomically set the FD\_CLOFORK flag on the new file descriptor.
- 20736 SOCK\_NONBLOCK Set the O\_NONBLOCK file status flag on the new file description.
- 20737 Implementations may define additional flags.

#### 20738 RETURN VALUE

- 20739 Upon successful completion, *accept()* and *accept4()* shall return the non-negative file descriptor of the accepted socket. Otherwise, *-1* shall be returned, *errno* shall be set to indicate the error, and any object pointed to by *address\_len* shall remain unchanged.

#### 20742 ERRORS

- 20743 The *accept()* and *accept4()* functions shall fail if:
- 20744 [EAGAIN] or [EWOULDBLOCK]  
20745 O\_NONBLOCK is set for the socket file descriptor and no connections are  
20746 present to be accepted.
- 20747 [EBADF] The *socket* argument is not a valid file descriptor.
- 20748 [ECONNABORTED]  
20749 A connection has been aborted.
- 20750 [EINTR] The *accept()* function was interrupted by a signal that was caught before a  
20751 valid connection arrived.
- 20752 [EINVAL] The *socket* is not accepting connections.
- 20753 [EMFILE] All file descriptors available to the process are currently open.
- 20754 [ENFILE] The maximum number of file descriptors in the system are already open.
- 20755 [ENOBUFS] No buffer space is available.
- 20756 [ENOMEM] There was insufficient memory available to complete the operation.
- 20757 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 20758 [EOPNOTSUPP] The socket type of the specified socket does not support accepting  
20759 connections.

20760 The *accept()* and *accept4()* functions may fail if:

- 20761 [EPROTO] A protocol error has occurred.
- 20762 The *accept4()* function may fail if:
- 20763 [EINVAL] The value of the *flag* argument is invalid.

#### 20764 EXAMPLES

- 20765 None.

#### 20766 APPLICATION USAGE

- 20767 When a connection is available, *select()* indicates that the file descriptor for the socket is ready  
20768 for reading.

- 20769 Many socket options are described as having implementation-defined default values, which  
20770 may differ according to the protocol in use by the socket. Existing practice differs on whether  
20771 socket options such as SO\_SNDBUF that were customized on the original listening socket will  
20772 impact the corresponding option on the newly returned socket. Implementations are permitted  
20773 to allow inheritance of customized settings where it makes sense, although the most portable

20774 approach for applications is to limit *setsockopt()* customizations to only the accepted socket.

20775 For AF\_UNIX sockets, it is recommended that *address* points to a buffer of length greater than  
 20776 `sizeof(struct sockaddr_un)` which has been initialized with null bytes. That way, even if  
 20777 the implementation supports the use of all bytes of *sun\_path* without a terminating null byte, the  
 20778 larger buffer guarantees that the *sun\_path* member can then be passed to other interfaces that  
 20779 expect a null-terminated string. If no truncation occurred based on the input value of *address\_len*,  
 20780 it is unspecified whether the returned *address\_len* will be `sizeof(struct sockaddr_un)`, or  
 20781 merely a value at least as large as `offsetof(struct sockaddr_un, sun_path)` plus the  
 20782 number of non-null bytes stored in *sun\_path*.

#### 20783 RATIONALE

20784 The SOCK\_CLOEXEC and SOCK\_CLOFORK flags of *accept4()* are necessary to avoid a data  
 20785 race in multi-threaded applications. Without SOCK\_CLOFORK, a file descriptor is leaked into a  
 20786 child process created by one thread in the window between another thread creating a file  
 20787 descriptor with *accept()* and then using *fcntl()* to set the FD\_CLOFORK flag. Without  
 20788 SOCK\_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked  
 20789 into an executed program if FD\_CLOEXEC is not set atomically.

20790 Two designs often used for network servers are multi-threaded servers with a pre-created pool  
 20791 of worker threads, where the thread that accepts the connection request hands over the new file  
 20792 descriptor to a worker thread for servicing, and pre-fork servers with a pre-created pool of  
 20793 worker processes, where the process that accepts the connection request passes the new file  
 20794 descriptor (for example via *sendmsg()*) to a worker process. In both of these designs, *accept4()*  
 20795 should be used with the SOCK\_CLOFORK flag set. Simpler designs are also sometimes used  
 20796 that do not pre-create a pool. For a multi-threaded server that creates a thread to handle each  
 20797 request, SOCK\_CLOFORK should still be used. For a forking server that creates a child to  
 20798 service each request, clearly SOCK\_CLOFORK cannot be used if the child is to inherit the file  
 20799 descriptor to be serviced, and therefore this type of server needs to use an alternative method of  
 20800 indicating the end of communications, for example using *shutdown()*, to ensure the client sees  
 20801 end-of-file, rather than just closing the socket. Such child processes should set FD\_CLOFORK on  
 20802 the inherited file descriptor before they attempt to start any additional child processes to avoid  
 20803 leakage into those children.

20804 The SOCK\_NONBLOCK flag is for convenience in avoiding additional *fcntl()* calls, as well as  
 20805 providing specific control over the O\_NONBLOCK flag, since traditional implementations of  
 20806 *accept()* differ on whether O\_NONBLOCK is inherited from the *socket* argument.

#### 20807 FUTURE DIRECTIONS

20808 None.

#### 20809 SEE ALSO

20810 [Section 2.6](#) (on page 525), *bind()*, *connect()*, *listen()*, *socket()*

20811 XBD [<sys/socket.h>](#)

#### 20812 CHANGE HISTORY

20813 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

20814 The **restrict** keyword is added to the *accept()* prototype for alignment with the  
 20815 ISO/IEC 9899:1999 standard.

#### 20816 Issue 7

20817 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

20818 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS]  
 20819 and [ENOMEM] errors to become “shall fail” errors.

- 20820            Functionality relating to XSI STREAMS is marked obsolescent.
- 20821            POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0018 [464] is applied.
- 20822            POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0035 [835] and XSH/TC2-2008/0036  
20823            [836] are applied.
- 20824    **Issue 8**
- 20825            Austin Group Defects 411 and 1318 are applied, adding *accept4()*, requiring FD\_CLOEXEC and  
20826            FD\_CLOFORK to be clear for the file descriptor returned by *accept()*, and clarifying the  
20827            requirements for O\_NONBLOCK on the file description created by *accept()*.
- 20828            Austin Group Defect 561 is applied, adding a paragraph about *sun\_path* to APPLICATION  
20829            USAGE.
- 20830            Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 20831            Austin Group Defect 1337 is applied, clarifying socket option default values.
- 20832            Austin Group Defect 1565 is applied, changing the description of *address\_len*.

20833 **NAME**

20834 access, faccessat — determine accessibility of a file descriptor

20835 **SYNOPSIS**

20836 #include &lt;unistd.h&gt;

20837 int access(const char \*path, int amode);

20838 OH #include &lt;fcntl.h&gt;

20839 int faccessat(int fd, const char \*path, int amode, int flag);

20840 **DESCRIPTION**

20841 The `access()` function shall check the file named by the pathname pointed to by the `path`  
 20842 argument for accessibility according to the bit pattern contained in `amode`. The checks for  
 20843 accessibility (including directory permissions checked during pathname resolution) shall be  
 20844 performed using the real user ID in place of the effective user ID and the real group ID in place  
 20845 of the effective group ID.

20846 The value of `amode` is either the bitwise-inclusive OR of the access permissions to be checked  
 20847 (R\_OK, W\_OK, X\_OK) or the existence test (F\_OK).

20848 If any access permissions are checked, each shall be checked individually, as described in XBD  
 20849 [Section 4.7](#) (on page 97), except that where that description refers to execute permission for a  
 20850 process with appropriate privileges, an implementation may indicate success for X\_OK even if  
 20851 execute permission is not granted to any user.

20852 The `faccessat()` function, when called with a `flag` value of zero, shall be equivalent to the `access()`  
 20853 function, except in the case where `path` specifies a relative path. In this case the file whose  
 20854 accessibility is to be determined shall be located relative to the directory associated with the file  
 20855 descriptor `fd` instead of the current working directory. If the access mode of the open file  
 20856 description associated with the file descriptor is not O\_SEARCH, the function shall check  
 20857 whether directory searches are permitted using the current permissions of the directory  
 20858 underlying the file descriptor. If the access mode is O\_SEARCH, the function shall not perform  
 20859 the check.

20860 If `faccessat()` is passed the special value AT\_FDCWD in the `fd` parameter, the current working  
 20861 directory shall be used and, if `flag` is zero, the behavior shall be identical to a call to `access()`.

20862 Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 20863 in <fcntl.h>:

20864 AT\_EACCESS The checks for accessibility (including directory permissions checked during  
 20865 pathname resolution) shall be performed using the effective user ID and  
 20866 group ID instead of the real user ID and group ID as required in a call to  
 20867 `access()`.

20868 **RETURN VALUE**

20869 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 20870 return -1 and set `errno` to indicate the error.

20871 **ERRORS**

20872 These functions shall fail if:

20873 [EACCES] Permission bits of the file mode do not permit the requested access, or search  
 20874 permission is denied on a component of the path prefix.

20875 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`  
 20876 argument.



20877	[ENAMETOOLONG]	
20878		The length of a component of a pathname is longer than {NAME_MAX}.
20879	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
20880	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>path</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
20881		
20882		
20883		
20884		
20885	[EROFS]	Write access is requested for a file on a read-only file system.
20886		The <i>faccessat()</i> function shall fail if:
20887	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not <code>O_SEARCH</code> and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
20888		
20889		
20890	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
20891		
20892	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
20893		
20894		These functions may fail if:
20895	[EINVAL]	The value of the <i>amode</i> argument is invalid.
20896	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
20897		
20898	[ENAMETOOLONG]	
20899		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
20900		
20901		
20902	[ETXTBSY]	Write access is requested for a pure procedure (shared text) file that is being executed.
20903		
20904		The <i>faccessat()</i> function may fail if:
20905	[EINVAL]	The value of the <i>flag</i> argument is not valid.

## 20906 EXAMPLES

### 20907 Testing for the Existence of a File

20908 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```

20909 #include <unistd.h>
20910 ...
20911 int result;
20912 const char *pathname = "/tmp/myfile";
20913 result = access (pathname, F_OK);

```

## 20914 APPLICATION USAGE

20915 Use of these functions is discouraged since by the time the returned information is acted upon, it  
 20916 is out-of-date. (That is, acting upon the information always leads to a time-of-check-to-time-of-  
 20917 use race condition.) An application should instead attempt the action itself and handle the  
 20918 [EACCES] error that occurs if the file is not accessible (with a change of effective user and group  
 20919 IDs beforehand, and perhaps a change back afterwards, in the case where *access()* or *faccessat()*  
 20920 without AT\_EACCESS would have been used.)

20921 Historically, one of the uses of *access()* was in set-user-ID root programs to check whether the  
 20922 user running the program had access to a file. This relied on “super-user” privileges which were  
 20923 granted based on the effective user ID being zero, so that when *access()* used the real user ID to  
 20924 check accessibility those privileges were not taken into account. On newer systems where  
 20925 privileges can be assigned which have no association with user or group IDs, if a program with  
 20926 such privileges calls *access()*, the change of IDs has no effect on the privileges and therefore they  
 20927 are taken into account in the accessibility checks. Thus, *access()* (and *faccessat()* with flag zero)  
 20928 cannot be used for this historical purpose in such programs. Likewise, if a system provides any  
 20929 additional or alternate file access control mechanisms that are not user ID-based, they will still  
 20930 be taken into account.

20931 If a relative pathname is used, no account is taken of whether the current directory (or the  
 20932 directory associated with the file descriptor *fd*) is accessible via any absolute pathname.  
 20933 Applications using *access()*, or *faccessat()* without AT\_EACCESS, may consequently act as if the  
 20934 file would be accessible to a user with the real user ID and group ID of the process when such a  
 20935 user would not in practice be able to access the file because access would be denied at some  
 20936 point above the current directory (or the directory associated with the file descriptor *fd*) in the  
 20937 file hierarchy.

20938 If *access()* or *faccessat()* is used with W\_OK to check for write access to a directory which has the  
 20939 S\_ISVTX bit set, a return value indicating the directory is writable can be misleading since some  
 20940 operations on files in the directory would not be permitted based on the ownership of those files  
 20941 (see XBD Section 4.5, on page 96).

20942 Additional values of *amode* other than the set defined in the description may be valid; for  
 20943 example, if a system has extended access controls.

20944 The use of the AT\_EACCESS value for *flag* enables functionality not available in *access()*.

## 20945 RATIONALE

20946 In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()*  
 20947 function because:

- 20948 1. Historical implementations of *access()* do not test file access correctly when the process’  
 20949 real user ID is superuser. In particular, they always return zero when testing execute  
 20950 permissions without regard to whether the file is executable.
- 20951 2. The superuser has complete access to all files on a system. As a consequence, programs  
 20952 started by the superuser and switched to the effective user ID with lesser privileges  
 20953 cannot use *access()* to test their file access permissions.

20954 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of  
 20955 POSIX.1-2024 now allows *access()* to behave in the desired way because several implementations  
 20956 have corrected the problem. It was also argued that problem (2) is more easily solved by using  
 20957 *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the error, rather  
 20958 than creating a new function that would not be as reliable. Therefore, *eaccess()* is not included in  
 20959 this volume of POSIX.1-2024.

20960 The sentence concerning appropriate privileges and execute permission bits reflects the two

20961 possibilities implemented by historical implementations when checking superuser access for  
20962 X\_OK.

20963 New implementations are discouraged from returning X\_OK unless at least one execution  
20964 permission bit is set.

20965 The purpose of the *faccessat()* function is to enable the checking of the accessibility of files in  
20966 directories other than the current working directory without exposure to race conditions. Any  
20967 part of the path of a file could be changed in parallel to a call to *access()*, resulting in unspecified  
20968 behavior. By opening a file descriptor for the target directory and using the *faccessat()* function it  
20969 can be guaranteed that the file tested for accessibility is located relative to the desired directory.

#### 20970 **FUTURE DIRECTIONS**

20971 These functions may be formally deprecated (for example, by shading them OB) in a future  
20972 version of this standard.

#### 20973 **SEE ALSO**

20974 *chmod()*, *fstatat()*

20975 XBD Section 4.7 (on page 97), *<fcntl.h>*, *<unistd.h>*

#### 20976 **CHANGE HISTORY**

20977 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 20978 **Issue 6**

20979 The following new requirements on POSIX implementations derive from alignment with the  
20980 Single UNIX Specification:

- 20981 • The [ELOOP] mandatory error condition is added.
- 20982 • A second [ENAMETOOLONG] is added as an optional error condition.
- 20983 • The [ETXTBSY] optional error condition is added.

20984 The following changes were made to align with the IEEE P1003.1a draft standard:

- 20985 • The [ELOOP] optional error condition is added.

#### 20986 **Issue 7**

20987 Austin Group Interpretations 1003.1-2001 #046 and #143 are applied.

20988 The *faccessat()* function is added from The Open Group Technical Standard, 2006, Extended API  
20989 Set Part 2.

20990 Changes are made to allow a directory to be opened for searching.

20991 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
20992 pathname exists but is not a directory or a symbolic link to a directory.

20993 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0019 [461], XSH/TC1-2008/0020 [324],  
20994 XSH/TC1-2008/0021 [278], XSH/TC1-2008/0022 [278], and XSH/TC1-2008/0023 [291] are  
20995 applied.

20996 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0037 [873], XSH/TC2-2008/0038 [591],  
20997 XSH/TC2-2008/0039 [838], XSH/TC2-2008/0040 [817], XSH/TC2-2008/0041 [487],  
20998 XSH/TC2-2008/0042 [838], XSH/TC2-2008/0043 [817], and XSH/TC2-2008/0044 [838] are  
20999 applied.

21000 **NAME**

21001           acos, acosf, acosl — arc cosine functions

21002 **SYNOPSIS**

```
21003       #include <math.h>
21004       double acos(double x);
21005       float acosf(float x);
21006       long double acosl(long double x);
```

21007 **DESCRIPTION**

21008 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 21009 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21010 volume of POSIX.1-2024 defers to the ISO C standard.

21011       These functions shall compute the principal value of the arc cosine of their argument  $x$ . The  
 21012 value of  $x$  should be in the range  $[-1,1]$ .

21013       An application wishing to check for error situations should set *errno* to zero and call  
 21014 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 21015 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 21016 zero, an error has occurred.

21017 **RETURN VALUE**

21018       Upon successful completion, these functions shall return the arc cosine of  $x$ , in the range  $[0,\pi]$   
 21019 radians.

21020 MX       For finite values of  $x$  not in the range  $[-1,1]$ , a domain error shall occur, and either a NaN (if  
 21021 supported), or an implementation-defined value shall be returned.

21022 MX       If  $x$  is NaN, a NaN shall be returned.

21023       If  $x$  is +1, +0 shall be returned.

21024       If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and a NaN shall be returned.

21025 **ERRORS**

21026       These functions shall fail if:

21027 MX       Domain Error    The  $x$  argument is finite and is not in the range  $[-1,1]$ , or is  $\pm\text{Inf}$ .

21028       If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 21029 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 21030 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 21031 shall be raised.

21032 **EXAMPLES**

21033       None.

21034 **APPLICATION USAGE**

21035       On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 21036 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21037 **RATIONALE**

21038       None.

21039 **FUTURE DIRECTIONS**

21040       None.

21041 **SEE ALSO**21042 *cos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*21043 XBD Section 4.23 (on page 109), **<math.h>**21044 **CHANGE HISTORY**

21045 First released in Issue 1. Derived from Issue 1 of the SVID.

21046 **Issue 5**21047 The DESCRIPTION is updated to indicate how an application should check for an error. This  
21048 text was previously published in the APPLICATION USAGE section.21049 **Issue 6**21050 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.21051 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
21052 revised to align with the ISO/IEC 9899:1999 standard.21053 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
21054 marked.21055 **Issue 7**

21056 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0024 [320] is applied.

21057 **NAME**

21058 acosh, acoshf, acoshl — inverse hyperbolic cosine functions

21059 **SYNOPSIS**

```
21060 #include <math.h>
21061 double acosh(double x);
21062 float acoshf(float x);
21063 long double acoshl(long double x);
```

21064 **DESCRIPTION**

21065 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21066 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21067 volume of POSIX.1-2024 defers to the ISO C standard.

21068 These functions shall compute the inverse hyperbolic cosine of their argument  $x$ .

21069 An application wishing to check for error situations should set *errno* to zero and call  
 21070 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 21071 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 21072 zero, an error has occurred.

21073 **RETURN VALUE**

21074 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their  
 21075 argument.

21076 MX For finite values of  $x < 1$ , a domain error shall occur, and either a NaN (if supported), or an  
 21077 implementation-defined value shall be returned.

21078 MX If  $x$  is NaN, a NaN shall be returned.

21079 If  $x$  is +1, +0 shall be returned.

21080 If  $x$  is +Inf, +Inf shall be returned.

21081 If  $x$  is -Inf, a domain error shall occur, and a NaN shall be returned.

21082 **ERRORS**

21083 These functions shall fail if:

21084 MX Domain Error The  $x$  argument is finite and less than +1.0, or is -Inf.

21085 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 21086 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 21087 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 21088 shall be raised.

21089 **EXAMPLES**

21090 None.

21091 **APPLICATION USAGE**

21092 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 21093 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21094 **RATIONALE**

21095 None.

21096 **FUTURE DIRECTIONS**

21097 None.

21098 **SEE ALSO**21099 *cosh()*, *feclearexcept()*, *fetestexcept()*21100 XBD Section 4.23 (on page 109), **<math.h>**21101 **CHANGE HISTORY**

21102 First released in Issue 4, Version 2.

21103 **Issue 5**

21104 Moved from X/OPEN UNIX extension to BASE.

21105 **Issue 6**21106 The *acosh()* function is no longer marked as an extension.21107 The *acoshf()* and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999  
21108 standard.21109 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
21110 revised to align with the ISO/IEC 9899:1999 standard.21111 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
21112 marked.21113 **Issue 7**

21114 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0025 [320] is applied.

21115 **NAME**

21116           acosl — arc cosine functions

21117 **SYNOPSIS**

21118           #include &lt;math.h&gt;

21119           long double acosl(long double *x*);21120 **DESCRIPTION**21121           Refer to *acos()*.



21122 **NAME**

21123 aio\_cancel — cancel an asynchronous I/O request

21124 **SYNOPSIS**

21125 #include &lt;aio.h&gt;

21126 int aio\_cancel(int *fildev*, struct aiocb \**aiocbp*);21127 **DESCRIPTION**

21128 The *aio\_cancel()* function shall attempt to cancel one or more asynchronous I/O requests  
 21129 currently outstanding against file descriptor *fildev*. The *aiocbp* argument points to the  
 21130 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then  
 21131 all outstanding cancelable asynchronous I/O requests against *fildev* shall be canceled.

21132 Normal asynchronous notification shall occur for asynchronous I/O operations that are  
 21133 successfully canceled. If there are requests that cannot be canceled, then the normal  
 21134 asynchronous completion process shall take place for those requests when they are completed.

21135 For requested operations that are successfully canceled, the associated error status shall be set to  
 21136 [ECANCELED] and the return status shall be -1. For requested operations that are not  
 21137 successfully canceled, the *aiocbp* shall not be modified by *aio\_cancel()*.

21138 If *aiocbp* is not NULL, then if *fildev* does not have the same value as the file descriptor with which  
 21139 the asynchronous operation was initiated, unspecified results occur.

21140 Which operations are cancelable is implementation-defined.

21141 **RETURN VALUE**

21142 The *aio\_cancel()* function shall return the value AIO\_CANCELED if the requested operation(s)  
 21143 were canceled. The value AIO\_NOTCANCELED shall be returned if at least one of the  
 21144 requested operation(s) cannot be canceled because it is in progress. In this case, the state of the  
 21145 other operations, if any, referenced in the call to *aio\_cancel()* is not indicated by the return value  
 21146 of *aio\_cancel()*. The application may determine the state of affairs for these operations by using  
 21147 *aio\_error()*. The value AIO\_ALLDONE is returned if all of the operations have already  
 21148 completed. Otherwise, the function shall return -1 and set *errno* to indicate the error.

21149 **ERRORS**

21150 The *aio\_cancel()* function shall fail if:

21151 [EBADF] The *fildev* argument is not a valid file descriptor.

21152 **EXAMPLES**

21153 None.

21154 **APPLICATION USAGE**

21155 None.

21156 **RATIONALE**

21157 None.

21158 **FUTURE DIRECTIONS**

21159 None.

21160 **SEE ALSO**

21161 [aio\\_read\(\)](#), [aio\\_write\(\)](#)

21162 XBD [<aio.h>](#)

21163 **CHANGE HISTORY**

21164 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21165 **Issue 6**

21166 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
21167 implementation does not support the Asynchronous Input and Output option.

21168 The APPLICATION USAGE section is added.

21169 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/10 is applied, removing the words “to the  
21170 calling process” in the RETURN VALUE section. The term was unnecessary and precluded  
21171 threads.

21172 **Issue 7**

21173 The *aio\_cancel()* function is moved from the Asynchronous Input and Output option to the Base.

21174 **NAME**

21175 aio\_error — retrieve errors status for an asynchronous I/O operation

21176 **SYNOPSIS**

21177 #include &lt;aio.h&gt;

21178 int aio\_error(const struct aiocb \*aiocbp);

21179 **DESCRIPTION**

21180 The *aio\_error()* function shall return the error status associated with the **aiocb** structure  
 21181 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the  
 21182 SIO *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()*  
 21183 operation. If the operation has not yet completed, then the error status shall be equal to  
 21184 [EINPROGRESS].

21185 If the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been  
 21186 scheduled, the results are undefined.

21187 **RETURN VALUE**

21188 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the  
 21189 asynchronous operation has completed unsuccessfully, then the error status, as described for  
 21190 SIO *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has  
 21191 not yet completed, then [EINPROGRESS] shall be returned.

21192 If the *aio\_error()* function fails, it shall return  $-1$  and set *errno* to indicate the error.

21193 **ERRORS**21194 The *aio\_error()* function may fail if:

21195 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose  
 21196 return status has not yet been retrieved.

21197 **EXAMPLES**

21198 None.

21199 **APPLICATION USAGE**

21200 None.

21201 **RATIONALE**

21202 None.

21203 **FUTURE DIRECTIONS**

21204 None.

21205 **SEE ALSO**

21206 *aio\_cancel()*, *aio\_fsync()*, *aio\_read()*, *aio\_return()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*,  
 21207 *lseek()*, *read()*

21208 XBD &lt;aio.h&gt;

21209 **CHANGE HISTORY**

21210 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21211 **Issue 6**

21212 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 21213 implementation does not support the Asynchronous Input and Output option.

21214 The APPLICATION USAGE section is added.

21215 **Issue 7**

21216 Austin Group Interpretation 1003.1-2001 #045 is applied.

21217 SD5-XSH-ERN-148 is applied.

21218 The *aio\_error()* function is moved from the Asynchronous Input and Output option to the Base.

21219 **NAME**

21220 aio\_fsync — asynchronous file synchronization

21221 **SYNOPSIS**

```
21222 FSC|SIO #include <aio.h>
21223 int aio_fsync(int op, struct aiocb *aiocbp);
```

21224 **DESCRIPTION**

21225 The *aio\_fsync()* function shall asynchronously perform a file synchronization operation, as  
 21226 specified by the *op* argument, for I/O operations associated with the file indicated by the file  
 21227 descriptor *aio\_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument and  
 21228 queued at the time of the call to *aio\_fsync()*. The function call shall return when the  
 21229 synchronization request has been initiated or queued to the file or device (even when the data  
 21230 cannot be synchronized immediately).

21231 SIO If *op* is *O\_DSYNC*, all currently queued I/O operations shall be completed as if by a call to  
 21232 *fdatsync()*; that is, as defined for synchronized I/O data integrity completion.

21233 FSC If *op* is *O\_SYNC*, all currently queued I/O operations shall be completed as if by a call to *fsync()*;  
 21234 FSC SIO that is, as defined for synchronized I/O file integrity completion. If the *aio\_fsync()* function  
 21235 fails, or if the operation queued by *aio\_fsync()* fails, then outstanding I/O operations are not  
 21236 guaranteed to have been completed.

21237 If *aio\_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to  
 21238 *aio\_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of  
 21239 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized  
 21240 fashion.

21241 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used  
 21242 as an argument to *aio\_error()* and *aio\_return()* in order to determine the error status and return  
 21243 status, respectively, of the asynchronous operation while it is proceeding. When the request is  
 21244 queued, the error status for the operation is [EINPROGRESS]. When all data has been  
 21245 successfully transferred, the error status shall be reset to reflect the success or failure of the  
 21246 operation. If the operation does not complete successfully, the error status for the operation shall  
 21247 be set to indicate the error. The *aio\_sigevent* member determines the asynchronous notification to  
 21248 occur as specified in Section 2.4.1 (on page 513) when all operations have achieved synchronized  
 21249 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the  
 21250 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O  
 21251 completion, then the behavior is undefined.

21252 If the *aio\_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to  
 21253 have been successfully transferred.

21254 **RETURN VALUE**

21255 The *aio\_fsync()* function shall return the value 0 if the I/O operation is successfully queued;  
 21256 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

21257 **ERRORS**

21258 The *aio\_fsync()* function shall fail if:

21259 [EAGAIN] The requested asynchronous operation was not queued due to temporary  
 21260 resource limitations.

21261 [EBADF] The *aio\_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument  
 21262 is not a valid file descriptor.

- 21263 SIO [EINVAL] This implementation does not support synchronized I/O for this file.
- 21264 FSC [EINVAL] The *aio\_fildes* member of the *aiocb* structure refers to a file on which an *fsync()* operation is not possible.
- 21265
- 21266 [EINVAL] A value of *op* other than O\_DSYNC or O\_SYNC was specified, or O\_DSYNC was specified and the implementation does not provide runtime support for the Synchronized Input and Output option, or O\_SYNC was specified and the implementation does not provide runtime support for the File Synchronization option.
- 21267
- 21268
- 21269
- 21270
- 21271 In the event that any of the queued I/O operations fail, *aio\_fsync()* shall return the error condition defined for *read()* and *write()*. The error is returned in the error status for the asynchronous operation, which can be retrieved using *aio\_error()*.
- 21272
- 21273

#### 21274 EXAMPLES

21275 None.

#### 21276 APPLICATION USAGE

21277 Note that even if the file descriptor is not open for writing, if there are any pending write requests on the underlying file, then that I/O will be completed prior to the return of a call to *aio\_error()* or *aio\_return()* indicating that the operation has completed.

21278

21279

21280 See also the APPLICATION USAGE for *fdatasync()* and *fsync()*.

#### 21281 RATIONALE

21282 None.

#### 21283 FUTURE DIRECTIONS

21284 None.

#### 21285 SEE ALSO

21286 *aio\_error()*, *aio\_return()*, *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*

21287 XBD <[aio.h](#)>

#### 21288 CHANGE HISTORY

21289 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 21290 Issue 6

21291 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

21292

21293 The APPLICATION USAGE section is added.

21294 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/11 is applied, removing the words “to the calling process” in the RETURN VALUE section. The term was unnecessary and precluded threads.

21295

21296

#### 21297 Issue 7

21298 The *aio\_fsync()* function is moved from the Asynchronous Input and Output option to the Base.

21299 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0026 [98] and XSH/TC1-2008/0027 [98] are applied.

21300

21301 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0045 [671] is applied.

21302 **Issue 8**  
21303

Austin Group Defect 672 is applied, changing the APPLICATION USAGE section.

21304 **NAME**

21305 aio\_read — asynchronous read from a file

21306 **SYNOPSIS**

21307 #include &lt;aio.h&gt;

21308 int aio\_read(struct aiocb \*aiocbp);

21309 **DESCRIPTION**

21310 The *aio\_read()* function shall read *aiocbp->aio\_nbytes* from the file associated with  
 21311 *aiocbp->aio\_fildes* into the buffer pointed to by *aiocbp->aio\_buf*. The function call shall return  
 21312 when the read request has been initiated or queued to the file or device (even when the data  
 21313 cannot be delivered immediately).

21314 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted  
 21315 at a priority equal to a base scheduling priority minus *aiocbp->aio\_reqprio*. If Thread Execution  
 21316 Scheduling is not supported, then the base scheduling priority is that of the calling process;  
 21317 PIO TPS otherwise, the base scheduling priority is that of the calling thread.

21318 The *aiocbp* value may be used as an argument to *aio\_error()* and *aio\_return()* in order to  
 21319 determine the error status and return status, respectively, of the asynchronous operation while it  
 21320 is proceeding. If an error condition is encountered during queuing, the function call shall return  
 21321 without having initiated or queued the request. The requested operation takes place at the  
 21322 absolute position in the file as given by *aio\_offset*, as if *lseek()* were called immediately prior to  
 21323 the operation with an *offset* equal to *aio\_offset* and a *whence* equal to *SEEK\_SET*. After a  
 21324 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file  
 21325 is unspecified.

21326 The *aio\_sigevent* member specifies the notification which occurs when the request is completed.

21327 The *aiocbp->aio\_lio\_opcode* field shall be ignored by *aio\_read()*.

21328 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio\_buf* or  
 21329 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O  
 21330 completion, then the behavior is undefined.

21331 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

21332 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio\_fildes*, the behavior of this  
 21333 function shall be according to the definitions of synchronized I/O data integrity completion and  
 21334 synchronized I/O file integrity completion.

21335 For any system action that changes the process memory space while an asynchronous I/O is  
 21336 outstanding to the address range being changed, the result of that action is undefined.

21337 For regular files, no data transfer shall occur past the offset maximum established in the open  
 21338 file description associated with *aiocbp->aio\_fildes*.

21339 **RETURN VALUE**

21340 The *aio\_read()* function shall return the value zero if the I/O operation is successfully queued;  
 21341 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

21342 **ERRORS**

21343 The *aio\_read()* function shall fail if:

21344 [EAGAIN] The requested asynchronous I/O operation was not queued due to system  
 21345 resource limitations.

21346 Each of the following conditions may be detected synchronously at the time of the call to  
 21347 *aio\_read()*, or asynchronously. If any of the conditions below are detected synchronously, the



21348 *aio\_read()* function shall return `-1` and set *errno* to the corresponding value. If any of the  
 21349 conditions below are detected asynchronously, the return status of the asynchronous operation  
 21350 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.

21351 [EBADF] The *aiocbp->aio\_fildes* argument is not a valid file descriptor open for reading.

21352 [EINVAL] The file offset value implied by *aiocbp->aio\_offset* would be invalid,  
 21353 PIO *aiocbp->aio\_reqprio* is not a valid value, or *aiocbp->aio\_nbytes* is an invalid  
 21354 value.

21355 In the case that the *aio\_read()* successfully queues the I/O operation but the operation is  
 21356 subsequently canceled or encounters an error, the return status of the asynchronous operation is  
 21357 one of the values normally returned by the *read()* function call. In addition, the error status of  
 21358 the asynchronous operation is set to one of the error statuses normally set by the *read()* function  
 21359 call, or one of the following values:

21360 [EBADF] The *aiocbp->aio\_fildes* argument is not a valid file descriptor open for reading.

21361 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
 21362 *aio\_cancel()* request.

21363 [EINVAL] The file offset value implied by *aiocbp->aio\_offset* would be invalid.

21364 The following condition may be detected synchronously or asynchronously:

21365 [EOVERFLOW] The file is a regular file, *aiocbp->aio\_nbytes* is greater than 0, and the starting  
 21366 offset in *aiocbp->aio\_offset* is before the end-of-file and is at or beyond the  
 21367 offset maximum in the open file description associated with *aiocbp->aio\_fildes*.

#### 21368 EXAMPLES

21369 None.

#### 21370 APPLICATION USAGE

21371 None.

#### 21372 RATIONALE

21373 None.

#### 21374 FUTURE DIRECTIONS

21375 None.

#### 21376 SEE ALSO

21377 *aio\_cancel()*, *aio\_error()*, *lio\_listio()*, *aio\_return()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,  
 21378 *read()*

21379 XBD <[aio.h](#)>

#### 21380 CHANGE HISTORY

21381 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21382 Large File Summit extensions are added.

#### 21383 Issue 6

21384 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 21385 implementation does not support the Asynchronous Input and Output option.

21386 The APPLICATION USAGE section is added.

21387 The following new requirements on POSIX implementations derive from alignment with the  
21388 Single UNIX Specification:

21389 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
21390 file description. This change is to support large files.

21391 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
21392 large files.

21393 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/12 is applied, rewording the  
21394 DESCRIPTION when prioritized I/O is supported to account for threads, and removing the  
21395 words “to the calling process” in the RETURN VALUE section.

21396 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/13 is applied, updating the [EINVAL]  
21397 error, so that detection of an [EINVAL] error for an invalid value of *aiocbp*→*aio\_reqprio* is only  
21398 required if the Prioritized Input and Output option is supported.

21399 **Issue 7**

21400 Austin Group Interpretation 1003.1-2001 #082 is applied.

21401 The *aio\_read()* function is moved from the Asynchronous Input and Output option to the Base.

21402 **NAME**

21403 aio\_return — retrieve return status of an asynchronous I/O operation

21404 **SYNOPSIS**

```
21405 #include <aio.h>
21406 ssize_t aio_return(struct aiocb *aiocbp);
```

21407 **DESCRIPTION**

21408 The *aio\_return()* function shall return the return status associated with the **aiocb** structure  
 21409 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the  
 21410 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the  
 21411 error status for the operation is equal to [EINPROGRESS], then the return status for the  
 21412 operation is undefined. The *aio\_return()* function may be called exactly once to retrieve the  
 21413 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in  
 21414 a call to *aio\_return()* or *aio\_error()*, an error may be returned. When the **aiocb** structure referred  
 21415 to by *aiocbp* is used to submit another asynchronous operation, then *aio\_return()* may be  
 21416 successfully used to retrieve the return status of that operation.

21417 **RETURN VALUE**

21418 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,  
 21419 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,  
 21420 the results of *aio\_return()* are undefined.

21421 If the *aio\_return()* function fails, it shall return `-1` and set *errno* to indicate the error.

21422 **ERRORS**

21423 The *aio\_return()* function may fail if:

21424 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose  
 21425 return status has not yet been retrieved.

21426 **EXAMPLES**

21427 None.

21428 **APPLICATION USAGE**

21429 None.

21430 **RATIONALE**

21431 None.

21432 **FUTURE DIRECTIONS**

21433 None.

21434 **SEE ALSO**

21435 *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*, *aio\_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio\_listio()*,  
 21436 *lseek()*, *read()*

21437 XBD [<aio.h>](#)

21438 **CHANGE HISTORY**

21439 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21440 **Issue 6**

21441 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 21442 implementation does not support the Asynchronous Input and Output option.

21443 The APPLICATION USAGE section is added.

21444 The [EINVAL] error condition is made optional. This is for consistency with the DESCRIPTION.

21445 **Issue 7**

21446 SD5-XSH-ERN-148 is applied.

21447 The *aio\_return()* function is moved from the Asynchronous Input and Output option to the Base.

21448 **NAME**

21449 aio\_suspend — wait for an asynchronous I/O request

21450 **SYNOPSIS**

```
21451 #include <aio.h>
21452 int aio_suspend(const struct aiocb *const list[], int nent,
21453               const struct timespec *timeout);
```

21454 **DESCRIPTION**

21455 The *aio\_suspend()* function shall suspend the calling thread until at least one of the asynchronous  
 21456 I/O operations referenced by the *list* argument has completed, until a signal interrupts the  
 21457 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any  
 21458 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,  
 21459 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the  
 21460 function shall return without suspending the calling thread. The *list* argument is an array of  
 21461 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of  
 21462 elements in the array. Each **aiocb** structure pointed to has been used in initiating an  
 21463 asynchronous I/O request via *aio\_read()*, *aio\_write()*, or *lio\_listio()*. This array may contain null  
 21464 pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have  
 21465 not been used in submitting asynchronous I/O, the effect is undefined.

21466 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of  
 21467 the I/O operations referenced by *list* are completed, then *aio\_suspend()* shall return with an error.  
 21468 The clock that is used to measure this time interval shall be the CLOCK\_MONOTONIC clock.

21469 **RETURN VALUE**

21470 If the *aio\_suspend()* function returns after one or more asynchronous I/O operations have  
 21471 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and  
 21472 set *errno* to indicate the error.

21473 The application may determine which asynchronous I/O completed by scanning the associated  
 21474 error and return status using *aio\_error()* and *aio\_return()*, respectively.

21475 **ERRORS**

21476 The *aio\_suspend()* function shall fail if:

21477 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the  
 21478 time interval indicated by *timeout*.

21479 [EINTR] A signal interrupted the *aio\_suspend()* function. Note that, since each  
 21480 asynchronous I/O operation may possibly provoke a signal when it  
 21481 completes, this error return may be caused by the completion of one (or more)  
 21482 of the very I/O operations being awaited.

21483 **EXAMPLES**

21484 None.

21485 **APPLICATION USAGE**

21486 None.

21487 **RATIONALE**

21488 None.

21489 **FUTURE DIRECTIONS**

21490 None.

21491 **SEE ALSO**21492 [aio\\_read\(\)](#), [aio\\_write\(\)](#), [lio\\_listio\(\)](#)21493 XBD <[aio.h](#)>21494 **CHANGE HISTORY**

21495 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21496 **Issue 6**21497 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
21498 implementation does not support the Asynchronous Input and Output option.

21499 The APPLICATION USAGE section is added.

21500 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the  
21501 CLOCK\_MONOTONIC clock, if supported, is used.21502 **Issue 7**21503 The *aio\_suspend()* function is moved from the Asynchronous Input and Output option to the  
21504 Base.21505 **Issue 8**

21506 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

21507 **NAME**

21508 aio\_write — asynchronous write to a file

21509 **SYNOPSIS**

21510 #include &lt;aio.h&gt;

21511 int aio\_write(struct aiocb \*aiocbp);

21512 **DESCRIPTION**

21513 The *aio\_write()* function shall write *aiocbp->aio\_nbytes* to the file associated with  
 21514 *aiocbp->aio\_fildes* from the buffer pointed to by *aiocbp->aio\_buf*. The function shall return when  
 21515 the write request has been initiated or, at a minimum, queued to the file or device.

21516 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted  
 21517 at a priority equal to a base scheduling priority minus *aiocbp->aio\_reqprio*. If Thread Execution  
 21518 Scheduling is not supported, then the base scheduling priority is that of the calling process;  
 21519 PIO TPS otherwise, the base scheduling priority is that of the calling thread.

21520 The *aiocbp* argument may be used as an argument to *aio\_error()* and *aio\_return()* in order to  
 21521 determine the error status and return status, respectively, of the asynchronous operation while it  
 21522 is proceeding.

21523 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio\_buf* or  
 21524 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O  
 21525 completion, then the behavior is undefined.

21526 If O\_APPEND is not set for the file descriptor *aio\_fildes*, then the requested operation shall take  
 21527 place at the absolute position in the file as given by *aio\_offset*, as if *lseek()* were called  
 21528 immediately prior to the operation with an *offset* equal to *aio\_offset* and a *whence* equal to  
 21529 SEEK\_SET. If O\_APPEND is set for the file descriptor, or if *aio\_fildes* is associated with a device  
 21530 that is incapable of seeking, write operations append to the file in the same order as the calls  
 21531 were made, except under circumstances described in Section 2.8.2. After a successful call to  
 21532 enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

21533 The *aio\_sigevent* member specifies the notification which occurs when the request is completed.

21534 The *aiocbp->aio\_lio\_opcode* field shall be ignored by *aio\_write()*.

21535 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

21536 SIO If synchronized I/O is enabled on the file associated with *aiocbp->aio\_fildes*, the behavior of this  
 21537 function shall be according to the definitions of synchronized I/O data integrity completion, and  
 21538 synchronized I/O file integrity completion.

21539 For any system action that changes the process memory space while an asynchronous I/O is  
 21540 outstanding to the address range being changed, the result of that action is undefined.

21541 For regular files, no data transfer shall occur past the offset maximum established in the open  
 21542 file description associated with *aiocbp->aio\_fildes*.

21543 If the request would cause the file size to exceed the soft file size limit for the process and there  
 21544 XSI is no room for any bytes to be written, the request shall fail and the implementation shall  
 21545 generate a SIGXFSZ signal for the thread.

21546 **RETURN VALUE**

21547 The *aio\_write()* function shall return the value zero if the I/O operation is successfully queued;  
 21548 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

21549 **ERRORS**21550 The *aio\_write()* function shall fail if:21551 [EAGAIN] The requested asynchronous I/O operation was not queued due to system  
21552 resource limitations.21553 Each of the following conditions may be detected synchronously at the time of the call to  
21554 *aio\_write()*, or asynchronously. If any of the conditions below are detected synchronously, the  
21555 *aio\_write()* function shall return `-1` and set *errno* to the corresponding value. If any of the  
21556 conditions below are detected asynchronously, the return status of the asynchronous operation  
21557 shall be set to `-1`, and the error status of the asynchronous operation is set to the corresponding  
21558 value.21559 [EBADF] The *aioctx->aio\_fildes* argument is not a valid file descriptor open for writing.21560 [EINVAL] The file offset value implied by *aioctx->aio\_offset* would be invalid,  
21561 PIO *aioctx->aio\_reqprio* is not a valid value, or *aioctx->aio\_nbytes* is an invalid  
21562 value.21563 In the case that the *aio\_write()* successfully queues the I/O operation, the return status of the  
21564 asynchronous operation shall be one of the values normally returned by the *write()* function call.  
21565 If the operation is successfully queued but is subsequently canceled or encounters an error, the  
21566 error status for the asynchronous operation contains one of the values normally set by the  
21567 *write()* function call, or one of the following:21568 [EBADF] The *aioctx->aio\_fildes* argument is not a valid file descriptor open for writing.21569 [EINVAL] The file offset value implied by *aioctx->aio\_offset* would be invalid.21570 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit  
21571 *aio\_cancel()* request.

21572 The following condition may be detected synchronously or asynchronously:

21573 [EFBIG] The file is a regular file, *aioctx->aio\_nbytes* is greater than 0, and the starting  
21574 position is greater than or equal to the offset maximum in the open file  
21575 description associated with *aioctx->aio\_fildes*.21576 [EFBIG] The file is a regular file, *aioctx->aio\_nbytes* is greater than 0, and there is no  
21577 room for any bytes to be written at the starting position without exceeding the  
21578 XSI file size limit for the process. A SIGXFSZ signal shall also be generated for  
21579 the thread.21580 [EFBIG] The file is a regular file, *aioctx->aio\_nbytes* is greater than 0, and there is no  
21581 room for any bytes to be written at the starting position without exceeding the  
21582 implementation-defined maximum file size.21583 **EXAMPLES**

21584 None.

21585 **APPLICATION USAGE**

21586 None.

21587 **RATIONALE**

21588 None.



21589 **FUTURE DIRECTIONS**

21590 None.

21591 **SEE ALSO**21592 Section 2.8.2 (on page 528), *aio\_cancel()*, *aio\_error()*, *aio\_read()*, *aio\_return()*, *close()*, *exec*, *exit()*,  
21593 *fork()*, *lio\_listio()*, *lseek()*, *write()*21594 XBD <**aio.h**>21595 **CHANGE HISTORY**

21596 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

21597 Large File Summit extensions are added.

21598 **Issue 6**21599 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
21600 implementation does not support the Asynchronous Input and Output option.

21601 The APPLICATION USAGE section is added.

21602 The following new requirements on POSIX implementations derive from alignment with the  
21603 Single UNIX Specification:

- 21604 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs  
21605 past the offset maximum established in the open file description associated with  
21606 *aioctx->aio\_fildes*.
- 21607 • The [EFBIG] error is added as part of the large file support extensions.

21608 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/14 is applied, rewording the  
21609 DESCRIPTION when prioritized I/O is supported to account for threads, and removing the  
21610 words “to the calling process” in the RETURN VALUE section.21611 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/15 is applied, updating the [EINVAL]  
21612 error, so that detection of an [EINVAL] error for an invalid value of *aioctx->aio\_reqprio* is only  
21613 required if the Prioritized Input and Output option is supported.21614 **Issue 7**

21615 Austin Group Interpretation 1003.1-2001 #082 is applied.

21616 The *aio\_write()* function is moved from the Asynchronous Input and Output option to the Base.

21617 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0028 [317] is applied.

21618 **Issue 8**

21619 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

21620 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
21621 relating to the file size limit for the process.

21622 **NAME**

21623 alarm — schedule an alarm signal

21624 **SYNOPSIS**

21625 #include &lt;unistd.h&gt;

21626 unsigned alarm(unsigned *seconds*);21627 **DESCRIPTION**

21628 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after  
 21629 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays  
 21630 may prevent the process from handling the signal as soon as it is generated.

21631 If *seconds* is 0, a pending alarm request, if any, is canceled.

21632 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.  
 21633 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time  
 21634 at which the SIGALRM signal is generated.

21635 **RETURN VALUE**

21636 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value  
 21637 that is the number of seconds until the previous request would have generated a SIGALRM  
 21638 signal. Otherwise, *alarm()* shall return 0.

21639 **ERRORS**

21640 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

21641 **EXAMPLES**

21642 None.

21643 **APPLICATION USAGE**

21644 The *fork()* function clears pending alarms in the child process. A new process image created by  
 21645 one of the *exec* functions inherits the time left to an alarm signal in the image of the old process.

21646 Application developers should note that the type of the argument *seconds* and the return value of  
 21647 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces  
 21648 Application cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX},  
 21649 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting  
 21650 its portability. A different type was considered, but historical implementations, including those  
 21651 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

21652 Application developers should be aware of possible interactions when the same process uses  
 21653 both the *alarm()* and *sleep()* functions.

21654 **RATIONALE**

21655 Many historical implementations (including Version 7 and System V) allow an alarm to occur up  
 21656 to a second early. Other implementations allow alarms up to half a second or one clock tick  
 21657 early or do not allow them to occur early at all. The latter is considered most appropriate, since it  
 21658 gives the most predictable behavior, especially since the signal can always be delayed for an  
 21659 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument  
 21660 as the minimum amount of time they wish to have elapse before the signal.

21661 The term “realtime” here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time  
 21662 as common English usage, and has nothing to do with “realtime operating systems”. It is in  
 21663 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

21664 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are  
 21665 silently rounded down to an implementation-specific maximum value. This maximum is large  
 21666 enough (to the order of several months) that the effect is not noticeable.

21667 There were two possible choices for alarm generation in multi-threaded applications: generation  
21668 for the calling thread or generation for the process. The first option would not have been  
21669 particularly useful since the alarm state is maintained on a per-process basis and the alarm that  
21670 is established by the last invocation of *alarm()* is the only one that would be active.

21671 Furthermore, allowing generation of an asynchronous signal for a thread would have  
21672 introduced an exception to the overall signal model. This requires a compelling reason in order  
21673 to be justified.

#### 21674 **FUTURE DIRECTIONS**

21675 None.

#### 21676 **SEE ALSO**

21677 *alarm()*, *exec*, *fork()*, *pause()*, *sigaction()*, *sleep()*, *timer\_create()*

21678 XBD [<signal.h>](#), [<unistd.h>](#)

#### 21679 **CHANGE HISTORY**

21680 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 21681 **Issue 6**

21682 The following new requirements on POSIX implementations derive from alignment with the  
21683 Single UNIX Specification:

- 21684 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*,  
21685 and *usleep()* functions are unspecified.

21686 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/16 is applied, replacing “an  
21687 implementation-defined maximum value” with “an implementation-specific maximum value”  
21688 in the RATIONALE.

#### 21689 **Issue 8**

21690 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

21691 **NAME**

21692 aligned\_alloc — allocate memory with a specified alignment

21693 **SYNOPSIS**

21694 #include &lt;stdlib.h&gt;

21695 void \*aligned\_alloc(size\_t alignment, size\_t size);

21696 **DESCRIPTION**

21697 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21698 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21699 volume of POSIX.1-2024 defers to the ISO C standard.

21700 The *aligned\_alloc()* function shall allocate unused space for an object whose alignment is  
 21701 specified by *alignment*, whose size in bytes is specified by *size*, and whose value is indeterminate.

21702 The order and contiguity of storage allocated by successive calls to *aligned\_alloc()* is unspecified.  
 21703 Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer  
 21704 returned shall point to the start (lowest byte address) of the allocated space. If the value of  
 21705 *alignment* is not a valid alignment supported by the implementation, a null pointer shall be  
 21706 returned. If the space cannot be allocated, a null pointer shall be returned. If the size of the space  
 21707 requested is 0, the behavior is implementation-defined: either a null pointer shall be returned to  
 21708 indicate an error, or the behavior shall be as if the size were some non-zero value, except that the  
 21709 behavior is undefined if the returned pointer is used to access an object.

21710 For purposes of determining the existence of a data race, *aligned\_alloc()* shall behave as though it  
 21711 accessed only memory locations accessible through its arguments and not other static duration  
 21712 storage. The function may, however, visibly modify the storage that it allocates. Calls to  
 21713 ADV *aligned\_alloc()*, *calloc()*, *free()*, *malloc()*, *posix\_memalign()*,  
 21714 CX *reallocarray()*, and *realloc()* that allocate or deallocate a particular region of memory shall occur  
 21715 in a single total order (see [Section 4.15.1](#), on page 100), and each such deallocation call shall  
 21716 synchronize with the next allocation (if any) in this order.

21717 **RETURN VALUE**

21718 Upon successful completion, *aligned\_alloc()* shall return a pointer to the allocated space; if *size* is  
 21719 0, the application shall ensure that the pointer is not used to access an object.

21720 CX Otherwise, it shall return a null pointer and set *errno* to indicate the error.

21721 **ERRORS**

21722 The *aligned\_alloc()* function shall fail if:

21723 CX [EINVAL] The value of *alignment* is not a valid alignment supported by the  
 21724 implementation.

21725 CX [ENOMEM] Insufficient storage space is available.

21726 The *aligned\_alloc()* function may fail if:

21727 CX [EINVAL] *size* is 0 and the implementation does not support 0 sized allocations.

21728 **EXAMPLES**

21729 None.

21730 **APPLICATION USAGE**

21731 None.

21732 **RATIONALE**21733 See the RATIONALE for *malloc()*.21734 **FUTURE DIRECTIONS**

21735 None.

21736 **SEE ALSO**21737 *calloc()*, *free()*, *getrlimit()*, *malloc()*, *posix\_memalign()*, *realloc()*

21738 XBD &lt;stdlib.h&gt;

21739 **CHANGE HISTORY**

21740 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

21741 **NAME**

21742           alphasort, scandir — scan a directory

21743 **SYNOPSIS**

```
21744       #include <dirent.h>
21745       int alphasort(const struct dirent **d1, const struct dirent **d2);
21746       int scandir(const char *dir, struct dirent ***namelist,
21747                   int (*sel)(const struct dirent *),
21748                   int (*compar)(const struct dirent **, const struct dirent **));
```

21749 **DESCRIPTION**

21750       The *alphasort()* function can be used as the comparison function for the *scandir()* function to sort  
 21751       the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the  
 21752       *strcoll()* function on the *d\_name* element of the **dirent** structures passed as the two parameters. If  
 21753       the *strcoll()* function fails, the return value of *alphasort()* is unspecified.

21754       The *alphasort()* function shall not change the setting of *errno* if successful. Since no return value  
 21755       is reserved to indicate an error, an application wishing to check for error situations should set  
 21756       *errno* to 0, then call *alphasort()*, then check *errno*.

21757       The *scandir()* function shall scan the directory *dir*, calling the function referenced by *sel* on each  
 21758       directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored  
 21759       in strings allocated as if by a call to *malloc()*, and sorted as if by a call to *qsort()* with the  
 21760       comparison function *compar*, except that *compar* need not provide total ordering. The strings are  
 21761       collected in array *namelist* which shall be allocated as if by a call to *malloc()*. If *sel* is a null  
 21762       pointer, all entries shall be selected. If the comparison function *compar* does not provide total  
 21763       ordering, the order in which the directory entries are stored is unspecified.

21764 **RETURN VALUE**

21765       Upon successful completion, the *alphasort()* function shall return an integer greater than, equal  
 21766       to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is  
 21767       lexically greater than, equal to, or less than the directory pointed to by *d2* when both are  
 21768       interpreted as appropriate to the current locale. There is no return value reserved to indicate an  
 21769       error.

21770       Upon successful completion, the *scandir()* function shall return the number of entries in the  
 21771       array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir()*  
 21772       function shall return -1.

21773 **ERRORS**21774       The *scandir()* function shall fail if:

21775       [EACCES]       Search permission is denied for the component of the path prefix of *dir* or read  
 21776       permission is denied for *dir*.

21777       [ELOOP]        A loop exists in symbolic links encountered during resolution of the *dir*  
 21778       argument.

21779       [ENAMETOOLONG]       The length of a component of a pathname is longer than {NAME\_MAX}.

21781       [ENOENT]        A component of *dir* does not name an existing directory or *dir* is an empty  
 21782       string.

21783       [ENOMEM]        Insufficient storage space is available.

21784	[ENOTDIR]	A component of <i>dir</i> names an existing file that is neither a directory nor a symbolic link to a directory.
21785		
21786	[EOVERFLOW]	One of the values to be returned or passed to a callback function cannot be represented correctly.
21787		
21788		The <i>scandir()</i> function may fail if:
21789	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>dir</i> argument.
21790		
21791	[EMFILE]	All file descriptors available to the process are currently open.
21792	[ENAMETOOLONG]	
21793		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
21794		
21795		
21796	[ENFILE]	Too many files are currently open in the system.

**EXAMPLES**

An example to print the files in the current directory:

```

21799 #include <dirent.h>
21800 #include <stdio.h>
21801 #include <stdlib.h>
21802 ...
21803 struct dirent **namelist;
21804 int i,n;

21805     n = scandir(".", &namelist, 0, alphasort);
21806     if (n < 0)
21807         perror("scandir");
21808     else {
21809         for (i = 0; i < n; i++) {
21810             printf("%s\n", namelist[i]->d_name);
21811             free(namelist[i]);
21812         }
21813     }
21814     free(namelist);
21815     ...

```

**APPLICATION USAGE**

If *dir* contains filenames that do not form character strings, or which contain characters outside the domain of the collating sequence of the current locale, the *alphasort()* function need not provide a total ordering. This condition is not possible if all filenames within the directory consist only of characters from the portable filename character set.

The *scandir()* function may allocate dynamic storage during its operation. If *scandir()* is forcibly terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *sel* or *compar*, or by an interrupt routine, *scandir()* does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, then wait until *scandir()* returns to act on the interrupt.

For functions that allocate memory as if by *malloc()*, the application should release such memory when it is no longer required by a call to *free()*. For *scandir()*, this is *namelist* (including all of the individual strings in *namelist*).

21829 **RATIONALE**

21830       None.

21831 **FUTURE DIRECTIONS**

21832       None.

21833 **SEE ALSO**21834       *qsort()*, *strcoll()*21835       XBD <**dirent.h**>21836 **CHANGE HISTORY**

21837       First released in Issue 7.

21838       POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0029 [324], XSH/TC1-2008/0030 [404],  
21839       XSH/TC1-2008/0031 [393], and XSH/TC1-2008/0032 [291] are applied.



21840 **NAME**

21841 asctime — convert date and time to a string

21842 **SYNOPSIS**

```
21843 OB #include <time.h>
21844 char *asctime(const struct tm *timeptr);
```

21845 **DESCRIPTION**

21846 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21847 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21848 volume of POSIX.1-2024 defers to the ISO C standard.

21849 The *asctime()* function shall convert the broken-down time in the structure pointed to by *timeptr*  
 21850 into a string in the form:

```
21851 Sun Sep 16 01:03:52 1973\n\0
```

21852 using the equivalent of the following algorithm:

```
21853 char *asctime(const struct tm *timeptr)
21854 {
21855     static char wday_name[7][3] = {
21856         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
21857     };
21858     static char mon_name[12][3] = {
21859         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
21860         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
21861     };
21862     static char result[26];
21863     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
21864         wday_name[timeptr->tm_wday],
21865         mon_name[timeptr->tm_mon],
21866         timeptr->tm_mday, timeptr->tm_hour,
21867         timeptr->tm_min, timeptr->tm_sec,
21868         1900 + timeptr->tm_year);
21869     return result;
21870 }
```

21871 If any of the members of the broken-down time contain values that are outside their normal  
 21872 ranges (see XBD [<time.h>](#)), the behavior of the *asctime()* function is undefined. Likewise, if the  
 21873 calculated year exceeds four digits or is less than the year 1000, the behavior is undefined.

21874 The **tm** structure is defined in the [<time.h>](#) header.

21875 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static  
 21876 objects: a broken-down time structure and an array of type **char**. Execution of any of the  
 21877 functions that return a pointer to one of these object types may overwrite the information in any  
 21878 object of the same type pointed to by the value returned from any previous call to any of them.

21879 The *asctime()* function need not be thread-safe; however, *asctime()* shall avoid data races with all  
 21880 functions other than itself, *ctime()*, *gmtime()*, and *localtime()*.

21881 **RETURN VALUE**

21882 CX Upon successful completion, *asctime()* shall return a pointer to the string. If the function is  
21883 unsuccessful, it shall return NULL.

21884 **ERRORS**

21885 No errors are defined.

21886 **EXAMPLES**

21887 None.

21888 **APPLICATION USAGE**

21889 This function is included only for compatibility with older implementations. It has undefined  
21890 behavior if the resulting string would be too long, so the use of this function should be  
21891 discouraged. On implementations that do not detect output string length overflow, it is possible  
21892 to overflow the output buffer in such a way as to cause applications to fail, or possible system  
21893 security violations. Also, this function does not support localized date and time formats. To  
21894 avoid these problems, applications should use *strftime()* to generate strings from broken-down  
21895 times.

21896 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

21897 **RATIONALE**

21898 The standard developers decided to mark the *asctime()* function obsolescent even though it is in  
21899 the ISO C standard due to the possibility of buffer overflow. The ISO C standard also provides  
21900 the *strftime()* function which can be used to avoid these problems.

21901 **FUTURE DIRECTIONS**

21902 This function may be removed in a future version, but not until after it has been removed from  
21903 the ISO C standard.

21904 **SEE ALSO**

21905 *clock()*, *ctime()*, *difftime()*, *futimens()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*

21906 XBD <time.h>

21907 **CHANGE HISTORY**

21908 First released in Issue 1. Derived from Issue 1 of the SVID.

21909 **Issue 5**

21910 Normative text previously in the APPLICATION USAGE section is moved to the  
21911 DESCRIPTION.

21912 The *asctime\_r()* function is included for alignment with the POSIX Threads Extension.

21913 A note indicating that the *asctime()* function need not be reentrant is added to the  
21914 DESCRIPTION.

21915 **Issue 6**

21916 The *asctime\_r()* function is marked as part of the Thread-Safe Functions option.

21917 Extensions beyond the ISO C standard are marked.

21918 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
21919 its avoidance of possibly using a static data area.

21920 The DESCRIPTION of *asctime\_r()* is updated to describe the format of the string returned.

21921 The **restrict** keyword is added to the *asctime\_r()* prototype for alignment with the  
21922 ISO/IEC 9899:1999 standard

- 21923 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/17 is applied, adding the CX extension in  
21924 the RETURN VALUE section requiring that if the *asctime()* function is unsuccessful it returns  
21925 NULL.
- 21926 **Issue 7**
- 21927 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.
- 21928 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 21929 The *asctime\_r()* function is moved from the Thread-Safe Functions option to the Base.
- 21930 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0033 [86,429] is applied.
- 21931 **Issue 8**
- 21932 Austin Group Defect 469 is applied, clarifying the conditions under which the behavior of  
21933 *asctime()* is undefined.
- 21934 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
21935 standard.
- 21936 Austin Group Defect 1330 is applied, changing the FUTURE DIRECTIONS section.
- 21937 Austin Group Defect 1376 is applied, removing CX shading from some text derived from the  
21938 ISO C standard and updating it to match the ISO C standard.
- 21939 Austin Group Defect 1410 is applied, removing the *asctime\_r()* function.

21940 **NAME**

21941 asin, asinf, asinl — arc sine function

21942 **SYNOPSIS**

```
21943 #include <math.h>
21944 double asin(double x);
21945 float asinf(float x);
21946 long double asinl(long double x);
```

21947 **DESCRIPTION**

21948 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 21949 conflict between the requirements described here and the ISO C standard is unintentional. This  
 21950 volume of POSIX.1-2024 defers to the ISO C standard.

21951 These functions shall compute the principal value of the arc sine of their argument  $x$ . The value  
 21952 of  $x$  should be in the range  $[-1,1]$ .

21953 An application wishing to check for error situations should set *errno* to zero and call  
 21954 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 21955 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 21956 zero, an error has occurred.

21957 **RETURN VALUE**

21958 Upon successful completion, these functions shall return the arc sine of  $x$ , in the range  
 21959  $[-\pi/2, \pi/2]$  radians.

21960 MX For finite values of  $x$  not in the range  $[-1,1]$ , a domain error shall occur, and either a NaN (if  
 21961 supported), or an implementation-defined value shall be returned.

21962 MX If  $x$  is NaN, a NaN shall be returned.

21963 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

21964 If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and a NaN shall be returned.

21965 MXX If  $x$  is subnormal,  $x$  should be returned.

21966 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *asin()*, *asinf()*, and *asinl()*  
 21967 shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
 21968 FLT\_MIN, and LDBL\_MIN, respectively.

21969 **ERRORS**

21970 These functions shall fail if:

21971 MX **Domain Error** The  $x$  argument is finite and is not in the range  $[-1,1]$ , or is  $\pm\text{Inf}$ .

21972 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 21973 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 21974 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 21975 shall be raised.

21976 These functions may fail if:

21977 MX **Range Error** The value of  $x$  is subnormal.

21978 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 21979 then *errno* shall be set to [ERANGE]. If the integer expression  
 21980 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 21981 floating-point exception shall be raised.

21982 **EXAMPLES**

21983 None.

21984 **APPLICATION USAGE**

21985 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
21986 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21987 **RATIONALE**

21988 None.

21989 **FUTURE DIRECTIONS**

21990 None.

21991 **SEE ALSO**21992 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [sin\(\)](#)21993 XBD Section 4.23 (on page 109), [<math.h>](#)21994 **CHANGE HISTORY**

21995 First released in Issue 1. Derived from Issue 1 of the SVID.

21996 **Issue 5**

21997 The DESCRIPTION is updated to indicate how an application should check for an error. This  
21998 text was previously published in the APPLICATION USAGE section.

21999 **Issue 6**22000 The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

22001 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
22002 revised to align with the ISO/IEC 9899:1999 standard.

22003 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
22004 marked.

22005 **Issue 7**

22006 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0034 [320] and XSH/TC1-2008/0035  
22007 [68] are applied.

22008 **Issue 8**

22009 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is  
22010 subnormal to avoid the need for two shading changes.

22011 **NAME**

22012 asinh, asinhf, asinhl — inverse hyperbolic sine functions

22013 **SYNOPSIS**

```
22014 #include <math.h>
22015 double asinh(double x);
22016 float asinhf(float x);
22017 long double asinhl(long double x);
```

22018 **DESCRIPTION**

22019 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22020 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22021 volume of POSIX.1-2024 defers to the ISO C standard.

22022 These functions shall compute the inverse hyperbolic sine of their argument  $x$ .

22023 An application wishing to check for error situations should set *errno* to zero and call  
 22024 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22025 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22026 zero, an error has occurred.

22027 **RETURN VALUE**

22028 Upon successful completion, these functions shall return the inverse hyperbolic sine of their  
 22029 argument.

22030 MX If  $x$  is NaN, a NaN shall be returned.

22031 If  $x$  is  $\pm 0$ , or  $\pm\text{Inf}$ ,  $x$  shall be returned.

22032 MXX If  $x$  is subnormal,  $x$  should be returned.

22033 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *asinh()*, *asinhf()*, and *asinhl()*  
 22034 shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
 22035 FLT\_MIN, and LDBL\_MIN, respectively.

22036 **ERRORS**

22037 These functions may fail if:

22038 MX **Range Error** The value of  $x$  is subnormal.

22039 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 22040 then *errno* shall be set to [ERANGE]. If the integer expression  
 22041 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 22042 floating-point exception shall be raised.

22043 **EXAMPLES**

22044 None.

22045 **APPLICATION USAGE**

22046 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 22047 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22048 **RATIONALE**

22049 None.

22050 **FUTURE DIRECTIONS**

22051 None.

22052 **SEE ALSO**22053 *feclearexcept()*, *fetestexcept()*, *sinh()*

22054 XBD Section 4.23 (on page 109), &lt;math.h&gt;

22055 **CHANGE HISTORY**

22056 First released in Issue 4, Version 2.

22057 **Issue 5**

22058 Moved from X/OPEN UNIX extension to BASE.

22059 **Issue 6**22060 The *asinh()* function is no longer marked as an extension.22061 The *asinhf()* and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

22063 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

22065 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

22067 **Issue 7**

22068 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0036 [68] is applied.

22069 **Issue 8**22070 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is subnormal to avoid the need for two shading changes.

22072 **NAME**22073        **asinl** — arc sine function22074 **SYNOPSIS**

22075        #include &lt;math.h&gt;

22076        long double asinl(long double *x*);22077 **DESCRIPTION**22078        Refer to *asin()*.



22079 **NAME**

22080       asprintf — print formatted output

22081 **SYNOPSIS**

22082       #include &lt;stdio.h&gt;

22083       int asprintf(char \*\*restrict *ptr*, const char \*restrict *format*, ...);22084 **DESCRIPTION**22085       Refer to *fprintf()*.

22086 **NAME**

22087       assert — insert program diagnostics

22088 **SYNOPSIS**

22089       #include &lt;assert.h&gt;

22090       void assert(*scalar expression*);22091 **DESCRIPTION**

22092 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 22093 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22094 volume of POSIX.1-2024 defers to the ISO C standard.

22095       The *assert()* macro shall insert diagnostics into programs; it shall expand to a **void** expression.  
 22096 When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal  
 22097 to 0), *assert()* shall write information about the particular call that failed on *stderr* and shall call  
 22098 *abort()*.

22099       The information written about the call that failed shall include the text of the argument, the  
 22100 name of the source file, the source file line number, and the name of the enclosing function; the  
 22101 latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of  
 22102 the identifier `__func__`.

22103       Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the  
 22104 preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,  
 22105 shall stop assertions from being compiled into the program.

22106 **RETURN VALUE**22107       The *assert()* macro shall not return a value.22108 **ERRORS**

22109       No errors are defined.

22110 **EXAMPLES**

22111       None.

22112 **APPLICATION USAGE**

22113       None.

22114 **RATIONALE**

22115       None.

22116 **FUTURE DIRECTIONS**

22117       None.

22118 **SEE ALSO**22119       *abort()*, *stdin*

22120       XBD &lt;assert.h&gt;

22121 **CHANGE HISTORY**

22122       First released in Issue 1. Derived from Issue 1 of the SVID.

22123 **Issue 6**

22124       The prototype for the *expression* argument to *assert()* is changed from **int** to **scalar** for alignment  
 22125 with the ISO/IEC 9899:1999 standard.

22126       The DESCRIPTION of *assert()* is updated for alignment with the ISO/IEC 9899:1999 standard.

22127 **NAME**

22128           at\_quick\_exit — register a function to to be called from *quick\_exit()*

22129 **SYNOPSIS**

22130           #include <stdlib.h>

22131           int at\_quick\_exit(void (\*func)(void));

22132 **DESCRIPTION**

22133 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
22134 conflict between the requirements described here and the ISO C standard is unintentional. This  
22135 volume of POSIX.1-2024 defers to the ISO C standard.

22136       The *at\_quick\_exit()* function shall register the function pointed to by *func*, to be called without  
22137 arguments should *quick\_exit()* be called. It is unspecified whether a call to the *at\_quick\_exit()*  
22138 function that does not happen before the *quick\_exit()* function is called will succeed.

22139       At least 32 functions can be registered with *at\_quick\_exit()*.

22140 **RETURN VALUE**

22141       Upon successful completion, *at\_quick\_exit()* shall return 0; otherwise, it shall return a non-zero  
22142 value.

22143 **ERRORS**

22144       No errors are defined.

22145 **EXAMPLES**

22146       None.

22147 **APPLICATION USAGE**

22148       The *at\_quick\_exit()* function registrations are distinct from the *atexit()* registrations, so  
22149 applications might need to call both registration functions with the same argument.

22150       The functions registered by a call to *at\_quick\_exit()* must return to ensure that all registered  
22151 functions are called.

22152       The application should call *sysconf()* to obtain the value of {ATEXIT\_MAX}, the number of  
22153 functions that can be registered. There is no way for an application to tell how many functions  
22154 have already been registered with *at\_quick\_exit()*.

22155       Since the behavior is undefined if the *quick\_exit()* function is called more than once, portable  
22156 applications calling *at\_quick\_exit()* must ensure that the *quick\_exit()* function is not called when  
22157 the functions registered by the *at\_quick\_exit()* function are called.

22158       If a function registered by the *at\_quick\_exit()* function is called and a portable application needs  
22159 to stop further *quick\_exit()* processing, it must call the *\_exit()* function or the *\_Exit()* function or  
22160 one of the functions which cause abnormal process termination.

22161 **RATIONALE**

22162       None.

22163 **FUTURE DIRECTIONS**

22164       None.

22165 **SEE ALSO**

22166       *atexit()*, *exec*, *exit()*, *quick\_exit()*, *sysconf()*

22167       XBD <stdlib.h>

22168 **CHANGE HISTORY**

22169 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22170 **NAME**

22171 atan, atanf, atanl — arc tangent function

22172 **SYNOPSIS**

```
22173 #include <math.h>
22174 double atan(double x);
22175 float atanf(float x);
22176 long double atanl(long double x);
```

22177 **DESCRIPTION**

22178 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22179 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22180 volume of POSIX.1-2024 defers to the ISO C standard.

22181 These functions shall compute the principal value of the arc tangent of their argument  $x$ .

22182 An application wishing to check for error situations should set *errno* to zero and call  
 22183 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22184 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22185 zero, an error has occurred.

22186 **RETURN VALUE**

22187 Upon successful completion, these functions shall return the arc tangent of  $x$  in the range  
 22188  $[-\pi/2, \pi/2]$  radians.

22189 MX If  $x$  is NaN, a NaN shall be returned.

22190 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

22191 If  $x$  is  $\pm\text{Inf}$ ,  $\pm\pi/2$  shall be returned.

22192 MXX If  $x$  is subnormal,  $x$  should be returned.

22193 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *atan()*, *atanf()*, and *atanl()*  
 22194 shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
 22195 FLT\_MIN, and LDBL\_MIN, respectively.

22196 **ERRORS**

22197 These functions may fail if:

22198 MX **Range Error** The value of  $x$  is subnormal.

22199 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 22200 then *errno* shall be set to [ERANGE]. If the integer expression  
 22201 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 22202 floating-point exception shall be raised.

22203 **EXAMPLES**

22204 None.

22205 **APPLICATION USAGE**

22206 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 22207 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22208 **RATIONALE**

22209 None.

22210 **FUTURE DIRECTIONS**

22211 None.

22212 **SEE ALSO**22213 [atan2\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [tan\(\)](#)22214 XBD [Section 4.23](#) (on page 109), [<math.h>](#)22215 **CHANGE HISTORY**

22216 First released in Issue 1. Derived from Issue 1 of the SVID.

22217 **Issue 5**22218 The DESCRIPTION is updated to indicate how an application should check for an error. This  
22219 text was previously published in the APPLICATION USAGE section.22220 **Issue 6**22221 The [atanf\(\)](#) and [atanl\(\)](#) functions are added for alignment with the ISO/IEC 9899:1999 standard.22222 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
22223 revised to align with the ISO/IEC 9899:1999 standard.22224 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
22225 marked.22226 **Issue 7**

22227 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0037 [68] is applied.

22228 **Issue 8**22229 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when  $x$  is  
22230 subnormal to avoid the need for two shading changes.

22231 **NAME**

22232 atan2, atan2f, atan2l — arc tangent functions

22233 **SYNOPSIS**

22234 #include &lt;math.h&gt;

22235 double atan2(double y, double x);

22236 float atan2f(float y, float x);

22237 long double atan2l(long double y, long double x);

22238 **DESCRIPTION**

22239 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22240 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22241 volume of POSIX.1-2024 defers to the ISO C standard.

22242 These functions shall compute the principal value of the arc tangent of  $y/x$ , using the signs of  
 22243 both arguments to determine the quadrant of the return value.

22244 An application wishing to check for error situations should set *errno* to zero and call  
 22245 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 22246 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 22247 zero, an error has occurred.

22248 **RETURN VALUE**

22249 Upon successful completion, these functions shall return the arc tangent of  $y/x$  in the range  
 22250  $[-\pi, \pi]$  radians.

22251 If  $y$  is  $\pm 0$  and  $x$  is  $< 0$ ,  $\pm\pi$  shall be returned.22252 If  $y$  is  $\pm 0$  and  $x$  is  $> 0$ ,  $\pm 0$  shall be returned.22253 If  $y$  is  $< 0$  and  $x$  is  $\pm 0$ ,  $-\pi/2$  shall be returned.22254 If  $y$  is  $> 0$  and  $x$  is  $\pm 0$ ,  $\pi/2$  shall be returned.22255 If  $x$  is 0, a pole error shall not occur.22256 MX If either  $x$  or  $y$  is NaN, a NaN shall be returned.

22257 If the correct value would cause underflow, a range error may occur, and *atan*(*y*), *atan2f*(*y*), and  
 22258 *atan2l*(*y*) shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
 22259 MXX FLT\_MIN, and LDBL\_MIN, respectively. If the IEC 60559 Floating-Point option is supported,  
 22260  $y/x$  should be returned.

22261 MX If  $y$  is  $\pm 0$  and  $x$  is  $-0$ ,  $\pm\pi$  shall be returned.22262 If  $y$  is  $\pm 0$  and  $x$  is  $+0$ ,  $\pm 0$  shall be returned.22263 For finite values of  $\pm y > 0$ , if  $x$  is  $-\text{Inf}$ ,  $\pm\pi$  shall be returned.22264 For finite values of  $\pm y > 0$ , if  $x$  is  $+\text{Inf}$ ,  $\pm 0$  shall be returned.22265 For finite values of  $x$ , if  $y$  is  $\pm\text{Inf}$ ,  $\pm\pi/2$  shall be returned.22266 If  $y$  is  $\pm\text{Inf}$  and  $x$  is  $-\text{Inf}$ ,  $\pm 3\pi/4$  shall be returned.22267 If  $y$  is  $\pm\text{Inf}$  and  $x$  is  $+\text{Inf}$ ,  $\pm\pi/4$  shall be returned.

22268 If both arguments are 0, a domain error shall not occur.

22269 **ERRORS**

22270 These functions may fail if:

22271 MX **Range Error** The result underflows.

22272 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 22273 then *errno* shall be set to [ERANGE]. If the integer expression  
 22274 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 22275 floating-point exception shall be raised.

22276 **EXAMPLES**22277 **Converting Cartesian to Polar Coordinates System**

22278 The function below uses *atan2()* to convert a 2d vector expressed in cartesian coordinates (*x,y*) to  
 22279 the polar coordinates (*rho,theta*). There are other ways to compute the angle *theta*, using *asin()*  
 22280 *acos()*, or *atan()*. However, *atan2()* presents here two advantages:

- 22281 • The angle's quadrant is automatically determined.
- 22282 • The singular cases (0,*y*) are taken into account.

22283 Finally, this example uses *hypot()* rather than *sqrt()* since it is better for special cases; see *hypot()*  
 22284 for more information.

```
22285 #include <math.h>
22286
22287 void
22288 cartesian_to_polar(const double x, const double y,
22289                  double *rho, double *theta
22290                  )
22291 {
22292     *rho = hypot (x,y); /* better than sqrt(x*x+y*y) */
22293     *theta = atan2 (y,x);
22294 }
```

22294 **APPLICATION USAGE**

22295 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 22296 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

22297 **RATIONALE**

22298 None.

22299 **FUTURE DIRECTIONS**

22300 None.

22301 **SEE ALSO**22302 [acos\(\)](#), [asin\(\)](#), [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [hypot\(\)](#), [isnan\(\)](#), [sqrt\(\)](#), [tan\(\)](#)22303 XBD Section 4.23 (on page 109), [<math.h>](#)22304 **CHANGE HISTORY**

22305 First released in Issue 1. Derived from Issue 1 of the SVID.

22306 **Issue 5**

22307 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 22308 text was previously published in the APPLICATION USAGE section.



22309 **Issue 6**

22310 The *atan2f()* and *atan2l()* functions are added for alignment with the ISO/IEC 9899:1999  
22311 standard.

22312 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
22313 revised to align with the ISO/IEC 9899:1999 standard, and the IEC 60559:1989 standard  
22314 floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

22315 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/18 is applied, adding to the EXAMPLES  
22316 section.

22317 **Issue 7**

22318 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0038 [68,428] is applied.

22319 **Issue 8**

22320 Austin Group Defect 1178 is applied, removing MX shading from a paragraph in the RETURN  
22321 VALUE section and joining it with the following paragraph.

22322 **NAME**

22323           atanf — arc tangent function

22324 **SYNOPSIS**

22325           #include <math.h>

22326           float atanf(float x);

22327 **DESCRIPTION**

22328           Refer to *atan()*.

22329 **NAME**

22330           atanh, atanhf, atanh1 — inverse hyperbolic tangent functions

22331 **SYNOPSIS**

```
22332       #include <math.h>
22333       double atanh(double x);
22334       float atanhf(float x);
22335       long double atanh1(long double x);
```

22336 **DESCRIPTION**

22337 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
22338 conflict between the requirements described here and the ISO C standard is unintentional. This  
22339 volume of POSIX.1-2024 defers to the ISO C standard.

22340       These functions shall compute the inverse hyperbolic tangent of their argument  $x$ .

22341       An application wishing to check for error situations should set *errno* to zero and call  
22342 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
22343 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
22344 zero, an error has occurred.

22345 **RETURN VALUE**

22346       Upon successful completion, these functions shall return the inverse hyperbolic tangent of their  
22347 argument.

22348       If  $x$  is  $\pm 1$ , a pole error shall occur, and *atanh*( $x$ ), *atanhf*( $x$ ), and *atanh1*( $x$ ) shall return the value of the  
22349 macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively, with the same sign as the  
22350 correct value of the function.

22351 MX       For finite  $|x| > 1$ , a domain error shall occur, and either a NaN (if supported), or an  
22352 implementation-defined value shall be returned.

22353 MX       If  $x$  is NaN, a NaN shall be returned.

22354       If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

22355       If  $x$  is  $\pm \text{Inf}$ , a domain error shall occur, and a NaN shall be returned.

22356 MXX      If  $x$  is subnormal,  $x$  should be returned.

22357 MX       If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *atanh*( $x$ ), *atanhf*( $x$ ), and *atanh1*( $x$ )  
22358 shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
22359 FLT\_MIN, and LDBL\_MIN, respectively.

22360 **ERRORS**

22361       These functions shall fail if:

22362 MX       Domain Error    The  $x$  argument is finite and not in the range  $[-1, 1]$ , or is  $\pm \text{Inf}$ .

22363           If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
22364 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
22365 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
22366 shall be raised.

22367       Pole Error        The  $x$  argument is  $\pm 1$ .

22368           If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
22369 then *errno* shall be set to [ERANGE]. If the integer expression  
22370 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
22371 floating-point exception shall be raised.

22372 These functions may fail if:

22373 MX **Range Error** The value of  $x$  is subnormal.

22374 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 22375 then *errno* shall be set to [ERANGE]. If the integer expression  
 22376 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 22377 floating-point exception shall be raised.

#### 22378 EXAMPLES

22379 None.

#### 22380 APPLICATION USAGE

22381 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 22382 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

#### 22383 RATIONALE

22384 None.

#### 22385 FUTURE DIRECTIONS

22386 None.

#### 22387 SEE ALSO

22388 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [tanh\(\)](#)

22389 XBD Section 4.23 (on page 109), [<math.h>](#)

#### 22390 CHANGE HISTORY

22391 First released in Issue 4, Version 2.

#### 22392 Issue 5

22393 Moved from X/OPEN UNIX extension to BASE.

#### 22394 Issue 6

22395 The *atanh()* function is no longer marked as an extension.

22396 The *atanhf()* and *atanhl()* functions are added for alignment with the ISO/IEC 9899:1999  
 22397 standard.

22398 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 22399 revised to align with the ISO/IEC 9899:1999 standard.

22400 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 22401 marked.

#### 22402 Issue 7

22403 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0039 [320] and XSH/TC1-2008/0040  
 22404 [680] are applied.

#### 22405 Issue 8

22406 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when  $x$  is  
 22407 subnormal to avoid the need for two shading changes.

22408 **NAME**

22409           atanl — arc tangent function

22410 **SYNOPSIS**

22411           #include <math.h>

22412           long double atanl(long double x);

22413 **DESCRIPTION**

22414           Refer to *atan()*.

22415 **NAME**

22416 atexit — register a function to be called from *exit()* or after return from *main()*

22417 **SYNOPSIS**

22418 #include <stdlib.h>

22419 int atexit(void (\*func)(void));

22420 **DESCRIPTION**

22421 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22422 conflict between the requirements described here and the ISO C standard is unintentional. This  
22423 volume of POSIX.1-2024 defers to the ISO C standard.

22424 The *atexit()* function shall register the function pointed to by *func*, to be called without  
22425 arguments from *exit()*, or after return from the initial call to *main()*, or on the last thread  
22426 termination. If the *exit()* function is called, it is unspecified whether a call to the *atexit()* function  
22427 that does not happen before *exit()* is called will succeed.

22428 At least 32 functions can be registered with *atexit()*.

22429 **RETURN VALUE**

22430 Upon successful completion, *atexit()* shall return 0; otherwise, it shall return a non-zero value.

22431 **ERRORS**

22432 No errors are defined.

22433 **EXAMPLES**

22434 None.

22435 **APPLICATION USAGE**

22436 The *atexit()* function registrations are distinct from the *at\_quick\_exit()* registrations, so  
22437 applications might need to call both registration functions with the same argument.

22438 The functions registered by a call to *atexit()* must return to ensure that all registered functions  
22439 are called.

22440 The application should call *sysconf()* to obtain the value of {ATEXIT\_MAX}, the number of  
22441 functions that can be registered. There is no way for an application to tell how many functions  
22442 have already been registered with *atexit()*.

22443 Since the behavior is undefined if the *exit()* function is called more than once, portable  
22444 applications calling *atexit()* must ensure that the *exit()* function is not called when the functions  
22445 registered by the *atexit()* function are called.

22446 If a function registered by the *atexit()* function is called and a portable application needs to stop  
22447 further *exit()* processing, it must call the *\_exit()* function or the *\_Exit()* function or one of the  
22448 functions which cause abnormal process termination.

22449 **RATIONALE**

22450 None.

22451 **FUTURE DIRECTIONS**

22452 None.

22453 **SEE ALSO**

22454 [at\\_quick\\_exit\(\)](#), [exec](#), [exit\(\)](#), [sysconf\(\)](#)

22455 XBD [<stdlib.h>](#)

22456 **CHANGE HISTORY**

22457 First released in Issue 4. Derived from the ANSI C standard.

22458 **Issue 6**

22459 Extensions beyond the ISO C standard are marked.

22460 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

22461 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/19 is applied, adding further clarification  
22462 to the APPLICATION USAGE section.

22463 **Issue 8**

22464 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
22465 standard.

22466 Austin Group Defect 1646 is applied, removing redundant text relating to the *exec* family of  
22467 functions.

22468 **NAME**

22469            atof — convert a string to a double-precision number

22470 **SYNOPSIS**

22471            #include &lt;stdlib.h&gt;

22472            double atof(const char \*str);

22473 **DESCRIPTION**

22474 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
22475            conflict between the requirements described here and the ISO C standard is unintentional. This  
22476            volume of POSIX.1-2024 defers to the ISO C standard.

22477            The call *atof(str)* shall be equivalent to:

22478            strtod(str, (char \*\*) NULL),

22479            except that the handling of errors may differ. If the value cannot be represented, the behavior is  
22480            undefined.

22481 **RETURN VALUE**22482            The *atof()* function shall return the converted value if the value can be represented.22483 **ERRORS**

22484            No errors are defined.

22485 **EXAMPLES**

22486            None.

22487 **APPLICATION USAGE**

22488            The *atof()* function is subsumed by *strtod()* but is retained because it is used extensively in  
22489            existing code. If the number is not known to be in range, *strtod()* should be used because *atof()*  
22490            is not required to perform any error checking.

22491 **RATIONALE**

22492            None.

22493 **FUTURE DIRECTIONS**

22494            None.

22495 **SEE ALSO**22496            [strtod\(\)](#)22497            XBD [<stdlib.h>](#)22498 **CHANGE HISTORY**

22499            First released in Issue 1. Derived from Issue 1 of the SVID.



22500 **NAME**

22501            atoi — convert a string to an integer

22502 **SYNOPSIS**

22503            #include &lt;stdlib.h&gt;

22504            int atoi(const char \*str);

22505 **DESCRIPTION**

22506 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
22507            conflict between the requirements described here and the ISO C standard is unintentional. This  
22508            volume of POSIX.1-2024 defers to the ISO C standard.

22509        The call *atoi(str)* shall be equivalent to:

22510            (int) strtol(str, (char \*\*)NULL, 10)

22511        except that the handling of errors may differ. If the value cannot be represented, the behavior is  
22512        undefined.22513 **RETURN VALUE**22514        The *atoi()* function shall return the converted value if the value can be represented.22515 **ERRORS**

22516        No errors are defined.

22517 **EXAMPLES**22518            **Converting an Argument**

22519        The following example checks for proper usage of the program. If there is an argument and the  
22520        decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program  
22521        has a valid number of minutes to wait for an event.

22522            #include &lt;stdlib.h&gt;

22523            #include &lt;stdio.h&gt;

22524            ...

22525            int minutes\_to\_event;

22526            ...

22527            if (argc < 2 || (minutes\_to\_event = atoi (argv[1])) <= 0) {  
22528                fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);

22529            }

22530            ...

22531 **APPLICATION USAGE**

22532        The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in  
22533        existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is  
22534        not required to perform any error checking.

22535 **RATIONALE**

22536        None.

22537 **FUTURE DIRECTIONS**

22538        None.

22539 **SEE ALSO**22540            [strtol\(\)](#)22541            XBD [<stdlib.h>](#)

22542 **CHANGE HISTORY**

22543 First released in Issue 1. Derived from Issue 1 of the SVID.

22544 **Issue 8**

22545 Austin Group Defect 1541 is applied, changing the EXAMPLES section.

22546 **NAME**

22547 atol, atoll — convert a string to a long integer

22548 **SYNOPSIS**

22549 #include &lt;stdlib.h&gt;

22550 long atol(const char \*nptr);

22551 long long atoll(const char \*nptr);

22552 **DESCRIPTION**

22553 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22554 conflict between the requirements described here and the ISO C standard is unintentional. This  
22555 volume of POSIX.1-2024 defers to the ISO C standard.

22556 Except as noted below, the call *atol(nptr)* shall be equivalent to:22557 `strtol(nptr, (char **)NULL, 10)`22558 Except as noted below, the call to *atoll(nptr)* shall be equivalent to:22559 `strtoll(nptr, (char **)NULL, 10)`

22560 The handling of errors may differ. If the value cannot be represented, the behavior is undefined.

22561 **RETURN VALUE**

22562 These functions shall return the converted value if the value can be represented.

22563 **ERRORS**

22564 No errors are defined.

22565 **EXAMPLES**

22566 None.

22567 **APPLICATION USAGE**

22568 If the number is not known to be in range, *strtol()* or *strtoll()* should be used because *atol()* and  
22569 *atoll()* are not required to perform any error checking.

22570 **RATIONALE**

22571 None.

22572 **FUTURE DIRECTIONS**

22573 None.

22574 **SEE ALSO**22575 [strtol\(\)](#)22576 XBD [<stdlib.h>](#)22577 **CHANGE HISTORY**

22578 First released in Issue 1. Derived from Issue 1 of the SVID.

22579 **Issue 6**22580 The *atoll()* function is added for alignment with the ISO/IEC 9899:1999 standard.22581 **Issue 7**22582 SD5-XSH-ERN-61 is applied, correcting the DESCRIPTION of *atoll()*.

22583 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0046 [892] is applied.

22584 **NAME**

22585 atomic\_compare\_exchange\_strong, atomic\_compare\_exchange\_strong\_explicit,  
 22586 atomic\_compare\_exchange\_weak, atomic\_compare\_exchange\_weak\_explicit — atomically  
 22587 compare and exchange the values of two objects

22588 **SYNOPSIS**

```
22589 #include <stdatomic.h>
22590 _Bool atomic_compare_exchange_strong(volatile A *object,
22591     C *expected, C desired);
22592 _Bool atomic_compare_exchange_strong_explicit(volatile A *object,
22593     C *expected, C desired, memory_order success, memory_order failure);
22594 _Bool atomic_compare_exchange_weak(volatile A *object,
22595     C *expected, C desired);
22596 _Bool atomic_compare_exchange_weak_explicit(volatile A *object,
22597     C *expected, C desired, memory_order success, memory_order failure);
```

22598 **DESCRIPTION**

22599 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 22600 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22601 volume of POSIX.1-2024 defers to the ISO C standard.

22602 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
 22603 `<stdatomic.h>` header nor support these generic functions.

22604 The `atomic_compare_exchange_strong_explicit()` generic function shall atomically compare the  
 22605 contents of the memory pointed to by *object* for equality with that pointed to by *expected*, and if  
 22606 true, shall replace the contents of the memory pointed to by *object* with *desired*, and if false, shall  
 22607 update the contents of the memory pointed to by *expected* with that pointed to by *object*. This  
 22608 operation shall be an atomic read-modify-write operation (see XBD Section 4.15.1, on page 100).  
 22609 If the comparison is true, *memory* shall be affected according to the value of *success*, and if the  
 22610 comparison is false, *memory* shall be affected according to the value of *failure*. The application  
 22611 shall ensure that *failure* is not `memory_order_release` nor `memory_order_acq_rel`, and  
 22612 shall ensure that *failure* is no stronger than *success*.

22613 The `atomic_compare_exchange_strong()` generic function shall be equivalent to  
 22614 `atomic_compare_exchange_strong_explicit()` called with *success* and *failure* both set to  
 22615 `memory_order_seq_cst`.

22616 The `atomic_compare_exchange_weak_explicit()` generic function shall be equivalent to  
 22617 `atomic_compare_exchange_strong_explicit()`, except that the compare-and-exchange operation may  
 22618 fail spuriously. That is, even when the contents of *memory* referred to by *expected* and *object* are  
 22619 equal, it may return zero and store back to *expected* the same memory contents that were  
 22620 originally there.

22621 The `atomic_compare_exchange_weak()` generic function shall be equivalent to  
 22622 `atomic_compare_exchange_weak_explicit()` called with *success* and *failure* both set to  
 22623 `memory_order_seq_cst`.

22624 **RETURN VALUE**

22625 These generic functions shall return the result of the comparison.

22626 **ERRORS**

22627 No errors are defined.

22628 **EXAMPLES**

22629 None.

22630 **APPLICATION USAGE**

22631 A consequence of spurious failure is that nearly all uses of weak compare-and-exchange will be  
22632 in a loop. For example:

```
22633     exp = atomic_load(&cur);  
22634     do {  
22635         des = function(exp);  
22636     } while (!atomic_compare_exchange_weak(&cur, &exp, des));
```

22637 When a compare-and-exchange is in a loop, the weak version will yield better performance on  
22638 some platforms. When a weak compare-and-exchange would require a loop and a strong one  
22639 would not, the strong one is preferable.

22640 **RATIONALE**

22641 None.

22642 **FUTURE DIRECTIONS**

22643 None.

22644 **SEE ALSO**22645 [XBD Section 4.15.1](#), [<stdatomic.h>](#)22646 **CHANGE HISTORY**

22647 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22648 **NAME**

22649 atomic\_exchange, atomic\_exchange\_explicit — atomically exchange the value of an object

22650 **SYNOPSIS**

22651 #include &lt;stdatomic.h&gt;

22652 C atomic\_exchange(volatile A \*object, C desired);

22653 C atomic\_exchange\_explicit(volatile A \*object, C desired,

22654 memory\_order order);

22655 **DESCRIPTION**22656 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22657 conflict between the requirements described here and the ISO C standard is unintentional. This  
22658 volume of POSIX.1-2024 defers to the ISO C standard.22659 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
22660 `<stdatomic.h>` header nor support these generic functions.22661 The `atomic_exchange_explicit()` generic function shall atomically replace the value pointed to by  
22662 `object` with `desired`. This operation shall be an atomic read-modify-write operation (see XBD  
22663 [Section 4.15.1](#), on page 100). Memory shall be affected according to the value of `order`.22664 The `atomic_exchange()` generic function shall be equivalent to `atomic_exchange_explicit()` called  
22665 with `order` set to `memory_order_seq_cst`.22666 **RETURN VALUE**22667 These generic functions shall return the value pointed to by `object` immediately before the effects.22668 **ERRORS**

22669 No errors are defined.

22670 **EXAMPLES**

22671 None.

22672 **APPLICATION USAGE**

22673 None.

22674 **RATIONALE**

22675 None.

22676 **FUTURE DIRECTIONS**

22677 None.

22678 **SEE ALSO**22679 XBD [Section 4.15.1](#), `<stdatomic.h>`22680 **CHANGE HISTORY**

22681 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22682 **NAME**

22683 atomic\_fetch\_add, atomic\_fetch\_add\_explicit, atomic\_fetch\_and, atomic\_fetch\_and\_explicit,  
 22684 atomic\_fetch\_or, atomic\_fetch\_or\_explicit, atomic\_fetch\_sub, atomic\_fetch\_sub\_explicit,  
 22685 atomic\_fetch\_xor, atomic\_fetch\_xor\_explicit — atomically replace the value of an object with the  
 22686 result of a computation

22687 **SYNOPSIS**

```
22688 #include <stdatomic.h>
22689 C atomic_fetch_add(volatile A *object, M operand);
22690 C atomic_fetch_add_explicit(volatile A *object, M operand,
22691 memory_order order);
22692 C atomic_fetch_and(volatile A *object, M operand);
22693 C atomic_fetch_and_explicit(volatile A *object, M operand,
22694 memory_order order);
22695 C atomic_fetch_or(volatile A *object, M operand);
22696 C atomic_fetch_or_explicit(volatile A *object, M operand,
22697 memory_order order);
22698 C atomic_fetch_sub(volatile A *object, M operand);
22699 C atomic_fetch_sub_explicit(volatile A *object, M operand,
22700 memory_order order);
22701 C atomic_fetch_xor(volatile A *object, M operand);
22702 C atomic_fetch_xor_explicit(volatile A *object, M operand,
22703 memory_order order);
```

22704 **DESCRIPTION**

22705 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22706 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22707 volume of POSIX.1-2024 defers to the ISO C standard.

22708 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
 22709 `<stdatomic.h>` header nor support these generic functions.

22710 The `atomic_fetch_add_explicit()` generic function shall atomically replace the value pointed to by  
 22711 `object` with the result of adding `operand` to this value. This operation shall be an atomic read-  
 22712 modify-write operation (see XBD Section 4.15.1, on page 100). Memory shall be affected  
 22713 according to the value of `order`.

22714 The `atomic_fetch_add()` generic function shall be equivalent to `atomic_fetch_add_explicit()` called  
 22715 with `order` set to `memory_order_seq_cst`.

22716 The other `atomic_fetch_*`() generic functions shall be equivalent to `atomic_fetch_add_explicit()` if  
 22717 their name ends with `explicit`, or to `atomic_fetch_add()` if it does not, respectively, except that they  
 22718 perform the computation indicated in their name, instead of addition:

22719 *sub* subtraction

22720 *or* bitwise inclusive OR

22721 *xor* bitwise exclusive OR

22722 *and* bitwise AND

22723 For addition and subtraction, the application shall ensure that `A` is an atomic integer type or an  
 22724 atomic pointer type and is not `atomic_bool`. For the other operations, the application shall  
 22725 ensure that `A` is an atomic integer type and is not `atomic_bool`.

22726 For signed integer types, the computation shall silently wrap around on overflow; there are no  
 22727 undefined results. For pointer types, the result can be an undefined address, but the

22728 computations otherwise have no undefined behavior.

22729 **RETURN VALUE**

22730 These generic functions shall return the value pointed to by *object* immediately before the effects.

22731 **ERRORS**

22732 No errors are defined.

22733 **EXAMPLES**

22734 None.

22735 **APPLICATION USAGE**

22736 The operation of these generic functions is nearly equivalent to the operation of the  
22737 corresponding compound assignment operators +=, -=, etc. The only differences are that the  
22738 compound assignment operators are not guaranteed to operate atomically, and the value yielded  
22739 by a compound assignment operator is the updated value of the object, whereas the value  
22740 returned by these generic functions is the previous value of the atomic object.

22741 **RATIONALE**

22742 None.

22743 **FUTURE DIRECTIONS**

22744 None.

22745 **SEE ALSO**

22746 XBD Section 4.15.1, [<stdatomic.h>](#)

22747 **CHANGE HISTORY**

22748 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.



22749 **NAME**

22750 atomic\_flag\_clear, atomic\_flag\_clear\_explicit — clear an atomic flag

22751 **SYNOPSIS**

22752 #include &lt;stdatomic.h&gt;

22753 void atomic\_flag\_clear(volatile atomic\_flag \*object);

22754 void atomic\_flag\_clear\_explicit(volatile atomic\_flag \*object,

22755 memory\_order order);

22756 **DESCRIPTION**

22757 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22758 conflict between the requirements described here and the ISO C standard is unintentional. This  
22759 volume of POSIX.1-2024 defers to the ISO C standard.

22760 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
22761 `<stdatomic.h>` header nor support these generic functions.

22762 The `atomic_flag_clear_explicit()` function shall atomically place the atomic flag pointed to by *object*  
22763 into the clear state. Memory shall be affected according to the value of *order*, which the  
22764 application shall ensure is not `memory_order_acquire` nor `memory_order_acq_rel`.

22765 The `atomic_flag_clear()` function shall be equivalent to `atomic_flag_clear_explicit()` called with  
22766 *order* set to `memory_order_seq_cst`.

22767 **RETURN VALUE**

22768 These functions shall not return a value.

22769 **ERRORS**

22770 No errors are defined.

22771 **EXAMPLES**

22772 None.

22773 **APPLICATION USAGE**

22774 None.

22775 **RATIONALE**

22776 None.

22777 **FUTURE DIRECTIONS**

22778 None.

22779 **SEE ALSO**22780 XBD Section 4.15.1, `<stdatomic.h>`22781 **CHANGE HISTORY**

22782 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22783 **NAME**

22784 atomic\_flag\_test\_and\_set, atomic\_flag\_test\_and\_set\_explicit — test and set an atomic flag

22785 **SYNOPSIS**

```
22786 #include <stdatomic.h>
22787 _Bool atomic_flag_test_and_set(volatile atomic_flag *object);
22788 _Bool atomic_flag_test_and_set_explicit(volatile atomic_flag *object,
22789 memory_order order);
```

22790 **DESCRIPTION**

22791 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22792 conflict between the requirements described here and the ISO C standard is unintentional. This  
22793 volume of POSIX.1-2024 defers to the ISO C standard.

22794 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
22795 `<stdatomic.h>` header nor support these generic functions.

22796 The `atomic_flag_test_and_set_explicit()` function shall atomically place the atomic flag pointed to  
22797 by `object` into the set state and return the value corresponding to the immediately preceding  
22798 state. This operation shall be an atomic read-modify-write operation (see [Section 4.15.1](#), on page  
22799 100). Memory shall be affected according to the value of `order`.

22800 The `atomic_flag_test_and_set()` function shall be equivalent to `atomic_flag_test_and_set_explicit()`  
22801 called with `order` set to `memory_order_seq_cst`.

22802 **RETURN VALUE**

22803 These functions shall return the value that corresponds to the state of the atomic flag  
22804 immediately before the effects. The return value true shall correspond to the set state and the  
22805 return value false shall correspond to the clear state.

22806 **ERRORS**

22807 No errors are defined.

22808 **EXAMPLES**

22809 None.

22810 **APPLICATION USAGE**

22811 None.

22812 **RATIONALE**

22813 None.

22814 **FUTURE DIRECTIONS**

22815 None.

22816 **SEE ALSO**22817 [XBD Section 4.15.1](#), `<stdatomic.h>`22818 **CHANGE HISTORY**

22819 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22820 **NAME**

22821 atomic\_init — initialize an atomic object

22822 **SYNOPSIS**

22823 #include &lt;stdatomic.h&gt;

22824 void atomic\_init(volatile A \*obj, C value);

22825 **DESCRIPTION**

22826 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22827 conflict between the requirements described here and the ISO C standard is unintentional. This  
22828 volume of POSIX.1-2024 defers to the ISO C standard.

22829 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
22830 `<stdatomic.h>` header nor support these generic functions.

22831 The `atomic_init()` generic function shall initialize the atomic object pointed to by *obj* to the value  
22832 *value*, while also initializing any additional state that the implementation might need to carry for  
22833 the atomic object.

22834 Although this function initializes an atomic object, it does not avoid data races; concurrent  
22835 access to the variable being initialized, even via an atomic operation, constitutes a data race.

22836 **RETURN VALUE**22837 The `atomic_init()` generic function shall not return a value.22838 **ERRORS**

22839 No errors are defined.

22840 **EXAMPLES**

```
22841 atomic_int guide;  
22842 atomic_init(&guide, 42);
```

22843 **APPLICATION USAGE**

22844 None.

22845 **RATIONALE**

22846 None.

22847 **FUTURE DIRECTIONS**

22848 None.

22849 **SEE ALSO**22850 XBD [<stdatomic.h>](#)22851 **CHANGE HISTORY**

22852 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22853 **NAME**

22854 atomic\_is\_lock\_free — indicate whether or not atomic operations are lock-free

22855 **SYNOPSIS**

22856 #include &lt;stdatomic.h&gt;

22857 \_Bool atomic\_is\_lock\_free(const volatile A \*obj);

22858 **DESCRIPTION**

22859 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22860 conflict between the requirements described here and the ISO C standard is unintentional. This  
22861 volume of POSIX.1-2024 defers to the ISO C standard.

22862 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
22863 `<stdatomic.h>` header nor support these generic functions.

22864 The `atomic_is_lock_free()` generic function shall indicate whether or not atomic operations on  
22865 objects of the type pointed to by `obj` are lock-free; `obj` can be a null pointer.

22866 **RETURN VALUE**

22867 The `atomic_is_lock_free()` generic function shall return a non-zero value if and only if atomic  
22868 operations on objects of the type pointed to by `obj` are lock-free. During the lifetime of the calling  
22869 process, the result of the lock-free query shall be consistent for all pointers of the same type.

22870 **ERRORS**

22871 No errors are defined.

22872 **EXAMPLES**

22873 None.

22874 **APPLICATION USAGE**

22875 None.

22876 **RATIONALE**

22877 Operations that are lock-free should also be address-free. That is, atomic operations on the same  
22878 memory location via two different addresses will communicate atomically. The implementation  
22879 should not depend on any per-process state. This restriction enables communication via memory  
22880 mapped into a process more than once and memory shared between two processes.

22881 **FUTURE DIRECTIONS**

22882 None.

22883 **SEE ALSO**22884 XBD [<stdatomic.h>](#)22885 **CHANGE HISTORY**

22886 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22887 **NAME**

22888 atomic\_load, atomic\_load\_explicit — atomically obtain the value of an object

22889 **SYNOPSIS**

22890 #include &lt;stdatomic.h&gt;

22891 C atomic\_load(const volatile A \*object);

22892 C atomic\_load\_explicit(const volatile A \*object, memory\_order order);

22893 **DESCRIPTION**22894 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22895 conflict between the requirements described here and the ISO C standard is unintentional. This  
22896 volume of POSIX.1-2024 defers to the ISO C standard.22897 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
22898 `<stdatomic.h>` header nor support these generic functions.22899 The `atomic_load_explicit()` generic function shall atomically obtain the value pointed to by `object`.  
22900 Memory shall be affected according to the value of `order`, which the application shall ensure is  
22901 not `memory_order_release` nor `memory_order_acq_rel`.22902 The `atomic_load()` generic function shall be equivalent to `atomic_load_explicit()` called with `order`  
22903 set to `memory_order_seq_cst`.22904 **RETURN VALUE**22905 These generic functions shall return the value pointed to by `object`.22906 **ERRORS**

22907 No errors are defined.

22908 **EXAMPLES**

22909 None.

22910 **APPLICATION USAGE**

22911 None.

22912 **RATIONALE**

22913 None.

22914 **FUTURE DIRECTIONS**

22915 None.

22916 **SEE ALSO**22917 XBD Section 4.15.1, `<stdatomic.h>`22918 **CHANGE HISTORY**

22919 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22920 **NAME**

22921 atomic\_signal\_fence, atomic\_thread\_fence — fence operations

22922 **SYNOPSIS**

22923 #include &lt;stdatomic.h&gt;

22924 void atomic\_signal\_fence(memory\_order order);

22925 void atomic\_thread\_fence(memory\_order order);

22926 **DESCRIPTION**

22927 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 22928 conflict between the requirements described here and the ISO C standard is unintentional. This  
 22929 volume of POSIX.1-2024 defers to the ISO C standard.

22930 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
 22931 `<stdatomic.h>` header nor support these generic functions.

22932 The `atomic_signal_fence()` and `atomic_thread_fence()` functions provide synchronization primitives  
 22933 called *fences*. Fences can have acquire semantics, release semantics, or both. A fence with acquire  
 22934 semantics is called an *acquire fence*; a fence with release semantics is called a *release fence*.

22935 A release fence *A* synchronizes with an acquire fence *B* if there exist atomic operations *X* and *Y*,  
 22936 both operating on some atomic object *M*, such that *A* is sequenced before *X*, *X* modifies *M*, *Y* is  
 22937 sequenced before *B*, and *Y* reads the value written by *X* or a value written by any side effect in  
 22938 the hypothetical release sequence *X* would head if it were a release operation.

22939 A release fence *A* synchronizes with an atomic operation *B* that performs an acquire operation  
 22940 on an atomic object *M* if there exists an atomic operation *X* such that *A* is sequenced before *X*, *X*  
 22941 modifies *M*, and *B* reads the value written by *X* or a value written by any side effect in the  
 22942 hypothetical release sequence *X* would head if it were a release operation.

22943 An atomic operation *A* that is a release operation on an atomic object *M* synchronizes with an  
 22944 acquire fence *B* if there exists some atomic operation *X* on *M* such that *X* is sequenced before *B*  
 22945 and reads the value written by *A* or a value written by any side effect in the release sequence  
 22946 headed by *A*.

22947 Depending on the value of *order*, the operation performed by `atomic_thread_fence()` shall:

- 22948 • have no effects, if *order* is equal to `memory_order_relaxed`;
- 22949 • be an acquire fence, if *order* is equal to `memory_order_acquire` or  
 22950 `memory_order_consume`;
- 22951 • be a release fence, if *order* is equal to `memory_order_release`;
- 22952 • be both an acquire fence and a release fence, if *order* is equal to `memory_order_acq_rel`;
- 22953 • be a sequentially consistent acquire and release fence, if *order* is equal to  
 22954 `memory_order_seq_cst`.

22955 The `atomic_signal_fence()` function shall be equivalent to `atomic_thread_fence()`, except that the  
 22956 resulting ordering constraints shall be established only between a thread and a signal handler  
 22957 executed in the same thread.

22958 **RETURN VALUE**

22959 These functions shall not return a value.

**22960 ERRORS**

22961 No errors are defined.

**22962 EXAMPLES**

22963 None.

**22964 APPLICATION USAGE**

22965 The *atomic\_signal\_fence()* function can be used to specify the order in which actions performed  
22966 by the thread become visible to the signal handler. Implementation reorderings of loads and  
22967 stores are inhibited in the same way as with *atomic\_thread\_fence()*, but the hardware fence  
22968 instructions that *atomic\_thread\_fence()* would have inserted are not emitted.

**22969 RATIONALE**

22970 None.

**22971 FUTURE DIRECTIONS**

22972 None.

**22973 SEE ALSO**

22974 [XBD Section 4.15.1, <stdatomic.h>](#)

**22975 CHANGE HISTORY**

22976 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

22977 **NAME**

22978 atomic\_store, atomic\_store\_explicit — atomically store a value in an object

22979 **SYNOPSIS**

22980 #include <stdatomic.h>

22981 void atomic\_store(volatile A \*object, C desired);

22982 void atomic\_store\_explicit(volatile A \*object, C desired,

22983 memory\_order order);

22984 **DESCRIPTION**

22985 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
22986 conflict between the requirements described here and the ISO C standard is unintentional. This  
22987 volume of POSIX.1-2024 defers to the ISO C standard.

22988 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
22989 `<stdatomic.h>` header nor support these generic functions.

22990 The `atomic_store_explicit()` generic function shall atomically replace the value pointed to by *object*  
22991 with *desired*. Memory shall be affected according to the value of *order*, which the application  
22992 shall ensure is not `memory_order_acquire`, `memory_order_consume`, nor  
22993 `memory_order_acq_rel`.

22994 The `atomic_store()` generic function shall be equivalent to `atomic_store_explicit()` called with *order*  
22995 set to `memory_order_seq_cst`.

22996 **RETURN VALUE**

22997 These generic functions shall not return a value.

22998 **ERRORS**

22999 No errors are defined.

23000 **EXAMPLES**

23001 None.

23002 **APPLICATION USAGE**

23003 None.

23004 **RATIONALE**

23005 None.

23006 **FUTURE DIRECTIONS**

23007 None.

23008 **SEE ALSO**

23009 XBD Section 4.15.1, [<stdatomic.h>](#)

23010 **CHANGE HISTORY**

23011 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.



23012 **NAME**

23013            basename — return the last component of a pathname

23014 **SYNOPSIS**

```
23015 XSI        #include <libgen.h>
23016            char *basename(char *path);
```

23017 **DESCRIPTION**

23018            The *basename()* function shall take the pathname pointed to by *path* and return a pointer to the  
23019            final component of the pathname, deleting any trailing '/' characters.

23020            If the string pointed to by *path* consists entirely of the '/' character, *basename()* shall return a  
23021            pointer to the string "/", except that if the string pointed to by *path* is exactly "/", it is  
23022            implementation-defined whether "/" or "/" is returned.

23023            If *path* is a null pointer or points to an empty string, *basename()* shall return a pointer to the  
23024            string ".".

23025            The *basename()* function may modify the string pointed to by *path*, and may return a pointer into  
23026            the input string. The returned pointer might be invalidated if the input string is subsequently  
23027            modified or freed. If *path* is a null pointer or points to an empty string, or if the string pointed to  
23028            by *path* consists entirely of the '/' character, the returned pointer may point to constant data  
23029            that cannot be modified.

23030 **RETURN VALUE**

23031            The *basename()* function shall return a pointer to the final component of *path*.

23032            The *basename()* function shall always be successful and no return value is reserved to indicate an  
23033            error.

23034 **ERRORS**

23035            No errors are defined.

23036 **EXAMPLES**23037            **Using basename()**

23038            The following program fragment returns a pointer to the value *lib*, which is the base name of  
23039            */usr/lib*.

```
23040            #include <libgen.h>
23041            ...
23042            char name[] = "/usr/lib";
23043            char *base;
23044            base = basename(name);
23045            ...
```

Sample Input and Output Strings for the `basename()` and `dirname()` Functions and the `basename` and `dirname` Utilities

basename() and dirname() Functions path Argument	String Returned by basename()	String Returned by dirname()	basename and dirname Utilities string Operand	Output Written by basename Utility	Output Written by dirname Utility
"usr"	"usr"	". "	usr	usr	.
"usr/"	"usr"	". "	usr/	usr	.
" "	". "	". "	empty string	. or empty string	.
"/"	"/"	"/"	/	/	/
"/"	"/" or "/" (see note 1)	"/" or "/" (see note 1)	//	/ or // (see note 1)	/ or // (see note 1)
"/"	"/"	"/" or "/" (see note 1)	///	/	/ or ///
"/usr/"	"usr"	"/"	/usr/	usr	/
"/usr/lib"	"lib"	"/usr"	/usr/lib	lib	/usr
"/usr/lib/"	"lib"	"/usr" or "/usr" (see note 1)	///usr///lib//	lib	///usr or /usr (see note 1)
"/home//dwc//test"	"test"	"/home//dwc" or "/home/dwc"	/home//dwc//test	test	/home//dwc or /home/dwc
"/home/.../test"	"test"	"/home/.../" or "/home/.."	/home/.../test	test	/home/... or /home/..
"/home/dwc/."	". "	"/home/dwc"	/home/dwc/.	.	/home/dwc

**Note**

- Whether leading // can be converted to / depends on the implementation-defined behavior of // (see XBD Section 4.16 (on page 105); although the `basename()` and `dirname()` functions, and `basename` and `dirname` utilities, do not themselves perform pathname resolution, their results can be passed to a function or utility which does).

**APPLICATION USAGE**

Note that in some circumstances (in particular, when the returned string is required to be "/" or ". "), the returned pointer might point into constant data. Therefore, if the application needs to modify the returned data, it should be copied first.

**RATIONALE**

Earlier versions of this standard seemed to allow thread-safe and non-thread-safe implementations of `basename()` and `dirname()`, but did not allow implementations to return a null pointer and require that `errno` be set when that happened. The standard now requires thread-safe behavior for both of these functions and clearly states that they are always successful.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

[dirname\(\)](#)

XBD [<libgen.h>](#)

XCU [basename](#)

23090 **CHANGE HISTORY**

23091 First released in Issue 4, Version 2.

23092 **Issue 5**

23093 Moved from X/OPEN UNIX extension to BASE.

23094 Normative text previously in the APPLICATION USAGE section is moved to the  
23095 DESCRIPTION.

23096 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

23097 **Issue 6**

23098 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

23099 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/20 is applied, changing the  
23100 DESCRIPTION to make it clear that the string referenced is the string pointed to by *path*.

23101 **Issue 7**

23102 Austin Group Interpretation 1003.1-2001 #156 is applied.

23103 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0041 [75] is applied.

23104 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0047 [656], XSH/TC2-2008/0048 [928],  
23105 and XSH/TC2-2008/0049 [612] are applied.

23106 **Issue 8**

23107 Austin Group Defects 1064 and 1358 are applied, requiring *basename()* to be thread-safe and  
23108 allowing it to return a pointer to constant data under certain conditions.

23109 Austin Group Defect 1073 is applied, changing the EXAMPLES section.

23110 Austin Group Defect 1396 is applied, changing the EXAMPLES section.

23111 **NAME**

23112           be16toh, be32toh, be64toh, htobe16, htobe32, htobe64, htole16, htole32, htole64, le16toh, le32toh,  
23113           le64toh — convert values between host and specified byte order

23114 **SYNOPSIS**

```
23115       #include <endian.h>

23116       uint16_t be16toh(uint16_t big_endian_16bits);
23117       uint32_t be32toh(uint32_t big_endian_32bits);
23118       uint64_t be64toh(uint64_t big_endian_64bits);

23119       uint16_t htobe16(uint16_t host_16bits);
23120       uint32_t htobe32(uint32_t host_32bits);
23121       uint64_t htobe64(uint64_t host_64bits);

23122       uint16_t htole16(uint16_t host_16bits);
23123       uint32_t htole32(uint32_t host_32bits);
23124       uint64_t htole64(uint64_t host_64bits);

23125       uint16_t le16toh(uint16_t little_endian_16bits);
23126       uint32_t le32toh(uint32_t little_endian_32bits);
23127       uint64_t le64toh(uint64_t little_endian_64bits);
```

23128 **DESCRIPTION**

23129           These functions shall convert integer values of various sizes between host representations and  
23130           representations in a specified order.

23131           On some implementations, these functions are defined as macros.

23132           A little-endian representation of an integer has the least significant byte stored as the first byte,  
23133           with the significance of the bytes increasing as the byte address increases. A big-endian  
23134           representation has the most significant byte as the first byte, with the significance of the bytes  
23135           reducing as the byte address increases.

23136           **Note:**     Network byte order is big-endian.

23137           For example, the **uint32\_t** value 0x01020304 is represented as the four bytes 0x04, 0x03, 0x02,  
23138           0x01 on a little-endian host, and as 0x01, 0x02, 0x03, 0x04 on a big-endian host.

23139           For each of the sizes 16, 32 and 64, the *htobeSIZE()* function shall convert from whatever order  
23140           the host uses to big-endian representation, *htoleSIZE()* shall convert to little-endian  
23141           representation, *beSIZEtoh()* shall convert from big-endian to host order, and *leSIZEtoh()* shall  
23142           convert from little-endian to host order.

23143 **RETURN VALUE**

23144           These functions shall return an unsigned integer of the appropriate size and representation.

23145 **ERRORS**

23146           No errors are defined.

23147 **EXAMPLES**

```
23148       #include <endian.h>
23149       #include <stdio.h>
23150       #include <stdlib.h>

23151       int main(int argc, char *argv[])
23152       {
23153           uint32_t val;
23154           if (argc > 1) {
```

```

23155     val = (uint32_t)strtoul(argv[1], NULL, 0);
23156     printf("Value: %08x\n", val);

23157     printf("As bytes:\n");
23158     union {
23159         uint32_t asint;
23160         unsigned char asbytes[sizeof(uint32_t)];
23161     } u;
23162     printf("Little endian: ");
23163     u.asint = htobe32(val);
23164     for (int i = 0; i < sizeof(uint32_t); i++) {
23165         printf("%02x ", u.asbytes[i]);
23166     }
23167     printf("\n");

23168     printf("Big endian   : ");
23169     u.asint = htobe32(val);
23170     for (int i = 0; i < sizeof(uint32_t); i++) {
23171         printf("%02x ", u.asbytes[i]);
23172     }
23173     printf("\n");
23174 }
23175 return 0;
23176 }

```

**APPLICATION USAGE**

Since network order is defined as big-endian, the following functions are equivalent if `<arpa/inet.h>` is included:

23180  
23181  
23182  
23183  
23184

<code>&lt;endian.h&gt;</code>	<code>&lt;arpa/inet.h&gt;</code>
<i>htobe32</i>	<i>htonl</i>
<i>htobe16</i>	<i>htons</i>
<i>be32toh</i>	<i>ntohl</i>
<i>be16toh</i>	<i>ntohs</i>

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

[\*htonl\(\)\*](#)

XBD [`<arpa/inet.h>`](#), [`<endian.h>`](#)

**CHANGE HISTORY**

First released in Issue 8.

23194 **NAME**

23195 bind — bind a name to a socket

23196 **SYNOPSIS**

```
23197 #include <sys/socket.h>
23198 int bind(int socket, const struct sockaddr *address,
23199         socklen_t address_len);
```

23200 **DESCRIPTION**

23201 The *bind()* function shall assign a local socket address *address* to a socket identified by descriptor  
 23202 *socket* that has no local socket address assigned. Sockets created with the *socket()* function are  
 23203 initially unnamed; they are identified only by their address family.

23204 The *bind()* function takes the following arguments:

23205	<i>socket</i>	Specifies the file descriptor of the socket to be bound.
23206	<i>address</i>	Points to a <b>sockaddr</b> structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.
23207		
23208		
23209	<i>address_len</i>	Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>address</i> argument.
23210		

23211 The socket specified by *socket* may require the process to have appropriate privileges to use the  
 23212 *bind()* function.

23213 If the address family of the socket is AF\_UNIX, the application shall ensure that a null  
 23214 terminator after the pathname is included in the *sun\_path* member of *address* as a **sockaddr\_un**  
 23215 structure, and that *address\_len* is at least `offsetof(struct sockaddr_un, sun_path) +`  
 23216 `1` plus the length of the pathname. If the pathname in the *sun\_path* member of *address* names an  
 23217 existing file, including a symbolic link, *bind()* shall treat the address as already in use; see  
 23218 ERRORS below.

23219 If the socket address cannot be assigned immediately and O\_NONBLOCK is set for the file  
 23220 descriptor for the socket, *bind()* shall fail and set *errno* to [EINPROGRESS], but the assignment  
 23221 request shall not be aborted, and the assignment shall be completed asynchronously. Subsequent  
 23222 calls to *bind()* for the same socket, before the assignment is completed, shall fail and set *errno* to  
 23223 [EALREADY].

23224 When the assignment has been performed asynchronously, *pselect()*, *select()*, *poll()*, and *ppoll()*  
 23225 shall indicate that the file descriptor for the socket is ready for reading and writing.

23226 **RETURN VALUE**

23227 Upon successful completion, *bind()* shall return 0; otherwise, `-1` shall be returned and *errno* set  
 23228 to indicate the error.

23229 **ERRORS**

23230 The *bind()* function shall fail if:

23231 [EADDRINUSE] The specified address is already in use.

23232 [EADDRNOTAVAIL]

23233 The specified address is not available from the local machine.

23234 [EAFNOSUPPORT]

23235 The specified address is not a valid address for the address family of the  
 23236 specified socket.

23237	[EALREADY]	An assignment request is already in progress for the specified socket.
23238	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
23239	[EINPROGRESS]	O_NONBLOCK is set for the file descriptor for the socket and the assignment cannot be immediately performed; the assignment shall be performed asynchronously.
23240		
23241		
23242	[EINVAL]	The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.
23243		
23244	[ENOBUFS]	Insufficient resources were available to complete the call.
23245	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
23246	[EOPNOTSUPP]	The socket type of the specified socket does not support binding to an address.
23247		If the address family of the socket is AF_UNIX, then <i>bind()</i> shall fail if:
23248	[EACCES]	A component of the path prefix denies search permission, or the requested name requires writing in a directory with a mode that denies write permission.
23249		
23250		
23251	[EDESTADDRREQ] or [EISDIR]	
23252		The <i>address</i> argument is a null pointer.
23253	[EILSEQ]	The last pathname component is not a portable filename, and cannot be created in the target directory.
23254		
23255	[EIO]	An I/O error occurred.
23256	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname in <i>address</i> .
23257		
23258	[ENAMETOOLONG]	
23259		The length of a component of a pathname is longer than {NAME_MAX}.
23260	[ENOENT]	A component of the path prefix of the pathname in <i>address</i> does not name an existing file or the pathname is an empty string.
23261		
23262	[ENOENT] or [ENOTDIR]	
23263		The pathname in <i>address</i> contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters. If the pathname without the trailing <code>&lt;slash&gt;</code> characters would name an existing file, an [ENOENT] error shall not occur.
23264		
23265		
23266		
23267	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname in <i>address</i> contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
23268		
23269		
23270		
23271		
23272		
23273	[EROFS]	The name would reside on a read-only file system.
23274		The <i>bind()</i> function may fail if:
23275	[EACCES]	The specified address is protected and the current user does not have permission to bind to it.
23276		

23277	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family.
23278	[EISCONN]	The socket is already connected.
23279	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in <i>address</i> .
23280		
23281	[ENAMETOOLONG]	
23282		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
23283		
23284		

### 23285 EXAMPLES

23286 The following code segment shows how to create a socket and bind it to a name in the AF\_UNIX domain.

```

23288 #define MY_SOCKET_PATH "/somepath"
23289
23289 int sfd;
23290 struct sockaddr_un my_addr;
23291
23291 sfd = socket(AF_UNIX, SOCK_STREAM, 0);
23292 if (sfd == -1)
23293     /* Handle error */;
23294
23294 memset(&my_addr, '\0', sizeof(struct sockaddr_un));
23295                               /* Clear structure */
23296 my_addr.sun_family = AF_UNIX;
23297 strncpy(my_addr.sun_path, MY_SOCKET_PATH, sizeof(my_addr.sun_path) - 1);
23298
23298 if (bind(sfd, (struct sockaddr *) &my_addr,
23299         sizeof(struct sockaddr_un)) == -1)
23300     /* Handle error */;

```

### 23301 APPLICATION USAGE

23302 An application program can retrieve the assigned socket name with the *getsockname()* function.

23303 For AF\_UNIX sockets, some implementations support an extension where *address\_len* does not have to include a null terminator for the pathname stored in *sun\_path*, which in turn allows a pathname to be one byte longer. However, such usage is not portable, and carries a risk of accessing beyond the intended bounds of the pathname length.

### 23307 RATIONALE

23308 Implementations are encouraged to have *bind()* report an [EILSEQ] error if the last component of the address to be bound to an AF\_UNIX family socket contains any bytes that have the encoded value of a <newline> character.

### 23311 FUTURE DIRECTIONS

23312 None.

### 23313 SEE ALSO

23314 [connect\(\)](#), [getsockname\(\)](#), [listen\(\)](#), [socket\(\)](#)

23315 XBD [<sys/socket.h>](#)

### 23316 CHANGE HISTORY

23317 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



23318 **Issue 7**

23319 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the ``may fail'' [ENOBUFS]  
23320 error to become a ``shall fail'' error.

23321 Austin Group Interpretation 1003.1-2001 #143 is applied.

23322 SD5-XSH-ERN-185 is applied.

23323 An example is added.

23324 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0042 [146], XSH/TC1-2008/0043 [146],  
23325 and XSH/TC1-2008/0044 [324] are applied.

23326 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0050 [822] is applied.

23327 **Issue 8**

23328 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
23329 filenames containing any bytes that have the encoded value of a <newline> character.

23330 Austin Group Defects 293 and 1482 are applied, adding the [EILSEQ] error.

23331 Austin Group Defect 561 is applied, changing the requirements for the *sun\_path* member of the  
23332 **sockaddr\_un** structure.

23333 Austin Group Defect 1263 is applied, adding *ppoll()*.

23334 Austin Group Defect 1605 is applied, clarifying how the [EADDRINUSE] error applies to  
23335 AF\_UNIX sockets.

23336 **NAME**

23337 bindtextdomain, bind\_textdomain\_codeset, textdomain — text domain manipulation functions

23338 **SYNOPSIS**

```
23339 #include <libintl.h>
23340 char *bindtextdomain(const char *domainname, const char *dirname);
23341 char *bind_textdomain_codeset(const char *domainname,
23342     const char *codeset);
23343 char *textdomain(const char *domainname);
```

23344 **DESCRIPTION**

23345 The *textdomain()* function shall set or query the name of the current text domain of the calling  
 23346 process. The application shall ensure that the *domainname* argument is either a null pointer  
 23347 (when querying), an empty string, or a string that, when used by the *gettext* family of functions  
 23348 to construct a pathname to a messages object, results in a valid pathname. For portable  
 23349 applications, it should only contain characters from the portable filename character set.

23350 The text domain setting made by the last successful call to *textdomain()* shall remain in effect  
 23351 across subsequent calls to *setlocale()*, *uselocale()*, and the *gettext* family of functions.

23352 Applications should not use text domains whose names begin with the strings "SYS\_" or  
 23353 "libc". These prefixes are reserved for implementation use.

23354 The current setting of the text domain can be queried without affecting the current state of the  
 23355 domain by calling *textdomain()* with *domainname* set to a null pointer. Calling *textdomain()* with a  
 23356 *domainname* argument of an empty string shall set the text domain to the default domain,  
 23357 "messages".

23358 The *bindtextdomain()* function shall set or query the binding of a text domain to a *dirname* that is  
 23359 used by the *gettext* family of functions to construct a pathname to a messages object in the text  
 23360 domain:

- 23361 • If *domainname* is a null pointer or an empty string, *bindtextdomain()* shall make no changes  
 23362 and return a null pointer without changing *errno*.
- 23363 • Otherwise, if *dirname* is a non-empty string:
  - 23364 — If *domainname* is not already bound, *bindtextdomain()* shall bind the text domain  
 23365 specified by *domainname* to the pathname pointed to by *dirname* and return the bound  
 23366 directory pathname on success or a null pointer on failure.
  - 23367 — If *domainname* is already bound, *bindtextdomain()* shall replace the existing binding  
 23368 with the pathname pointed to by *dirname* and return the bound directory pathname  
 23369 on success or a null pointer on failure. On failure, the existing binding shall remain  
 23370 unchanged.

23371 It is unspecified whether the *bindtextdomain()* function performs pathname resolution on  
 23372 *dirname*, or whether that is done by the *gettext* family of functions.

- 23373 • Otherwise, if *dirname* is a null pointer:
  - 23374 — If *domainname* is bound, the function shall return the bound directory pathname.
  - 23375 — If *domainname* is not bound, the function shall return the implementation-defined  
 23376 default directory pathname used by the *gettext* family of functions.
- 23377 • Otherwise, *dirname* is an empty string and the behavior is unspecified.

23378 If a text domain is bound to a relative pathname and the current working directory is changed  
 23379 after the binding is established, the pathnames used by the *gettext* family of functions to locate

23380 messages objects for that text domain are unspecified.

23381 The *bind\_textdomain\_codeset()* function shall set or query the binding of a text domain to the  
23382 output codeset used by the *gettext* family of functions for message strings retrieved from  
23383 messages objects for the text domain specified by *domainname*:

- 23384 • If *domainname* is a null pointer or an empty string, *bind\_textdomain\_codeset()* shall make no  
23385 changes and return a null pointer without changing *errno*.
- 23386 • Otherwise, if *codeset* is a non-empty string:
  - 23387 — If *domainname* is not already bound, *bind\_textdomain\_codeset()* shall bind the text  
23388 domain specified by *domainname* to the codeset pointed to by *codeset* and return the  
23389 newly bound codeset on success or a null pointer on failure.
  - 23390 — If *domainname* is already bound, *bind\_textdomain\_codeset()* shall replace the existing  
23391 binding with the codeset pointed to by *codeset* and return the newly bound codeset  
23392 on success or a null pointer on failure. On failure, the existing binding shall remain  
23393 unchanged.

23394 The application shall ensure that the *codeset* argument, if non-empty, is a valid codeset  
23395 name that can be used as the *tocode* argument of the *iconv\_open()* function, and that in the  
23396 codeset it specifies, the <NUL> character corresponds to a single null byte.

- 23397 • Otherwise, if *codeset* is a null pointer:
  - 23398 — If *domainname* is bound, the function shall return the bound codeset.
  - 23399 — If *domainname* is not bound, the function shall return the implementation-defined  
23400 default codeset used by the *gettext* family of functions.
- 23401 • Otherwise, *codeset* is an empty string and the behavior is unspecified.

23402 If *codeset* is a null pointer and *domainname* is a non-empty string, *bind\_textdomain\_codeset()* shall  
23403 return the current codeset for the named domain, or a null pointer if a codeset has not yet been  
23404 set. The *bind\_textdomain\_codeset()* function can be called multiple times. If successfully called  
23405 multiple times with the same *domainname* argument, the last such call shall override the setting  
23406 made by the previous such call.

#### 23407 RETURN VALUE

23408 The return value from a successful *textdomain()* call shall be a pointer to a string containing the  
23409 current setting of the text domain. If *domainname* is a null pointer, *textdomain()* shall return a  
23410 pointer to the string containing the current text domain. If *textdomain()* was not previously  
23411 called and *domainname* is a null string, the name of the default text domain shall be returned.  
23412 The name of the default text domain shall be the string "messages". If *textdomain()* fails, a null  
23413 pointer shall be returned and *errno* shall be set to indicate the error.

23414 For *bindtextdomain()* return values see the DESCRIPTION. When *bindtextdomain()* is called with  
23415 a non-empty *domainname* and returns a null pointer, it shall set *errno* to indicate the error. When  
23416 *bindtextdomain()* returns a pathname for a bound text domain, the return value shall be a pointer  
23417 to a copy of the *dirname* string passed to the *bindtextdomain()* call that created the binding. The  
23418 returned string shall remain valid until the next successful call to *bindtextdomain()* with a non-  
23419 empty *dirname* and same *domainname*. The application shall ensure that it does not modify the  
23420 returned string.

23421 A call to the *bind\_textdomain\_codeset()* function with a non-empty *domainname* argument shall  
23422 return one of the following:

- 23423 • The currently bound codeset name for that text domain if *codeset* is a null pointer
- 23424 • The newly bound codeset if *codeset* is non-empty
- 23425 • A null pointer without changing *errno* if no codeset has yet been bound for that text
- 23426 domain

23427 The application shall ensure that it does not modify the returned string. A subsequent call to  
 23428 *bind\_textdomain\_codeset()* with a non-empty *domainname* argument might invalidate the returned  
 23429 pointer or overwrite the string content. The returned pointer might also be invalidated if the  
 23430 calling thread is terminated. If *bind\_textdomain\_codeset()* fails, a null pointer shall be returned  
 23431 and *errno* shall be set to indicate the error.

#### 23432 ERRORS

23433 For the conditions under which *bindtextdomain()*—if it performs pathname resolution—fails and  
 23434 may fail, refer to *open()*.

23435 In addition, the *textdomain()*, *bindtextdomain()*, and *bind\_textdomain\_codeset()* functions may fail  
 23436 if:

23437 [ENOMEM]      Insufficient memory available.

#### 23438 EXAMPLES

23439 See the examples for *gettext*.

#### 23440 APPLICATION USAGE

23441 A text *domainname* is limited to {TEXTDOMAIN\_MAX} bytes.

23442 Application developers are responsible for ensuring that the text domain used is not used by  
 23443 other applications. To minimize the chances of collision, developers can prefix text domains with  
 23444 their company or application name (or both) and an underscore. For example, if your  
 23445 application name was "foo" and you wanted to use the text domain "errors", you could  
 23446 instead use the text domain "foo\_errors". Note that if an application can be installed with a  
 23447 configurable name, a text domain prefix based on the application name should change with the  
 23448 application name.

23449 Specifying a relative pathname to the *bindtextdomain()* function should be avoided, since it may  
 23450 result in messages objects being searched for in a directory relative to the current working  
 23451 directory of the calling process; if the process calls the *chdir()* function, the directory searched for  
 23452 may also be changed.

23453 Since pathname resolution of *dirname* might not be performed by *bindtextdomain()*, but could be  
 23454 performed later by the *gettext* family of functions, and since the latter have no way to report an  
 23455 error, applications should verify, using for example *stat()*, that the directory is accessible if this is  
 23456 desired.

#### 23457 RATIONALE

23458 Although the return type of these functions ought to be **const char \***, it is **char \*** to match  
 23459 historical practice.

23460 Pathname resolution of the *dirname* argument passed to *bindtextdomain()* may be performed by  
 23461 *bindtextdomain()* itself or by the *gettext* family of functions. If pathname resolution fails in one of  
 23462 the *gettext* family of functions, it is neither allowed to modify *errno* nor to return an error, but if  
 23463 pathname resolution fails in *bindtextdomain()*, it is required to report an error and set *errno* just  
 23464 like *open()* does.

23465 Historically, *bindtextdomain()* did not perform pathname resolution. However, the standard  
 23466 developers decided to allow this as an option so that future implementations can, if desired,  
 23467 open a file descriptor for that directory in *bindtextdomain()* and then use that file descriptor with

23468 *openat()* in the *gettext* family of functions.

23469 The *dirname* parameter to *bindtextdomain()* may need to be copied to avoid the possibility of the  
23470 application releasing the memory used by the argument while the *gettext* family of functions  
23471 may still need to reference it.

23472 When *bindtextdomain()* is called with a non-empty *domainname* and an empty *dirname*, historical  
23473 implementations of the *gettext* family of functions use the empty string for the *dirname* part of  
23474 the messages object pathname, resulting in an absolute pathname of the form  
23475 */localename/categoryname/textdomainname.mo*. The standard developers did not believe this  
23476 behavior to be useful. Using the empty *dirname* case as a way to remove an existing binding  
23477 seemed to be a more useful behavior, and would be consistent with the behavior of *textdomain()*.  
23478 However, because no historical implementations behave this way, the behavior is left  
23479 unspecified.

23480 Some implementations set *errno* to [EAGAIN] to signal memory allocation failures that might  
23481 succeed if retried and [ENOMEM] for failures that are unlikely to ever succeed, for example due  
23482 to configured limits. Section 2.3 (on page 507) permits this behavior; when multiple error  
23483 conditions are simultaneously true there is no precedence between them.

#### 23484 FUTURE DIRECTIONS

23485 A future version of this standard may require implementations to prefix implementation-  
23486 provided text domains with either "SYS\_" or a prefix related to the implementor's company  
23487 name to avoid namespace collisions.

23488 A future version of this standard may require *bindtextdomain()* to remove any binding for  
23489 *domainname* when called with a non-empty *domainname* and an empty *dirname*.

#### 23490 SEE ALSO

23491 *gettext*, *iconv\_open()*, *setlocale()*, *uselocale()*

23492 XBD <libintl.h>, <limits.h>

23493 XCU *msgfmt*, *xgettext*

#### 23494 CHANGE HISTORY

23495 First released in Issue 8.



```

23540     char *string;
23541     int length;
23542 };
23543 struct node table[TABSIZE];    /* Table to be searched. */
23544     .
23545     .
23546     .
23547 {
23548     struct node *node_ptr, node;
23549     /* Routine to compare 2 nodes. */
23550     int node_compare(const void *, const void *);
23551     .
23552     .
23553     .
23554     while (scanf("%ms", &node.string) != EOF) {
23555         node_ptr = (struct node *)bsearch((void *)(&node),
23556             (void *)table, TABSIZE,
23557             sizeof(struct node), node_compare);
23558         if (node_ptr != NULL) {
23559             (void)printf("string = %20s, length = %d\n",
23560                 node_ptr->string, node_ptr->length);
23561         } else {
23562             (void)printf("not found: %s\n", node.string);
23563         }
23564         free(node.string);
23565     }
23566 }
23567 /*
23568     This routine compares two nodes based on an
23569     alphabetical ordering of the string field.
23570 */
23571 int
23572 node_compare(const void *node1, const void *node2)
23573 {
23574     return strcoll(((const struct node *)node1)->string,
23575         ((const struct node *)node2)->string);
23576 }

```

### 23577 APPLICATION USAGE

23578 The pointers to the key and the element at the base of the table should be of type pointer-to-  
23579 element.

23580 The comparison function need not compare every byte, so arbitrary data may be contained in  
23581 the elements in addition to the values being compared.

23582 In practice, the array is usually sorted according to the comparison function.

### 23583 RATIONALE

23584 The requirement that the second argument (hereafter referred to as *p*) to the comparison function  
23585 is a pointer to an element of the array implies that for every call all of the following expressions  
23586 are non-zero:

```

23587     ( (char *)p - (char *)base ) % width == 0
23588     (char *)p >= (char *)base

```

23589 (char \*)p < (char \*)base + nel \* width

23590 **FUTURE DIRECTIONS**

23591 None.

23592 **SEE ALSO**

23593 *hcreate(), lsearch(), qsort(), tdelete()*

23594 XBD <stdlib.h>

23595 **CHANGE HISTORY**

23596 First released in Issue 1. Derived from Issue 1 of the SVID.

23597 **Issue 6**

23598 The normative text is updated to avoid use of the term “must” for application requirements.

23599 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the  
23600 DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three  
23601 paragraphs. The RATIONALE section is also updated. These changes are for alignment with the  
23602 ISO C standard.

23603 **Issue 7**

23604 The EXAMPLES section is revised.

23605 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0051 [756] is applied.



23606 **NAME**

23607 btowc — single byte to wide character conversion

23608 **SYNOPSIS**

23609 #include &lt;stdio.h&gt;

23610 #include &lt;wchar.h&gt;

23611 wint\_t btowc(int c);

23612 **DESCRIPTION**

23613 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23614 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23615 volume of POSIX.1-2024 defers to the ISO C standard.

23616 The *btowc()* function shall determine whether *c* constitutes a valid (one-byte) character in the  
 23617 initial shift state.

23618 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

23619 **RETURN VALUE**

23620 The *btowc()* function shall return WEOF if *c* has the value EOF or if (**unsigned char**) *c* does not  
 23621 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the  
 23622 wide-character representation of that character.

23623 CX In the POSIX locale, *btowc()* shall not return WEOF if *c* has a value in the range 0 to 255  
 23624 inclusive.

23625 **ERRORS**

23626 No errors are defined.

23627 **EXAMPLES**

23628 None.

23629 **APPLICATION USAGE**

23630 None.

23631 **RATIONALE**

23632 None.

23633 **FUTURE DIRECTIONS**

23634 None.

23635 **SEE ALSO**23636 [wctob\(\)](#)23637 XBD [<stdio.h>](#), [<wchar.h>](#)23638 **CHANGE HISTORY**

23639 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 23640 (E).

23641 **Issue 7**

23642 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0052 [663] is applied.

23643 **NAME**

23644 c16rtomb, c32rtomb — convert a Unicode character code to a character (restartable)

23645 **SYNOPSIS**

23646 #include &lt;uchar.h&gt;

23647 size\_t c16rtomb(char \*restrict s, char16\_t c16, mbstate\_t \*restrict ps);

23648 size\_t c32rtomb(char \*restrict s, char32\_t c32, mbstate\_t \*restrict ps);

23649 **DESCRIPTION**

23650 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23651 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23652 volume of POSIX.1-2024 defers to the ISO C standard.

23653 If *s* is a null pointer, the *c16rtomb()* function shall be equivalent to the call:

23654 c16rtomb(buf, L'\0', ps)

23655 where *buf* is an internal buffer.

23656 If *s* is not a null pointer, the *c16rtomb()* function shall determine the number of bytes needed to  
 23657 represent the character that corresponds to the wide character given by *c16* (including any shift  
 23658 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At  
 23659 most {MB\_CUR\_MAX} bytes shall be stored. If *c16* is a null wide character, a null byte shall be  
 23660 stored, preceded by any shift sequence needed to restore the initial shift state; the resulting state  
 23661 described shall be the initial conversion state.

23662 If *ps* is a null pointer, the *c16rtomb()* function shall use its own internal **mbstate\_t** object, which  
 23663 shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t**  
 23664 object pointed to by *ps* shall be used to completely describe the current conversion state of the  
 23665 associated character sequence.

23666 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.

23667 The *mbrtoc16()* function shall not change the setting of *errno* if successful.

23668 The *c32rtomb()* function shall behave the same way as *c16rtomb()* except that the second  
 23669 parameter shall be an object of type **char32\_t** instead of **char16\_t**. References to *c16* in the above  
 23670 description shall apply as if they were *c32* when they are being read as describing *c32rtomb()*.

23671 If called with a null *ps* argument, the *c16rtomb()* function need not be thread-safe; however, such  
 23672 calls shall avoid data races with calls to *c16rtomb()* with a non-null argument and with calls to  
 23673 all other functions.

23674 If called with a null *ps* argument, the *c32rtomb()* function need not be thread-safe; however, such  
 23675 calls shall avoid data races with calls to *c32rtomb()* with a non-null argument and with calls to  
 23676 all other functions.

23677 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 23678 *c16rtomb()* or *c32rtomb()* with a null pointer for *ps*.

23679 **RETURN VALUE**

23680 These functions shall return the number of bytes stored in the array object (including any shift  
 23681 sequences). When *c16* or *c32* is not a valid wide character, an encoding error shall occur. In this  
 23682 case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size\_t**)−1;  
 23683 the conversion state is unspecified.

23684 **ERRORS**

23685 These functions shall fail if:

23686 [EILSEQ] An invalid wide-character code is detected.

23687 These functions may fail if:

23688 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.23689 **EXAMPLES**

23690 None.

23691 **APPLICATION USAGE**

23692 None.

23693 **RATIONALE**

23694 None.

23695 **FUTURE DIRECTIONS**

23696 None.

23697 **SEE ALSO**23698 [mbrtoc16\(\)](#)23699 XBD [<uchar.h>](#)23700 **CHANGE HISTORY**

23701 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

23702 **NAME**

23703 cabs, cabsf, cabsl — return a complex absolute value

23704 **SYNOPSIS**

```
23705 #include <complex.h>
23706 double cabs(double complex z);
23707 float cabsf(float complex z);
23708 long double cabsl(long double complex z);
```

23709 **DESCRIPTION**

23710 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23711 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23712 volume of POSIX.1-2024 defers to the ISO C standard.

23713 These functions shall compute the complex absolute value (also called norm, modulus, or  
 23714 magnitude) of  $z$ .

23715 **RETURN VALUE**

23716 These functions shall return the complex absolute value.

23717 MXC  $cabs(x + iy)$ ,  $cabs(y + ix)$ , and  $cabs(x - iy)$  shall return exactly the same value.23718 If  $z$  is  $\pm 0 \pm i0$ ,  $+0$  shall be returned.23719 If the real or imaginary part of  $z$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned, even if the other part is NaN.23720 If the real or imaginary part of  $z$  is NaN and the other part is not  $\pm\text{Inf}$ , NaN shall be returned.23721 **ERRORS**

23722 No errors are defined.

23723 **EXAMPLES**

23724 None.

23725 **APPLICATION USAGE**

23726 None.

23727 **RATIONALE**

23728 None.

23729 **FUTURE DIRECTIONS**

23730 None.

23731 **SEE ALSO**23732 XBD [<complex.h>](#)23733 **CHANGE HISTORY**

23734 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23735 **Issue 8**

23736 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
 23737 standard.

23738 **NAME**

23739           acos, acosf, acosl — complex arc cosine functions

23740 **SYNOPSIS**

```
23741       #include <complex.h>
23742       double complex acos(double complex z);
23743       float complex acosf(float complex z);
23744       long double complex acosl(long double complex z);
```

23745 **DESCRIPTION**

23746 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
23747       conflict between the requirements described here and the ISO C standard is unintentional. This  
23748       volume of POSIX.1-2024 defers to the ISO C standard.

23749       These functions shall compute the complex arc cosine of  $z$ , with branch cuts outside the interval  
23750        $[-1, +1]$  along the real axis.

23751 **RETURN VALUE**

23752       These functions shall return the complex arc cosine value, in the range of a strip mathematically  
23753       unbounded along the imaginary axis and in the interval  $[0, \pi]$  along the real axis.

23754 MXC      *acos(conj(z))*, *acosf(conjf(z))*, and *acosl(conjl(z))* shall return exactly the same value as  
23755      *conj(acos(z))*, *conjf(acosf(z))*, and *conjl(acosl(z))*, respectively, including for the special values of  
23756       $z$  below.

23757      If  $z$  is  $\pm 0 + i0$ ,  $\pi/2 - i0$  shall be returned.

23758      If  $z$  is  $\pm 0 + iNaN$ ,  $\pi/2 + iNaN$  shall be returned.

23759      If  $z$  is  $x + iInf$  where  $x$  is finite,  $\pi/2 - iInf$  shall be returned.

23760      If  $z$  is  $x + iNaN$  where  $x$  is non-zero and finite,  $NaN + iNaN$  shall be returned and the invalid  
23761      floating-point exception may be raised.

23762      If  $z$  is  $-Inf + iy$  where  $y$  is positive-signed and finite,  $\pi - iInf$  shall be returned.

23763      If  $z$  is  $+Inf + iy$  where  $y$  is positive-signed and finite,  $+0 - iInf$  shall be returned.

23764      If  $z$  is  $-Inf + iInf$ ,  $3\pi/4 - iInf$  shall be returned.

23765      If  $z$  is  $+Inf + iInf$ ,  $\pi/4 - iInf$  shall be returned.

23766      If  $z$  is  $\pm Inf + iNaN$ ,  $NaN \pm iInf$  shall be returned; the sign of the imaginary part of the result is  
23767      unspecified.

23768      If  $z$  is  $NaN + iy$  where  $y$  is finite,  $NaN + iNaN$  shall be returned and the invalid floating-point  
23769      exception may be raised.

23770      If  $z$  is  $NaN + iInf$ ,  $NaN - iInf$  shall be returned.

23771      If  $z$  is  $NaN + iNaN$ ,  $NaN - iNaN$  shall be returned.

23772 **ERRORS**

23773       No errors are defined.

23774 **EXAMPLES**

23775 None.

23776 **APPLICATION USAGE**

23777 None.

23778 **RATIONALE**

23779 None.

23780 **FUTURE DIRECTIONS**

23781 None.

23782 **SEE ALSO**23783 [ccos\(\)](#)23784 XBD [<complex.h>](#)23785 **CHANGE HISTORY**

23786 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23787 **Issue 8**23788 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
23789 standard.

23790 **NAME**

23791 cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

23792 **SYNOPSIS**

23793 #include &lt;complex.h&gt;

23794 double complex cacosh(double complex z);

23795 float complex cacoshf(float complex z);

23796 long double complex cacoshl(long double complex z);

23797 **DESCRIPTION**

23798 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23799 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23800 volume of POSIX.1-2024 defers to the ISO C standard.

23801 These functions shall compute the complex arc hyperbolic cosine of  $z$ , with a branch cut at  
 23802 values less than 1 along the real axis.

23803 **RETURN VALUE**

23804 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip  
 23805 of non-negative values along the real axis and in the interval  $[-i\pi, +i\pi]$  along the imaginary axis.

23806 MXC  $\text{cacosh}(\text{conj}(z))$ ,  $\text{cacoshf}(\text{conj}(z))$ , and  $\text{cacoshl}(\text{conj}(z))$  shall return exactly the same value as  
 23807  $\text{conj}(\text{cacosh}(z))$ ,  $\text{conj}(\text{cacoshf}(z))$ , and  $\text{conj}(\text{cacoshl}(z))$ , respectively, including for the special  
 23808 values of  $z$  below.

23809 If  $z$  is  $\pm 0 + i0$ ,  $+0 + i\pi/2$  shall be returned.23810 If  $z$  is  $x + i\text{Inf}$  where  $x$  is finite,  $+\text{Inf} + i\pi/2$  shall be returned.

23811 If  $z$  is  $0 + i\text{NaN}$ ,  $\text{NaN} \pm i\pi/2$  shall be returned; the sign of the imaginary part of the result is  
 23812 unspecified.

23813 If  $z$  is  $x + i\text{NaN}$  where  $x$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 23814 floating-point exception may be raised.

23815 If  $z$  is  $-\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $+\text{Inf} + i\pi$  shall be returned.23816 If  $z$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $+\text{Inf} + i0$  shall be returned.23817 If  $z$  is  $-\text{Inf} + i\text{Inf}$ ,  $+\text{Inf} + i3\pi/4$  shall be returned.23818 If  $z$  is  $+\text{Inf} + i\text{Inf}$ ,  $+\text{Inf} + i\pi/4$  shall be returned.23819 If  $z$  is  $\pm\text{Inf} + i\text{NaN}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.

23820 If  $z$  is  $\text{NaN} + iy$  where  $y$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
 23821 exception may be raised.

23822 If  $z$  is  $\text{NaN} + i\text{Inf}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.23823 If  $z$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.23824 **ERRORS**

23825 No errors are defined.

23826 **EXAMPLES**

23827 None.

23828 **APPLICATION USAGE**

23829 None.

23830 **RATIONALE**

23831 None.

23832 **FUTURE DIRECTIONS**

23833 None.

23834 **SEE ALSO**23835 [ccosh\(\)](#)23836 XBD [<complex.h>](#)23837 **CHANGE HISTORY**

23838 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23839 **Issue 8**23840 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
23841 standard.



23842 **NAME**

23843           cacosl — complex arc cosine functions

23844 **SYNOPSIS**

23845           #include &lt;complex.h&gt;

23846           long double complex cacosl(long double complex z);

23847 **DESCRIPTION**23848           Refer to *acos()*.

23849 **NAME**

23850 call\_once — dynamic package initialization

23851 **SYNOPSIS**

23852 #include &lt;threads.h&gt;

23853 void call\_once(once\_flag \*flag, void (\*init\_routine)(void));

23854 once\_flag flag = ONCE\_FLAG\_INIT;

23855 **DESCRIPTION**

23856 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 23857 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23858 volume of POSIX.1-2024 defers to the ISO C standard.

23859 The *call\_once()* function shall use the **once\_flag** pointed to by *flag* to ensure that *init\_routine* is  
 23860 called exactly once, the first time the *call\_once()* function is called with that value of *flag*.  
 23861 Completion of an effective call to the *call\_once()* function shall synchronize with all subsequent  
 23862 calls to the *call\_once()* function with the same value of *flag*.

23863 CX The *call\_once()* function is not a cancellation point. However, if *init\_routine* is a cancellation point  
 23864 and is canceled, the effect on *flag* shall be as if *call\_once()* was never called.

23865 If the call to *init\_routine* is terminated by a call to *longjmp()* or *siglongjmp()*, the behavior is  
 23866 undefined.

23867 The behavior of *call\_once()* is undefined if *flag* has automatic storage duration or is not  
 23868 initialized by ONCE\_FLAG\_INIT.

23869 The *call\_once()* function shall not be affected if the calling thread executes a signal handler  
 23870 during the call.

23871 **RETURN VALUE**23872 The *call\_once()* function shall not return a value.23873 **ERRORS**

23874 No errors are defined.

23875 **EXAMPLES**

23876 None.

23877 **APPLICATION USAGE**

23878 If *init\_routine* recursively calls *call\_once()* with the same *flag*, the recursive call will not call the  
 23879 specified *init\_routine*, and thus the specified *init\_routine* will not complete, and thus the recursive  
 23880 call to *call\_once()* will not return. Use of *longjmp()* or *siglongjmp()* within an *init\_routine* to jump  
 23881 to a point outside of *init\_routine* prevents *init\_routine* from returning.

23882 **RATIONALE**

23883 For dynamic library initialization in a multi-threaded process, if an initialization flag is used the  
 23884 flag needs to be protected against modification by multiple threads simultaneously calling into  
 23885 the library. This can be done by using a statically-initialized mutex. However, the better solution  
 23886 is to use *call\_once()* or *pthread\_once()* which are designed for exactly this purpose, for example:

23887 #include &lt;threads.h&gt;

23888 static once\_flag random\_is\_initialized = ONCE\_FLAG\_INIT;

23889 extern void initialize\_random(void);

23890 int random\_function()

23891 {

23892 call\_once(&amp;random\_is\_initialized, initialize\_random);

23893 ...

```
23894         /* Operations performed after initialization. */
23895     }
```

23896 The `call_once()` function is not affected by signal handlers for the reasons stated in XRAT [Section B.2.3](#) (on page 3742).  
23897

23898 **FUTURE DIRECTIONS**

23899 None.

23900 **SEE ALSO**

23901 [pthread\\_once\(\)](#)

23902 XBD [Section 4.15.2](#), [<threads.h>](#)

23903 **CHANGE HISTORY**

23904 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

23905 **NAME**

23906            calloc — a memory allocator

23907 **SYNOPSIS**

23908            #include &lt;stdlib.h&gt;

23909            void \*calloc(size\_t *nelem*, size\_t *elsize*);23910 **DESCRIPTION**

23911 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
 23912 conflict between the requirements described here and the ISO C standard is unintentional. This  
 23913 volume of POSIX.1-2024 defers to the ISO C standard.

23914        The `calloc()` function shall allocate unused space for an array of *nelem* elements each of whose  
 23915 size in bytes is *elsize*. The space shall be initialized to all bits 0.

23916        The order and contiguity of storage allocated by successive calls to `calloc()` is unspecified. The  
 23917 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
 23918 a pointer to any type of object with a fundamental alignment requirement and then used to  
 23919 access such an object or an array of such objects in the space allocated (until the space is  
 23920 explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint  
 23921 from any other object. The pointer returned shall point to the start (lowest byte address) of the  
 23922 allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the  
 23923 space requested is 0, the behavior is implementation-defined: either a null pointer shall be  
 23924 returned, or the behavior shall be as if the size were some non-zero value, except that the  
 23925 behavior is undefined if the returned pointer is used to access an object.

23926        For purposes of determining the existence of a data race, `calloc()` shall behave as though it  
 23927 accessed only memory locations accessible through its arguments and not other static duration  
 23928 storage. The function may, however, visibly modify the storage that it allocates. Calls to  
 23929 ADV `aligned_alloc()`, `calloc()`, `free()`, `malloc()`, `posix_memalign()`,  
 23930 CX `reallocarray()`, and `realloc()` that allocate or deallocate a particular region of memory shall occur  
 23931 in a single total order (see [Section 4.15.1](#), on page 100), and each such deallocation call shall  
 23932 synchronize with the next allocation (if any) in this order.

23933 **RETURN VALUE**

23934        Upon successful completion, `calloc()` shall return a pointer to the allocated space; if either *nelem*  
 23935 or *elsize* is 0, the application shall ensure that the pointer is not used to access an object.

23936 CX        Otherwise, it shall return a null pointer and set `errno` to indicate the error.

23937 **ERRORS**

23938        The `calloc()` function shall fail if:

23939 CX        [ENOMEM]        Insufficient memory is available, including the case when *nelem* \* *elsize* would  
 23940 overflow.

23941        The `calloc()` function may fail if:

23942 CX        [EINVAL]        *nelem* or *elsize* is 0 and the implementation does not support 0 sized  
 23943 allocations.

23944 **EXAMPLES**

23945 None.

23946 **APPLICATION USAGE**23947 There is now no requirement for the implementation to support the inclusion of `<malloc.h>`.23948 **RATIONALE**23949 See the RATIONALE for `malloc()`.23950 **FUTURE DIRECTIONS**

23951 None.

23952 **SEE ALSO**23953 `aligned_alloc()`, `free()`, `malloc()`, `realloc()`23954 XBD `<stdlib.h>`23955 **CHANGE HISTORY**

23956 First released in Issue 1. Derived from Issue 1 of the SVID.

23957 **Issue 6**

23958 Extensions beyond the ISO C standard are marked.

23959 The following new requirements on POSIX implementations derive from alignment with the  
23960 Single UNIX Specification:

- 23961
- The setting of `errno` and the [ENOMEM] error condition are mandatory if an insufficient  
23962 memory condition occurs.

23963 **Issue 7**

23964 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0053 [526] is applied.

23965 **Issue 8**23966 Austin Group Defect 374 is applied, changing the RETURN VALUE and ERRORS sections in  
23967 relation to 0 sized allocations.

23968 Austin Group Defect 1218 is applied, changing the [ENOMEM] error.

23969 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
23970 standard.

23971 Austin Group Defect 1387 is applied, changing the RATIONALE section.

23972 **NAME**

23973           carg, cargf, cargl — complex argument functions

23974 **SYNOPSIS**

```
23975       #include <complex.h>
23976       double carg(double complex z);
23977       float cargf(float complex z);
23978       long double cargl(long double complex z);
```

23979 **DESCRIPTION**

23980 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 23981           conflict between the requirements described here and the ISO C standard is unintentional. This  
 23982           volume of POSIX.1-2024 defers to the ISO C standard.

23983           These functions shall compute the argument (also called phase angle) of  $z$ , with a branch cut  
 23984           along the negative real axis.

23985 **RETURN VALUE**23986           These functions shall return the value of the argument in the interval  $[-\pi, +\pi]$ .23987 MXC       If  $z$  is  $-0 \pm i0$ ,  $\pm\pi$  shall be returned.23988           If  $z$  is  $+0 \pm i0$ ,  $\pm 0$  shall be returned.23989           If  $z$  is  $x \pm i0$  where  $x$  is negative,  $\pm\pi$  shall be returned.23990           If  $z$  is  $x \pm i0$  where  $x$  is positive,  $\pm 0$  shall be returned.23991           If  $z$  is  $\pm 0 + iy$  where  $y$  is negative,  $-\pi/2$  shall be returned.23992           If  $z$  is  $\pm 0 + iy$  where  $y$  is positive,  $\pi/2$  shall be returned.23993           If  $z$  is  $-\text{Inf} \pm iy$  where  $y$  is positive and finite,  $\pm\pi$  shall be returned.23994           If  $z$  is  $+\text{Inf} \pm iy$  where  $y$  is positive and finite,  $\pm 0$  shall be returned.23995           If  $z$  is  $x \pm i\text{Inf}$  where  $x$  is finite,  $\pm\pi/2$  shall be returned.23996           If  $z$  is  $-\text{Inf} \pm i\text{Inf}$ ,  $\pm 3\pi/4$  shall be returned.23997           If  $z$  is  $+\text{Inf} \pm i\text{Inf}$ ,  $\pm\pi/4$  shall be returned.23998           If the real or imaginary part of  $z$  is NaN, NaN shall be returned.23999 **ERRORS**

24000           No errors are defined.

24001 **EXAMPLES**

24002           None.

24003 **APPLICATION USAGE**

24004           None.

24005 **RATIONALE**

24006           None.

24007 **FUTURE DIRECTIONS**

24008           None.

24009 **SEE ALSO**24010 *cimag()*, *conj()*, *cproj()*

24011 XBD &lt;complex.h&gt;

24012 **CHANGE HISTORY**

24013 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24014 **Issue 8**24015 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24016 standard.

24017 **NAME**

24018 casin, casinf, casinl — complex arc sine functions

24019 **SYNOPSIS**

```
24020 #include <complex.h>
24021 double complex casin(double complex z);
24022 float complex casinf(float complex z);
24023 long double complex casinl(long double complex z);
```

24024 **DESCRIPTION**

24025 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24026 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24027 volume of POSIX.1-2024 defers to the ISO C standard.

24028 These functions shall compute the complex arc sine of  $z$ , with branch cuts outside the interval  
 24029  $[-1, +1]$  along the real axis.

24030 **RETURN VALUE**

24031 These functions shall return the complex arc sine value, in the range of a strip mathematically  
 24032 unbounded along the imaginary axis and in the interval  $[-\pi/2, +\pi/2]$  along the real axis.

24033 MXC *casin(conj(iz))*, *casinf(conj(iz))*, and *casinl(conjl(iz))* shall return exactly the same value as  
 24034 *conj(casin(iz))*, *conj(casinf(iz))*, and *conjl(casinl(iz))*, respectively, and *casin(-iz)*, *casinf(-iz)*, and  
 24035 *casinl(-iz)* shall return exactly the same value as  $-casin(iz)$ ,  $-casinf(iz)$ , and  $-casinl(iz)$ ,  
 24036 respectively, including for the special values of  $iz$  below.

24037 If  $iz$  is  $+0 + i0$ ,  $-i(0 + i0)$  shall be returned.

24038 If  $iz$  is  $x + i\text{Inf}$  where  $x$  is positive-signed and finite,  $-i(+\text{Inf} + i\pi/2)$  shall be returned.

24039 If  $iz$  is  $x + i\text{NaN}$  where  $x$  is finite,  $-i(\text{NaN} + i\text{NaN})$  shall be returned and the invalid floating-  
 24040 point exception may be raised.

24041 If  $iz$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $-i(+\text{Inf} + i0)$  shall be returned.

24042 If  $iz$  is  $+\text{Inf} + i\text{Inf}$ ,  $-i(+\text{Inf} + i\pi/4)$  shall be returned.

24043 If  $iz$  is  $+\text{Inf} + i\text{NaN}$ ,  $-i(+\text{Inf} + i\text{NaN})$  shall be returned.

24044 If  $iz$  is  $\text{NaN} + i0$ ,  $-i(\text{NaN} + i0)$  shall be returned.

24045 If  $iz$  is  $\text{NaN} + iy$  where  $y$  is non-zero and finite,  $-i(\text{NaN} + i\text{NaN})$  shall be returned and the  
 24046 invalid floating-point exception may be raised.

24047 If  $iz$  is  $\text{NaN} + i\text{Inf}$ ,  $-i(\pm\text{Inf} + i\text{NaN})$  shall be returned; the sign of the imaginary part of the result  
 24048 is unspecified.

24049 If  $iz$  is  $\text{NaN} + i\text{NaN}$ ,  $-i(\text{NaN} + i\text{NaN})$  shall be returned.

24050 **ERRORS**

24051 No errors are defined.



24052 **EXAMPLES**

24053 None.

24054 **APPLICATION USAGE**

24055 None.

24056 **RATIONALE**

24057 The MXC special cases for *casin()* are derived from those for *casinh()* by applying the formula  
24058  $casin(z) = -i casinh(iz)$ .

24059 **FUTURE DIRECTIONS**

24060 None.

24061 **SEE ALSO**24062 *casinh()*, *csin()*

24063 XBD &lt;complex.h&gt;

24064 **CHANGE HISTORY**

24065 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24066 **Issue 8**

24067 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24068 standard.

24069 **NAME**24070 `c sinh`, `c sinhf`, `c sinhl` — complex arc hyperbolic sine functions24071 **SYNOPSIS**24072 `#include <complex.h>`24073 `double complex c sinh(double complex z);`24074 `float complex c sinhf(float complex z);`24075 `long double complex c sinhl(long double complex z);`24076 **DESCRIPTION**24077 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
24078 conflict between the requirements described here and the ISO C standard is unintentional. This  
24079 volume of POSIX.1-2024 defers to the ISO C standard.24080 These functions shall compute the complex arc hyperbolic sine of  $z$ , with branch cuts outside the  
24081 interval  $[-i, +i]$  along the imaginary axis.24082 **RETURN VALUE**24083 These functions shall return the complex arc hyperbolic sine value, in the range of a strip  
24084 mathematically unbounded along the real axis and in the interval  $[-i\pi/2, +i\pi/2]$  along the  
24085 imaginary axis.24086 MXC `c sinh(conj(z))`, `c sinhf(conjf(z))`, and `c sinhl(conjl(z))` shall return exactly the same value as  
24087 `conj(c sinh(z))`, `conjf(c sinhf(z))`, and `conjl(c sinhl(z))`, respectively, and `c sinh(-z)`, `c sinhf(-z)`, and  
24088 `c sinhl(-z)` shall return exactly the same value as `-c sinh(z)`, `-c sinhf(z)`, and `-c sinhl(z)`,  
24089 respectively, including for the special values of  $z$  below.24090 If  $z$  is  $+0 + i0$ ,  $0 + i0$  shall be returned.24091 If  $z$  is  $x + i\text{Inf}$  where  $x$  is positive-signed and finite,  $+\text{Inf} + i\pi/2$  shall be returned.24092 If  $z$  is  $x + i\text{NaN}$  where  $x$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
24093 exception may be raised.24094 If  $z$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $+\text{Inf} + i0$  shall be returned.24095 If  $z$  is  $+\text{Inf} + i\text{Inf}$ ,  $+\text{Inf} + i\pi/4$  shall be returned.24096 If  $z$  is  $+\text{Inf} + i\text{NaN}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.24097 If  $z$  is  $\text{NaN} + i0$ ,  $\text{NaN} + i0$  shall be returned.24098 If  $z$  is  $\text{NaN} + iy$  where  $y$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
24099 floating-point exception may be raised.24100 If  $z$  is  $\text{NaN} + i\text{Inf}$ ,  $\pm\text{Inf} + i\text{NaN}$  shall be returned; the sign of the real part of the result is  
24101 unspecified.24102 If  $z$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.24103 **ERRORS**

24104 No errors are defined.

24105 **EXAMPLES**

24106 None.

24107 **APPLICATION USAGE**

24108 None.

24109 **RATIONALE**

24110 None.

24111 **FUTURE DIRECTIONS**

24112 None.

24113 **SEE ALSO**24114 [csinh\(\)](#)24115 XBD <[complex.h](#)>24116 **CHANGE HISTORY**

24117 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24118 **Issue 8**24119 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24120 standard.

24121 **NAME**24122            **casinl** — complex arc sine functions24123 **SYNOPSIS**

24124            #include &lt;complex.h&gt;

24125            long double complex casinl(long double complex z);

24126 **DESCRIPTION**24127            Refer to *casin()*.

24128 **NAME**

24129           catan, catanf, catanl — complex arc tangent functions

24130 **SYNOPSIS**

24131           #include &lt;complex.h&gt;

24132           double complex catan(double complex z);

24133           float complex catanf(float complex z);

24134           long double complex catanl(long double complex z);

24135 **DESCRIPTION**

24136 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 24137 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24138 volume of POSIX.1-2024 defers to the ISO C standard.

24139       These functions shall compute the complex arc tangent of  $z$ , with branch cuts outside the  
 24140 interval  $[-i, +i]$  along the imaginary axis.

24141 **RETURN VALUE**

24142       These functions shall return the complex arc tangent value, in the range of a strip  
 24143 mathematically unbounded along the imaginary axis and in the interval  $[-\pi/2, +\pi/2]$  along the  
 24144 real axis.

24145 MXC       $\text{catan}(\text{conj}(iz))$ ,  $\text{catanf}(\text{conj}(iz))$ , and  $\text{catanl}(\text{conj}(iz))$  shall return exactly the same value as  
 24146  $\text{conj}(\text{catan}(iz))$ ,  $\text{conj}(\text{catanf}(iz))$ , and  $\text{conj}(\text{catanl}(iz))$ , respectively, and  $\text{catan}(-iz)$ ,  $\text{catanf}(-iz)$ , and  
 24147  $\text{catanl}(-iz)$  shall return exactly the same value as  $-\text{catan}(iz)$ ,  $-\text{catanf}(iz)$ , and  $-\text{catanl}(iz)$ ,  
 24148 respectively, including for the special values of  $iz$  below.

24149       If  $iz$  is  $+0 + i0$ ,  $-i (+0 + i0)$  shall be returned.

24150       If  $iz$  is  $+0 + i\text{NaN}$ ,  $-i (+0 + i\text{NaN})$  shall be returned.

24151       If  $iz$  is  $+1 + i0$ ,  $-i (+\text{Inf} + i0)$  shall be returned and the divide-by-zero floating-point exception  
 24152 shall be raised.

24153       If  $iz$  is  $x + i\text{Inf}$  where  $x$  is positive-signed and finite,  $-i (+0 + i\pi/2)$  shall be returned.

24154       If  $iz$  is  $x + i\text{NaN}$  where  $x$  is non-zero and finite,  $-i (\text{NaN} + i\text{NaN})$  shall be returned and the  
 24155 invalid floating-point exception may be raised.

24156       If  $iz$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $-i (+0 + i\pi/2)$  shall be returned.

24157       If  $iz$  is  $+\text{Inf} + i\text{Inf}$ ,  $-i (+0 + i\pi/2)$  shall be returned.

24158       If  $iz$  is  $+\text{Inf} + i\text{NaN}$ ,  $-i (+0 + i\text{NaN})$  shall be returned.

24159       If  $iz$  is  $\text{NaN} + iy$  where  $y$  is finite,  $-i (\text{NaN} + i\text{NaN})$  shall be returned and the invalid floating-  
 24160 point exception may be raised.

24161       If  $iz$  is  $\text{NaN} + i\text{Inf}$ ,  $-i (\pm 0 + i\pi/2)$  shall be returned; the sign of the imaginary part of the result is  
 24162 unspecified.

24163       If  $iz$  is  $\text{NaN} + i\text{NaN}$ ,  $-i (\text{NaN} + i\text{NaN})$  shall be returned.

24164 **ERRORS**

24165       No errors are defined.

24166 **EXAMPLES**

24167 None.

24168 **APPLICATION USAGE**

24169 None.

24170 **RATIONALE**

24171 The MXC special cases for *catan()* are derived from those for *catanh()* by applying the formula  
24172  $catan(z) = -i \operatorname{catanh}(iz)$ .

24173 **FUTURE DIRECTIONS**

24174 None.

24175 **SEE ALSO**24176 *catanh()*, *ctan()*24177 XBD <**complex.h**>24178 **CHANGE HISTORY**

24179 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24180 **Issue 8**

24181 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24182 standard.

24183 **NAME**

24184 catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

24185 **SYNOPSIS**

```
24186 #include <complex.h>
24187 double complex catanh(double complex z);
24188 float complex catanhf(float complex z);
24189 long double complex catanhl(long double complex z);
```

24190 **DESCRIPTION**

24191 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24192 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24193 volume of POSIX.1-2024 defers to the ISO C standard.

24194 These functions shall compute the complex arc hyperbolic tangent of  $z$ , with branch cuts outside  
 24195 the interval  $[-1, +1]$  along the real axis.

24196 **RETURN VALUE**

24197 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip  
 24198 mathematically unbounded along the real axis and in the interval  $[-i\pi/2, +i\pi/2]$  along the  
 24199 imaginary axis.

24200 MXC *catanh(conj(z))*, *catanhf(conjf(z))*, and *catanhl(conjl(z))* shall return exactly the same value as  
 24201 *conj(catanh(z))*, *conjf(catanhf(z))*, and *conjl(catanhl(z))*, respectively, and *catanh(-z)*, *catanhf(-z)*,  
 24202 and *catanhl(-z)* shall return exactly the same value as  $-catanh(z)$ ,  $-catanhf(z)$ , and  $-catanhl(z)$ ,  
 24203 respectively, including for the special values of  $z$  below.

24204 If  $z$  is  $+0 + i0$ ,  $+0 + i0$  shall be returned.

24205 If  $z$  is  $+0 + iNaN$ ,  $+0 + iNaN$  shall be returned.

24206 If  $z$  is  $+1 + i0$ ,  $+Inf + i0$  shall be returned and the divide-by-zero floating-point exception shall be  
 24207 raised.

24208 If  $z$  is  $x + iInf$  where  $x$  is positive-signed and finite,  $+0 + i\pi/2$  shall be returned.

24209 If  $z$  is  $x + iNaN$  where  $x$  is non-zero and finite,  $NaN + iNaN$  shall be returned and the invalid  
 24210 floating-point exception may be raised.

24211 If  $z$  is  $+Inf + iy$  where  $y$  is positive-signed and finite,  $+0 + i\pi/2$  shall be returned.

24212 If  $z$  is  $+Inf + iInf$ ,  $+0 + i\pi/2$  shall be returned.

24213 If  $z$  is  $+Inf + iNaN$ ,  $+0 + iNaN$  shall be returned.

24214 If  $z$  is  $NaN + iy$  where  $y$  is finite,  $NaN + iNaN$  shall be returned and the invalid floating-point  
 24215 exception may be raised.

24216 If  $z$  is  $NaN + iInf$ ,  $\pm 0 + i\pi/2$  shall be returned; the sign of the real part of the result is unspecified.

24217 If  $z$  is  $NaN + iNaN$ ,  $NaN + iNaN$  shall be returned.

24218 **ERRORS**

24219 No errors are defined.

24220 **EXAMPLES**

24221 None.

24222 **APPLICATION USAGE**

24223 None.

24224 **RATIONALE**

24225 None.

24226 **FUTURE DIRECTIONS**

24227 None.

24228 **SEE ALSO**24229 [ctanh\(\)](#)24230 XBD <[complex.h](#)>24231 **CHANGE HISTORY**

24232 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24233 **Issue 8**24234 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24235 standard.



24236 **NAME**

24237           catanl — complex arc tangent functions

24238 **SYNOPSIS**

24239           #include &lt;complex.h&gt;

24240           long double complex catanl(long double complex z);

24241 **DESCRIPTION**24242           Refer to *catan()*.

24243 **NAME**

24244 catclose — close a message catalog descriptor

24245 **SYNOPSIS**

24246 #include &lt;nl\_types.h&gt;

24247 int catclose(nl\_catd catd);

24248 **DESCRIPTION**24249 The *catclose()* function shall close the message catalog identified by *catd*. If a file descriptor is  
24250 used to implement the type **nl\_catd**, that file descriptor shall be closed.24251 **RETURN VALUE**24252 Upon successful completion, *catclose()* shall return 0; otherwise, -1 shall be returned, and *errno*  
24253 set to indicate the error.24254 **ERRORS**24255 The *catclose()* function may fail if:

24256 [EBADF] The catalog descriptor is not valid.

24257 [EINTR] The *catclose()* function was interrupted by a signal.24258 **EXAMPLES**

24259 None.

24260 **APPLICATION USAGE**

24261 None.

24262 **RATIONALE**

24263 None.

24264 **FUTURE DIRECTIONS**

24265 None.

24266 **SEE ALSO**24267 *catgets()*, *catopen()*

24268 XBD &lt;nl\_types.h&gt;

24269 **CHANGE HISTORY**

24270 First released in Issue 2.

24271 **Issue 7**24272 The *catclose()* function is moved from the XSI option to the Base.

24273 **NAME**24274 `catgets` — read a program message24275 **SYNOPSIS**24276 `#include <nl_types.h>`24277 `char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);`24278 **DESCRIPTION**

24279 The `catgets()` function shall attempt to read message `msg_id`, in set `set_id`, from the message  
 24280 catalog identified by `catd`. The `catd` argument is a message catalog descriptor returned from an  
 24281 earlier call to `catopen()`. The results are undefined if `catd` is not a value returned by `catopen()` for  
 24282 a message catalog still open in the process. The `s` argument points to a default message string  
 24283 which shall be returned by `catgets()` if it cannot retrieve the identified message.

24284 The `catgets()` function need not be thread-safe.24285 **RETURN VALUE**

24286 If the identified message is retrieved successfully, `catgets()` shall return a pointer to an internal  
 24287 buffer area containing the null-terminated message string. If the call is unsuccessful for any  
 24288 reason, `s` shall be returned and `errno` shall be set to indicate the error.

24289 **ERRORS**24290 The `catgets()` function shall fail if:

24291 [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
 24292 was transferred.

24293 [ENOMSG] The message identified by `set_id` and `msg_id` is not in the message catalog.

24294 The `catgets()` function may fail if:

24295 [EBADF] The `catd` argument is not a valid message catalog descriptor open for reading.

24296 [EBADMSG] The message identified by `set_id` and `msg_id` in the specified message catalog  
 24297 did not satisfy implementation-defined security criteria.

24298 [EINVAL] The message catalog identified by `catd` is corrupted.

24299 **EXAMPLES**

24300 None.

24301 **APPLICATION USAGE**

24302 None.

24303 **RATIONALE**

24304 None.

24305 **FUTURE DIRECTIONS**

24306 None.

24307 **SEE ALSO**24308 `catclose()`, `catopen()`24309 XBD `<nl_types.h>`24310 **CHANGE HISTORY**

24311 First released in Issue 2.

24312 **Issue 5**

24313 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

24314 **Issue 6**

24315 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

24316 **Issue 7**

24317 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [EINTR] and [ENOMSG] errors to become “shall fail” errors, updating the RETURN VALUE section, and updating the DESCRIPTION to note that: “The results are undefined if *catd* is not a value returned by *catopen()* for a message catalog still open in the process.”

24321 The *catgets()* function is moved from the XSI option to the Base.

24322 **NAME**

24323 catopen — open a message catalog

24324 **SYNOPSIS**

24325 #include &lt;nl\_types.h&gt;

24326 nl\_catd catopen(const char \*name, int oflag);

24327 **DESCRIPTION**

24328 The *catopen()* function shall open a message catalog and return a message catalog descriptor.  
 24329 The *name* argument specifies the name of the message catalog to be opened. If *name* contains a  
 24330 XSI `'/'`, then *name* specifies a pathname for the message catalog. Otherwise, the environment  
 24331 variable *NLSPATH* is used with *name* substituted for the `%N` conversion specification (see XBD  
 24332 Chapter 8, on page 167); if *NLSPATH* exists in the environment when the process starts, then if  
 24333 the process has appropriate privileges, the behavior of *catopen()* is undefined. If *NLSPATH* does  
 24334 not exist in the environment, or if a message catalog cannot be found in any of the components  
 24335 specified by *NLSPATH*, then an implementation-defined default path shall be used. This default  
 24336 may be affected by the setting of *LC\_MESSAGES* if the value of *oflag* is *NL\_CAT\_LOCALE*, or  
 24337 XSI the *LANG* environment variable if *oflag* is 0. When searching *NLSPATH*, *catopen()* shall ignore  
 24338 any files it finds that are not valid message catalog files.

24339 A message catalog descriptor shall remain valid in a process until that process closes it, or a  
 24340 successful call to one of the *exec* functions. A change in the setting of the *LC\_MESSAGES*  
 24341 category may invalidate existing open catalogs.

24342 If a file descriptor is used to implement message catalog descriptors, the *FD\_CLOEXEC* flag  
 24343 shall be set; see <*fcntl.h*>.

24344 If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the  
 24345 catalog without regard to the *LC\_MESSAGES* category. If the *oflag* argument is  
 24346 *NL\_CAT\_LOCALE*, the *LC\_MESSAGES* category is used to locate the message catalog (see XBD  
 24347 Section 8.2, on page 169).

24348 **RETURN VALUE**

24349 Upon successful completion, *catopen()* shall return a message catalog descriptor for use on  
 24350 subsequent calls to *catgets()* and *catclose()*. Otherwise, *catopen()* shall return (*nl\_catd*) -1 and set  
 24351 *errno* to indicate the error.

24352 **ERRORS**24353 The *catopen()* function may fail if:

24354 [EACCES] Search permission is denied for the component of the path prefix of the  
 24355 message catalog or read permission is denied for the message catalog.

24356 [EMFILE] All file descriptors available to the process are currently open.

24357 [ENAMETOOLONG]

24358 The length of a component of a pathname is longer than {NAME\_MAX}.

24359 [ENAMETOOLONG]

24360 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 24361 symbolic link produced an intermediate result with a length that exceeds  
 24362 {PATH\_MAX}.

24363 [ENFILE] Too many files are currently open in the system.

24364 [ENOENT] The *name* argument contains a `'/'` and does not name an existing message  
 24365 XSI catalog, the *name* argument does not contain a `'/'` and searching *NLSPATH* (if  
 24366 set) and then the implementation-defined default path for a message catalog

24367 with that name failed, one or more files exist but all are of an invalid format,  
24368 or the *name* argument points to an empty string.

24369 [ENOMEM] Insufficient storage space is available.

24370 [ENOTDIR] A component of the path prefix of the message catalog names an existing file  
24371 that is neither a directory nor a symbolic link to a directory, or the pathname  
24372 of the message catalog contains at least one non-`<slash>` character and ends  
24373 with one or more trailing `<slash>` characters and the last pathname  
24374 component names an existing file that is neither a directory nor a symbolic  
24375 link to a directory.

#### 24376 EXAMPLES

24377 None.

#### 24378 APPLICATION USAGE

24379 Some implementations of *catopen()* use *malloc()* to allocate space for internal buffer areas. The  
24380 *catopen()* function may fail if there is insufficient storage space available to accommodate these  
24381 buffers.

24382 Conforming applications must assume that message catalog descriptors are not valid after a call  
24383 to one of the *exec* functions.

24384 Application developers should be aware that guidelines for the location of message catalogs  
24385 have not yet been developed. Therefore they should take care to avoid conflicting with catalogs  
24386 used by other applications and the standard utilities.

24387 To be sure that messages produced by an application running with appropriate privileges cannot  
24388 be used by an attacker setting an unexpected value for *NLSPATH* in the environment to confuse  
24389 a system administrator, such applications should use pathnames containing a `'/'` to get defined  
24390 behavior when using *catopen()* to open a message catalog.

#### 24391 RATIONALE

24392 None.

#### 24393 FUTURE DIRECTIONS

24394 None.

#### 24395 SEE ALSO

24396 *catclose()*, *catgets()*

24397 XBD Chapter 8 (on page 167), `<fcntl.h>`, `<nl_types.h>`,

#### 24398 CHANGE HISTORY

24399 First released in Issue 2.

#### 24400 Issue 7

24401 Austin Group Interpretation 1003.1-2001 #143 is applied.

24402 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

24403 The *catopen()* function is moved from the XSI option to the Base.

24404 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0045 [324] is applied.

24405 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0054 [645], XSH/TC2-2008/0055 [497],  
24406 and XSH/TC2-2008/0056 [497] are applied.

24407 **Issue 8**

24408 Austin Group Defect 1122 is applied, clarifying that *catopen()* ignores files that are not valid  
24409 message catalog files when performing an *NLSPATH* search.

24410 Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

24411 **NAME**

24412 cbrt, cbrtf, cbrtl — cube root functions

24413 **SYNOPSIS**

```
24414 #include <math.h>
24415 double cbrt(double x);
24416 float cbrtf(float x);
24417 long double cbrtl(long double x);
```

24418 **DESCRIPTION**

24419 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24420 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24421 volume of POSIX.1-2024 defers to the ISO C standard.

24422 These functions shall compute the real cube root of their argument  $x$ .24423 **RETURN VALUE**24424 Upon successful completion, these functions shall return the cube root of  $x$ .24425 MX If  $x$  is NaN, a NaN shall be returned.24426 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.24427 **ERRORS**

24428 No errors are defined.

24429 **EXAMPLES**

24430 None.

24431 **APPLICATION USAGE**

24432 None.

24433 **RATIONALE**

24434 For some applications, a true cube root function, which returns negative results for negative  
 24435 arguments, is more appropriate than  $\text{pow}(x, 1.0/3.0)$ , which returns a NaN for  $x$  less than 0.

24436 **FUTURE DIRECTIONS**

24437 None.

24438 **SEE ALSO**24439 XBD [<math.h>](#)24440 **CHANGE HISTORY**

24441 First released in Issue 4, Version 2.

24442 **Issue 5**

24443 Moved from X/OPEN UNIX extension to BASE.

24444 **Issue 6**24445 The `cbrt()` function is no longer marked as an extension.24446 The `cbrtf()` and `cbrtl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

24447 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 24448 revised to align with the ISO/IEC 9899:1999 standard.

24449 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 24450 marked.



24451 **NAME**24452 `ccos, ccosf, ccosl` — complex cosine functions24453 **SYNOPSIS**24454 `#include <complex.h>`24455 `double complex ccos(double complex z);`24456 `float complex ccosf(float complex z);`24457 `long double complex ccosl(long double complex z);`24458 **DESCRIPTION**

24459 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24460 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24461 volume of POSIX.1-2024 defers to the ISO C standard.

24462 These functions shall compute the complex cosine of  $z$ .24463 **RETURN VALUE**

24464 These functions shall return the complex cosine value.

24465 MXC `ccos(conj(iz))`, `ccosf(conj(iz))`, and `ccosl(conj(iz))` shall return exactly the same value as  
 24466 `conj(ccos(iz))`, `conjf(ccosf(iz))`, and `conjl(ccosl(iz))`, respectively, and `ccos(-iz)`, `ccosf(-iz)`, and  
 24467 `ccosl(-iz)` shall return exactly the same value as `ccos(iz)`, `ccosf(iz)`, and `ccosl(iz)`, respectively,  
 24468 including for the special values of  $iz$  below.

24469 If  $iz$  is  $+0 + i0$ ,  $1 + i0$  shall be returned.

24470 If  $iz$  is  $+0 + i\text{Inf}$ ,  $\text{NaN} \pm i0$  shall be returned and the invalid floating-point exception shall be  
 24471 raised; the sign of the imaginary part of the result is unspecified.

24472 If  $iz$  is  $+0 + i\text{NaN}$ ,  $\text{NaN} \pm i0$  shall be returned; the sign of the imaginary part of the result is  
 24473 unspecified.

24474 If  $iz$  is  $x + i\text{Inf}$  where  $x$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 24475 floating-point exception shall be raised.

24476 If  $iz$  is  $x + i\text{NaN}$  where  $x$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 24477 floating-point exception may be raised.

24478 If  $iz$  is  $+\text{Inf} + i0$ ,  $+\text{Inf} + i0$  shall be returned.24479 If  $iz$  is  $+\text{Inf} + iy$  where  $y$  is non-zero and finite,  $+\text{Inf} (\cos(y) + i \sin(y))$  shall be returned.

24480 If  $iz$  is  $+\text{Inf} + i\text{Inf}$ ,  $\pm\text{Inf} + i\text{NaN}$  shall be returned and the invalid floating-point exception shall be  
 24481 raised; the sign of the real part of the result is unspecified.

24482 If  $iz$  is  $+\text{Inf} + i\text{NaN}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.

24483 If  $iz$  is  $\text{NaN} + i0$ ,  $\text{NaN} \pm i0$  shall be returned; the sign of the imaginary part of the result is  
 24484 unspecified.

24485 If  $iz$  is  $\text{NaN} + iy$  where  $y$  is any non-zero number,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 24486 floating-point exception may be raised.

24487 If  $iz$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.24488 **ERRORS**

24489 No errors are defined.

24490 **EXAMPLES**

24491 None.

24492 **APPLICATION USAGE**

24493 None.

24494 **RATIONALE**

24495 The MXC special cases for `ccos()` are derived from those for `ccosh()` by applying the formula  
24496  $ccos(z) = ccosh(iz)$ .

24497 **FUTURE DIRECTIONS**

24498 None.

24499 **SEE ALSO**24500 [ccosh\(\)](#), [cacos\(\)](#)24501 XBD [<complex.h>](#)24502 **CHANGE HISTORY**

24503 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24504 **Issue 8**

24505 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24506 standard.

24507 **NAME**

24508 ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

24509 **SYNOPSIS**

24510 #include &lt;complex.h&gt;

24511 double complex ccosh(double complex z);

24512 float complex ccoshf(float complex z);

24513 long double complex ccoshl(long double complex z);

24514 **DESCRIPTION**

24515 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24516 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24517 volume of POSIX.1-2024 defers to the ISO C standard.

24518 These functions shall compute the complex hyperbolic cosine of  $z$ .24519 **RETURN VALUE**

24520 These functions shall return the complex hyperbolic cosine value.

24521 MXC  $ccosh(conj(z))$ ,  $ccoshf(conjf(z))$ , and  $ccoshl(conjl(z))$  shall return exactly the same value as  
 24522  $conj(ccosh(z))$ ,  $conjf(ccoshf(z))$ , and  $conjl(ccoshl(z))$ , respectively, and  $ccosh(-z)$ ,  $ccoshf(-z)$ , and  
 24523  $ccoshl(-z)$  shall return exactly the same value as  $ccosh(z)$ ,  $ccoshf(z)$ , and  $ccoshl(z)$ , respectively,  
 24524 including for the special values of  $z$  below.

24525 If  $z$  is  $+0 + i0$ ,  $1 + i0$  shall be returned.

24526 If  $z$  is  $+0 + i\text{Inf}$ ,  $\text{NaN} \pm i0$  shall be returned and the invalid floating-point exception shall be  
 24527 raised; the sign of the imaginary part of the result is unspecified.

24528 If  $z$  is  $+0 + i\text{NaN}$ ,  $\text{NaN} \pm i0$  shall be returned; the sign of the imaginary part of the result is  
 24529 unspecified.

24530 If  $z$  is  $x + i\text{Inf}$  where  $x$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 24531 floating-point exception shall be raised.

24532 If  $z$  is  $x + i\text{NaN}$  where  $x$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 24533 floating-point exception may be raised.

24534 If  $z$  is  $+\text{Inf} + i0$ ,  $+\text{Inf} + i0$  shall be returned.24535 If  $z$  is  $+\text{Inf} + iy$  where  $y$  is non-zero and finite,  $+\text{Inf} (\cos(y) + i \sin(y))$  shall be returned.

24536 If  $z$  is  $+\text{Inf} + i\text{Inf}$ ,  $\pm\text{Inf} + i\text{NaN}$  shall be returned and the invalid floating-point exception shall be  
 24537 raised; the sign of the real part of the result is unspecified.

24538 If  $z$  is  $+\text{Inf} + i\text{NaN}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.

24539 If  $z$  is  $\text{NaN} + i0$ ,  $\text{NaN} \pm i0$  shall be returned; the sign of the imaginary part of the result is  
 24540 unspecified.

24541 If  $z$  is  $\text{NaN} + iy$  where  $y$  is any non-zero number,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 24542 floating-point exception may be raised.

24543 If  $z$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.24544 **ERRORS**

24545 No errors are defined.

24546 **EXAMPLES**

24547 None.

24548 **APPLICATION USAGE**

24549 None.

24550 **RATIONALE**

24551 None.

24552 **FUTURE DIRECTIONS**

24553 None.

24554 **SEE ALSO**24555 [cacosh\(\)](#)24556 XBD [<complex.h>](#)24557 **CHANGE HISTORY**

24558 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24559 **Issue 8**24560 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24561 standard.

24562 **NAME**

24563           ccosl — complex cosine functions

24564 **SYNOPSIS**

24565           #include <complex.h>

24566           long double complex ccosl(long double complex z);

24567 **DESCRIPTION**

24568           Refer to *ccos()*.

24569 **NAME**

24570           ceil, ceilf, ceill — ceiling value function

24571 **SYNOPSIS**

```
24572       #include <math.h>
24573       double ceil(double x);
24574       float ceilf(float x);
24575       long double ceill(long double x);
```

24576 **DESCRIPTION**

24577 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 24578       conflict between the requirements described here and the ISO C standard is unintentional. This  
 24579       volume of POSIX.1-2024 defers to the ISO C standard.

24580       These functions shall compute the smallest integral value not less than  $x$ .

24581 MX       These functions may raise the inexact floating-point exception for finite non-integer arguments.

24582 **RETURN VALUE**

24583 MX       The returned value shall be independent of the current rounding direction mode and shall have  
 24584       the same sign as  $x$ .

24585       Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not  
 24586       less than  $x$ , expressed as a type **double**, **float**, or **long double**, respectively.

24587 MX       If  $x$  is NaN, a NaN shall be returned.

24588       If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.

24589 **ERRORS**

24590       No errors are defined.

24591 **EXAMPLES**

24592       None.

24593 **APPLICATION USAGE**

24594       The integral value returned by these functions need not be expressible as an **intmax\_t**. The  
 24595       return value should be tested before assigning it to an integer type to avoid the undefined  
 24596       results of an integer overflow.

24597 **RATIONALE**

24598       None.

24599 **FUTURE DIRECTIONS**

24600       None.

24601 **SEE ALSO**

24602       *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*

24603       XBD Section 4.23 (on page 109), **<math.h>**

24604 **CHANGE HISTORY**

24605       First released in Issue 1. Derived from Issue 1 of the SVID.

24606 **Issue 5**

24607       The DESCRIPTION is updated to indicate how an application should check for an error. This  
 24608       text was previously published in the APPLICATION USAGE section.

24609 **Issue 6**

24610 The *ceilf()* and *ceil()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

24611 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
24612 revised to align with the ISO/IEC 9899:1999 standard.

24613 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
24614 marked.

24615 **Issue 7**

24616 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0046 [346] is applied.

24617 **Issue 8**

24618 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24619 standard.

24620 **NAME**

24621 cexp, cexpf, cexpl — complex exponential functions

24622 **SYNOPSIS**

24623 #include &lt;complex.h&gt;

24624 double complex cexp(double complex z);

24625 float complex cexpf(float complex z);

24626 long double complex cexpl(long double complex z);

24627 **DESCRIPTION**

24628 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 24629 conflict between the requirements described here and the ISO C standard is unintentional. This  
 24630 volume of POSIX.1-2024 defers to the ISO C standard.

24631 These functions shall compute the complex exponent of  $z$ , defined as  $e^z$ .24632 **RETURN VALUE**24633 These functions shall return the complex exponential value of  $z$ .

24634 MXC  $cexp(conj(z))$ ,  $cexpf(conjf(z))$ , and  $cexpl(conjl(z))$  shall return exactly the same value as  
 24635  $conj(cexp(z))$ ,  $conjf(cexpf(z))$ , and  $conjl(cexpl(z))$ , respectively, including for the special values of  $z$   
 24636 below.

24637 If  $z$  is  $\pm 0 + i0$ ,  $1 + i0$  shall be returned.

24638 If  $z$  is  $x + i\text{Inf}$  where  $x$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
 24639 exception shall be raised.

24640 If  $z$  is  $x + i\text{NaN}$  where  $x$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
 24641 exception may be raised.

24642 If  $z$  is  $+\text{Inf} + i0$ ,  $+\text{Inf} + i0$  shall be returned.24643 If  $z$  is  $-\text{Inf} + iy$  where  $y$  is finite,  $+0 (\cos(y) + i \sin(y))$  shall be returned.24644 If  $z$  is  $+\text{Inf} + iy$  where  $y$  is non-zero and finite,  $+\text{Inf} (\cos(y) + i \sin(y))$  shall be returned.

24645 If  $z$  is  $-\text{Inf} + i\text{Inf}$ ,  $\pm 0 \pm i0$  shall be returned; the signs of the real and imaginary parts of the result  
 24646 are unspecified.

24647 If  $z$  is  $+\text{Inf} + i\text{Inf}$ ,  $\pm\text{Inf} + i\text{NaN}$  shall be returned and the invalid floating-point exception shall be  
 24648 raised; the sign of the real part of the result is unspecified.

24649 If  $z$  is  $-\text{Inf} + i\text{NaN}$ ,  $\pm 0 \pm i0$  shall be returned; the signs of the real and imaginary parts of the  
 24650 result are unspecified.

24651 If  $z$  is  $+\text{Inf} + i\text{NaN}$ ,  $\pm\text{Inf} + i\text{NaN}$  shall be returned; the sign of the real part of the result is  
 24652 unspecified.

24653 If  $z$  is  $\text{NaN} + i0$ ,  $\text{NaN} + i0$  shall be returned.

24654 If  $z$  is  $\text{NaN} + iy$  where  $y$  is any non-zero number,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 24655 floating-point exception may be raised.

24656 If  $z$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.24657 **ERRORS**

24658 No errors are defined.



24659 **EXAMPLES**

24660 None.

24661 **APPLICATION USAGE**

24662 None.

24663 **RATIONALE**

24664 None.

24665 **FUTURE DIRECTIONS**

24666 None.

24667 **SEE ALSO**24668 [clog\(\)](#)24669 XBD [<complex.h>](#)24670 **CHANGE HISTORY**

24671 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24672 **Issue 8**24673 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
24674 standard.

24675 **NAME**

24676 cfgetispeed — get input baud rate

24677 **SYNOPSIS**

24678 #include &lt;termios.h&gt;

24679 speed\_t cfgetispeed(const struct termios \*termios\_p);

24680 **DESCRIPTION**24681 The *cfgetispeed()* function shall extract the input baud rate from the **termios** structure to which  
24682 the *termios\_p* argument points.24683 This function shall return exactly the value in the **termios** data structure, without interpretation.24684 **RETURN VALUE**24685 Upon successful completion, *cfgetispeed()* shall return a value of type **speed\_t** representing the  
24686 input baud rate.24687 **ERRORS**

24688 No errors are defined.

24689 **EXAMPLES**

24690 None.

24691 **APPLICATION USAGE**

24692 None.

24693 **RATIONALE**24694 The term “baud” is used historically here, but is not technically correct. This is properly “bits per  
24695 second”, which may not be the same as baud. However, the term is used because of the  
24696 historical usage and understanding.24697 The *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, and *cfsetispeed()* functions do not take arguments as  
24698 numbers, but rather as symbolic names. There are two reasons for this:

- 24699 1. Historically, numbers were not used because of the way the rate was stored in the data
- 
- 24700 structure. This is retained even though a function is now used.
- 
- 24701 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
- 
- 24702 the application to that set.

24703 There is nothing to prevent an implementation accepting as an extension a number (such as 126),  
24704 and since the encoding of the Bxxx symbols is not specified, this can be done to avoid  
24705 introducing ambiguity.24706 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications  
24707 in this volume of POSIX.1-2024 have made it possible to determine whether split rates are  
24708 supported and to support them without having to treat zero as a special case. Since this  
24709 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is  
24710 the literal constant 0, not the symbolic constant B0. This volume of POSIX.1-2024 does not  
24711 preclude B0 from being defined as the value 0; in fact, implementations would likely benefit  
24712 from the two being equivalent. This volume of POSIX.1-2024 does not fully specify whether the  
24713 previous *cfsetispeed()* value is retained after a *tcgetattr()* as the actual value or as zero. Therefore,  
24714 conforming applications should always set both the input speed and output speed when setting  
24715 either.24716 In historical implementations, the baud rate information is traditionally kept in **c\_cflag**.  
24717 Applications should be written to presume that this might be the case (and thus not blindly copy  
24718 **c\_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c\_cflag**  
24719 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

24720 unused parts of the flag fields might be used by the implementation and should not be blindly  
24721 copied from the descriptions of one terminal device to another.

24722 **FUTURE DIRECTIONS**

24723 None.

24724 **SEE ALSO**

24725 *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*

24726 XBD Chapter 11 (on page 199), [<termios.h>](#)

24727 **CHANGE HISTORY**

24728 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

24729 **NAME**

24730 cfgetospeed — get output baud rate

24731 **SYNOPSIS**

24732 #include &lt;termios.h&gt;

24733 speed\_t cfgetospeed(const struct termios \*termios\_p);

24734 **DESCRIPTION**24735 The *cfgetospeed()* function shall extract the output baud rate from the **termios** structure to which  
24736 the *termios\_p* argument points.24737 This function shall return exactly the value in the **termios** data structure, without interpretation.24738 **RETURN VALUE**24739 Upon successful completion, *cfgetospeed()* shall return a value of type **speed\_t** representing the  
24740 output baud rate.24741 **ERRORS**

24742 No errors are defined.

24743 **EXAMPLES**

24744 None.

24745 **APPLICATION USAGE**

24746 None.

24747 **RATIONALE**24748 Refer to *cfgetispeed()*.24749 **FUTURE DIRECTIONS**

24750 None.

24751 **SEE ALSO**24752 *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*24753 XBD Chapter 11 (on page 199), <**termios.h**>24754 **CHANGE HISTORY**

24755 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

24756 **NAME**

24757 cfsetispeed — set input baud rate

24758 **SYNOPSIS**

24759 #include &lt;termios.h&gt;

24760 int cfsetispeed(struct termios \*termios\_p, speed\_t speed);

24761 **DESCRIPTION**24762 The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by  
24763 *termios\_p* to *speed*.24764 There shall be no effect on the baud rates set in the hardware until a subsequent successful call  
24765 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set  
24766 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*  
24767 function is called.24768 **RETURN VALUE**24769 Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and  
24770 *errno* may be set to indicate the error.24771 **ERRORS**24772 The *cfsetispeed()* function may fail if:24773 [EINVAL] The *speed* value is not a valid baud rate.24774 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in  
24775 **<termios.h>**.24776 **EXAMPLES**

24777 None.

24778 **APPLICATION USAGE**

24779 None.

24780 **RATIONALE**24781 Refer to *cfgetispeed()*.24782 **FUTURE DIRECTIONS**

24783 None.

24784 **SEE ALSO**24785 *cfgetispeed()*, *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*24786 XBD Chapter 11 (on page 199), **<termios.h>**24787 **CHANGE HISTORY**

24788 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

24789 **Issue 6**24790 The following new requirements on POSIX implementations derive from alignment with the  
24791 Single UNIX Specification:

- 24792
- The optional setting of *errno* and the [EINVAL] error conditions are added.

24793 **NAME**

24794 cfsetospeed — set output baud rate

24795 **SYNOPSIS**

24796 #include &lt;termios.h&gt;

24797 int cfsetospeed(struct termios \*termios\_p, speed\_t speed);

24798 **DESCRIPTION**24799 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by  
24800 *termios\_p* to *speed*.24801 There shall be no effect on the baud rates set in the hardware until a subsequent successful call  
24802 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set  
24803 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*  
24804 function is called.24805 **RETURN VALUE**24806 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*  
24807 may be set to indicate the error.24808 **ERRORS**24809 The *cfsetospeed()* function may fail if:24810 [EINVAL] The *speed* value is not a valid baud rate.24811 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in  
24812 <termios.h>.24813 **EXAMPLES**

24814 None.

24815 **APPLICATION USAGE**

24816 None.

24817 **RATIONALE**24818 Refer to *cfgetispeed()*.24819 **FUTURE DIRECTIONS**

24820 None.

24821 **SEE ALSO**24822 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*

24823 XBD Chapter 11 (on page 199), &lt;termios.h&gt;

24824 **CHANGE HISTORY**

24825 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

24826 **Issue 6**24827 The following new requirements on POSIX implementations derive from alignment with the  
24828 Single UNIX Specification:

- 24829
- The optional setting of *errno* and the [EINVAL] error conditions are added.

24830 **NAME**

24831 chdir — change working directory

24832 **SYNOPSIS**

```
24833 #include <unistd.h>
24834 int chdir(const char *path);
```

24835 **DESCRIPTION**

24836 The *chdir()* function shall cause the directory named by the pathname pointed to by the *path*  
 24837 argument to become the current working directory; that is, the starting point for path searches  
 24838 for pathnames not beginning with '/ '.

24839 **RETURN VALUE**

24840 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current  
 24841 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

24842 **ERRORS**

24843 The *chdir()* function shall fail if:

- 24844 [EACCES] Search permission is denied for any component of the pathname.
- 24845 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
24846 argument.
- 24847 [ENAMETOOLONG]  
24848 The length of a component of a pathname is longer than {NAME\_MAX}.
- 24849 [ENOENT] A component of *path* does not name an existing directory or *path* is an empty  
24850 string.
- 24851 [ENOTDIR] A component of the pathname names an existing file that is neither a directory  
24852 nor a symbolic link to a directory.

24853 The *chdir()* function may fail if:

- 24854 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
24855 resolution of the *path* argument.
- 24856 [ENAMETOOLONG]  
24857 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
24858 symbolic link produced an intermediate result with a length that exceeds  
24859 {PATH\_MAX}.

24860 **EXAMPLES**24861 **Changing the Current Working Directory**

24862 The following example makes the value pointed to by **directory**, */tmp*, the current working  
 24863 directory.

```
24864 #include <unistd.h>
24865 ...
24866 char *directory = "/tmp";
24867 int ret;
24868 ret = chdir (directory);
```

**24869 APPLICATION USAGE**

24870 None.

**24871 RATIONALE**

24872 The *chdir()* function only affects the working directory of the current process.

**24873 FUTURE DIRECTIONS**

24874 None.

**24875 SEE ALSO**

24876 [\*getcwd\(\)\*](#)

24877 XBD [\*\*<unistd.h>\*\*](#)

**24878 CHANGE HISTORY**

24879 First released in Issue 1. Derived from Issue 1 of the SVID.

**24880 Issue 6**

24881 The APPLICATION USAGE section is added.

24882 The following new requirements on POSIX implementations derive from alignment with the  
24883 Single UNIX Specification:

- 24884 • The [ELOOP] mandatory error condition is added.
- 24885 • A second [ENAMETOOLONG] is added as an optional error condition.

24886 The following changes were made to align with the IEEE P1003.1a draft standard:

- 24887 • The [ELOOP] optional error condition is added.

**24888 Issue 7**

24889 Austin Group Interpretation 1003.1-2001 #143 is applied.

24890 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0047 [324] is applied.



24891 **NAME**

24892 chmod, fchmodat — change mode of a file

24893 **SYNOPSIS**

24894 #include &lt;sys/stat.h&gt;

24895 int chmod(const char \*path, mode\_t mode);

24896 OH #include &lt;fcntl.h&gt;

24897 int fchmodat(int fd, const char \*path, mode\_t mode, int flag);

24898 **DESCRIPTION**

24899 XSI The `chmod()` function shall change S\_ISUID, S\_ISGID, S\_ISVTX, and the file permission bits of  
 24900 the file named by the pathname pointed to by the `path` argument to the corresponding bits in the  
 24901 `mode` argument. If any bits that can be set in the `st_mode` value returned by `lstat()` or `stat()` but  
 24902 cannot be changed using `chmod()`, such as the bits that are used to encode the file type, are set in  
 24903 the `mode` argument, these *read-only st\_mode bits* shall be ignored.

24904 If the effective user ID of the process does not match the owner of the file and the process does  
 24905 not have appropriate privileges, the `chmod()` function shall fail.

24906 XSI S\_ISUID, S\_ISGID, S\_ISVTX, and the file permission bits are described in <sys/stat.h>.

24907 If the calling process does not have appropriate privileges, and if the group ID of the file does  
 24908 not match the effective group ID or one of the supplementary group IDs and if the file is a  
 24909 regular file, bit S\_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon  
 24910 successful return from `chmod()`.

24911 Additional implementation-defined restrictions may cause the S\_ISUID and S\_ISGID bits in  
 24912 XSI `mode` to be ignored, and may cause the S\_ISVTX bit in `mode` to be ignored for non-directory files.

24913 Upon successful completion, `chmod()` shall mark for update the last file status change timestamp  
 24914 of the file.

24915 The `fchmodat()` function shall be equivalent to the `chmod()` function except in the case where `path`  
 24916 specifies a relative path. In this case the file to be changed is determined relative to the directory  
 24917 associated with the file descriptor `fd` instead of the current working directory. If the access mode  
 24918 of the open file description associated with the file descriptor is not O\_SEARCH, the function  
 24919 shall check whether directory searches are permitted using the current permissions of the  
 24920 directory underlying the file descriptor. If the access mode is O\_SEARCH, the function shall not  
 24921 perform the check.

24922 Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 24923 in <fcntl.h>:

24924 AT\_SYMLINK\_NOFOLLOW

24925 If `path` names a symbolic link, then the mode of the symbolic link is changed.

24926 If `fchmodat()` is passed the special value AT\_FDCWD in the `fd` parameter, the current working  
 24927 directory shall be used. If also `flag` is zero, the behavior shall be identical to a call to `chmod()`.

24928 **RETURN VALUE**

24929 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 24930 return -1 and set `errno` to indicate the error. If -1 is returned, no change to the file mode occurs.

24931 **ERRORS**

24932 These functions shall fail if:

- 24933 [EACCES] Search permission is denied on a component of the path prefix.
- 24934 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
24935 argument.
- 24936 [ENAMETOOLONG]  
24937 The length of a component of a pathname is longer than {NAME\_MAX}.
- 24938 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 24939 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
24940 directory nor a symbolic link to a directory, or the *path* argument contains at  
24941 least one non-`<slash>` character and ends with one or more trailing `<slash>`  
24942 characters and the last pathname component names an existing file that is  
24943 neither a directory nor a symbolic link to a directory.
- 24944 [EPERM] The effective user ID does not match the owner of the file and the process does  
24945 not have appropriate privileges.
- 24946 [EROFS] The named file resides on a read-only file system.

24947 The *fchmodat()* function shall fail if:

- 24948 [EACCES] The access mode of the open file description associated with *fd* is not  
24949 O\_SEARCH and the permissions of the directory underlying *fd* do not permit  
24950 directory searches.
- 24951 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
24952 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.
- 24953 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated  
24954 with a non-directory file.

24955 These functions may fail if:

- 24956 [EINTR] A signal was caught during execution of the function.
- 24957 [EINVAL] The value of the *mode* argument, ignoring *read-only st\_mode bits* (see the  
24958 DESCRIPTION), is invalid.
- 24959 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
24960 resolution of the *path* argument.
- 24961 [ENAMETOOLONG]  
24962 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
24963 symbolic link produced an intermediate result with a length that exceeds  
24964 {PATH\_MAX}.

24965 The *fchmodat()* function may fail if:

- 24966 [EINVAL] The value of the *flag* argument is invalid.
- 24967 [EOPNOTSUPP] The AT\_SYMLINK\_NOFOLLOW bit is set in the *flag* argument, *path* names a  
24968 symbolic link, and the system does not support changing the mode of a  
24969 symbolic link.

24970 **EXAMPLES**24971 **Setting Read Permissions for User, Group, and Others**

24972 The following example sets read permissions for the owner, group, and others.

```
24973 #include <sys/stat.h>
24974 const char *path;
24975 ...
24976 chmod(path, S_IRUSR | S_IRGRP | S_IROTH);
```

24977 **Setting Read, Write, and Execute Permissions for the Owner Only**

24978 The following example sets read, write, and execute permissions for the owner, and no  
24979 permissions for group and others.

```
24980 #include <sys/stat.h>
24981 const char *path;
24982 ...
24983 chmod(path, S_IRWXU);
```

24984 **Setting Different Permissions for Owner, Group, and Other**

24985 The following example sets owner permissions for CHANGEFILE to read, write, and execute,  
24986 group permissions to read and execute, and other permissions to read.

```
24987 #include <sys/stat.h>
24988 #define CHANGEFILE "/etc/myfile"
24989 ...
24990 chmod(CHANGEFILE, S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH);
```

24991 **Modifying File Permissions**

24992 The following example adds group write permission to the existing permission bits for a file if  
24993 that bit is not already set.

```
24994 #include <sys/stat.h>
24995 struct stat sbuf;
24996 ...
24997 if (stat(path, &sbuf) == 0 && (sbuf.st_mode & S_IWGRP) == 0)
24998     chmod(path, sbuf.st_mode | S_IWGRP);
```

24999 **Setting and Checking File Permissions**

25000 The following example sets the file permission bits for a file named `/home/cnd/mod1`, then calls  
25001 the `stat()` function to verify the permissions.

```
25002 #include <sys/types.h>
25003 #include <sys/stat.h>
25004 int status;
25005 struct stat buffer
25006 ...
```

```

25007     chmod("/home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
25008     status = stat("/home/cnd/mod1", &buffer);

```

### 25009 APPLICATION USAGE

25010 In order to ensure that the S\_ISUID and S\_ISGID bits are set, an application requiring this  
 25011 should use *stat()* after a successful *chmod()* to verify this.

25012 Any file descriptors currently open by any process on the file could possibly become invalid if  
 25013 the mode of the file is changed to a value which would deny access to that process. One  
 25014 situation where this could occur is on a stateless file system. This behavior will not occur in a  
 25015 conforming environment.

### 25016 RATIONALE

25017 This volume of POSIX.1-2024 specifies that the S\_ISGID bit is cleared by *chmod()* on a regular file  
 25018 under certain conditions. This is specified on the assumption that regular files may be executed,  
 25019 and the system should prevent users from making executable *setgid()* files perform with  
 25020 privileges that the caller does not have. On implementations that support execution of other file  
 25021 types, the S\_ISGID bit should be cleared for those file types under the same circumstances.

25022 Implementations that use the S\_ISUID bit to indicate some other function (for example,  
 25023 mandatory record locking) on non-executable files need not clear this bit on writing. They  
 25024 should clear the bit for executable files and any other cases where the bit grants special powers  
 25025 to processes that change the file contents. Similar comments apply to the S\_ISGID bit.

25026 The purpose of the *fchmodat()* function is to enable changing the mode of files in directories  
 25027 other than the current working directory without exposure to race conditions. Any part of the  
 25028 path of a file could be changed in parallel to a call to *chmod()*, resulting in unspecified behavior.  
 25029 By opening a file descriptor for the target directory and using the *fchmodat()* function it can be  
 25030 guaranteed that the changed file is located relative to the desired directory. Some  
 25031 implementations might allow changing the mode of symbolic links. This is not supported by the  
 25032 interfaces in the POSIX specification. Systems with such support provide an interface named  
 25033 *lchmod()*. To support such implementations *fchmodat()* has a *flag* parameter.

### 25034 FUTURE DIRECTIONS

25035 None.

### 25036 SEE ALSO

25037 *access()*, *chown()*, *exec*, *fstatat()*, *fstatvfs()*, *mkdir()*, *mkfifo()*, *mknod()*, *open()*

25038 XBD <fcntl.h>, <sys/stat.h>, <sys/types.h>

### 25039 CHANGE HISTORY

25040 First released in Issue 1. Derived from Issue 1 of the SVID.

### 25041 Issue 6

25042 The following new requirements on POSIX implementations derive from alignment with the  
 25043 Single UNIX Specification:

- 25044 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 25045 required for conforming implementations of previous POSIX specifications, it was not  
 25046 required for UNIX applications.
- 25047 • The [EINVAL] and [EINTR] optional error conditions are added.
- 25048 • A second [ENAMETOOLONG] is added as an optional error condition.

25049 The following changes were made to align with the IEEE P1003.1a draft standard:

- 25050           • The [ELOOP] optional error condition is added.
- 25051           The normative text is updated to avoid use of the term “must” for application requirements.
- 25052 **Issue 7**
- 25053           Austin Group Interpretation 1003.1-2001 #143 is applied.
- 25054           The *chmodat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- 25055
- 25056           Changes are made related to support for finegrained timestamps.
- 25057           Changes are made to allow a directory to be opened for searching.
- 25058           The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
- 25059           pathname exists but is not a directory or a symbolic link to a directory.
- 25060           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0048 [300], XSH/TC1-2008/0049 [461],
- 25061           XSH/TC1-2008/0050 [324], XSH/TC1-2008/0051 [278], and XSH/TC1-2008/0052 [278] are
- 25062           applied.
- 25063           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0057 [873], XSH/TC2-2008/0058 [591],
- 25064           XSH/TC2-2008/0059 [817], XSH/TC2-2008/0060 [817], and XSH/TC2-2008/0061 [893] are
- 25065           applied.
- 25066 **Issue 8**
- 25067           Austin Group Defect 1024 is applied, allowing the S\_ISVTX bit to be ignored for non-directory
- 25068           files.
- 25069           Austin Group Defect 1283 is applied, clarifying that *chmod()* ignores *read-only st\_mode bits* in the
- 25070           *mode* argument.

25071 **NAME**

25072 chown, fchownat — change owner and group of a file

25073 **SYNOPSIS**

25074 #include &lt;unistd.h&gt;

25075 int chown(const char \*path, uid\_t owner, gid\_t group);

25076 OH #include &lt;fcntl.h&gt;

25077 int fchownat(int fd, const char \*path, uid\_t owner, gid\_t group,

25078 int flag);

25079 **DESCRIPTION**25080 The *chown()* function shall change the user and group ownership of a file.25081 The *path* argument points to a pathname naming a file. The user ID and group ID of the named  
25082 file shall be set to the numeric values contained in *owner* and *group*, respectively.25083 Only processes with an effective user ID equal to the user ID of the file or with appropriate  
25084 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for  
25085 *path*:

- 25086 • Changing the user ID is restricted to processes with appropriate privileges.
- 25087 • Changing the group ID is permitted to a process with an effective user ID equal to the user  
25088 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's  
25089 user ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to  
25090 one of its supplementary group IDs.

25091 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of  
25092 the file mode are set, and the process does not have appropriate privileges, the set-user-ID  
25093 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful  
25094 return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`,  
25095 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is  
25096 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()*  
25097 function is successfully invoked on a file that is not a regular file and one or more of the  
25098 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID  
25099 bits may be cleared.

25100 If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the  
25101 file shall not be changed.

25102 Upon successful completion, *chown()* shall mark for update the last file status change timestamp  
25103 of the file, except that if *owner* is `(uid_t)-1` and *group* is `(gid_t)-1`, the file status change  
25104 timestamp need not be marked for update.

25105 The *fchownat()* function shall be equivalent to the *chown()* and *lchown()* functions except in the  
25106 case where *path* specifies a relative path. In this case the file to be changed is determined relative  
25107 to the directory associated with the file descriptor *fd* instead of the current working directory. If  
25108 the access mode of the open file description associated with the file descriptor is not  
25109 `O_SEARCH`, the function shall check whether directory searches are permitted using the current  
25110 permissions of the directory underlying the file descriptor. If the access mode is `O_SEARCH`, the  
25111 function shall not perform the check.

25112 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
25113 in `<fcntl.h>`:

25114 AT\_SYMLINK\_NOFOLLOW  
 25115 If *path* names a symbolic link, ownership of the symbolic link is changed.

25116 If *fchownat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 25117 directory shall be used and the behavior shall be identical to a call to *chown()* or *lchown()*  
 25118 respectively, depending on whether or not the AT\_SYMLINK\_NOFOLLOW bit is set in the *flag*  
 25119 argument.

25120 **RETURN VALUE**  
 25121 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 25122 return  $-1$  and set *errno* to indicate the error. If  $-1$  is returned, no changes are made in the user ID  
 25123 and group ID of the file.

25124 **ERRORS**  
 25125 These functions shall fail if:

25126 [EACCES] Search permission is denied on a component of the path prefix.

25127 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 25128 argument.

25129 [ENAMETOOLONG]  
 25130 The length of a component of a pathname is longer than {NAME\_MAX}.

25131 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

25132 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
 25133 directory nor a symbolic link to a directory, or the *path* argument contains at  
 25134 least one non-`<slash>` character and ends with one or more trailing `<slash>`  
 25135 characters and the last pathname component names an existing file that is  
 25136 neither a directory nor a symbolic link to a directory.

25137 [EPERM] The effective user ID does not match the owner of the file, or the calling  
 25138 process does not have appropriate privileges and  
 25139 \_POSIX\_CHOWN\_RESTRICTED indicates that such privilege is required.

25140 [EROFS] The named file resides on a read-only file system.

25141 The *fchownat()* function shall fail if:

25142 [EACCES] The access mode of the open file description associated with *fd* is not  
 25143 O\_SEARCH and the permissions of the directory underlying *fd* do not permit  
 25144 directory searches.

25145 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
 25146 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.

25147 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated  
 25148 with a non-directory file.

25149 These functions may fail if:

25150 [EIO] An I/O error occurred while reading or writing to the file system.

25151 [EINTR] The *chown()* function was interrupted by a signal which was caught.

25152 [EINVAL] The owner or group ID supplied is not a value supported by the  
 25153 implementation.

25154 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
25155 resolution of the *path* argument.

25156 [ENAMETOOLONG]  
25157 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
25158 symbolic link produced an intermediate result with a length that exceeds  
25159 {PATH\_MAX}.

25160 The *fchownat()* function may fail if:

25161 [EINVAL] The value of the *flag* argument is not valid.

#### 25162 EXAMPLES

25163 None.

#### 25164 APPLICATION USAGE

25165 Although *chown()* can be used on some implementations by the file owner to change the owner  
25166 and group to any desired values, the only portable use of this function is to change the group of  
25167 a file to the effective GID of the calling process or to a member of its group set.

#### 25168 RATIONALE

25169 System III and System V allow a user to give away files; that is, the owner of a file may change  
25170 its user ID to anything. This is a serious problem for implementations that are intended to meet  
25171 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the  
25172 user ID of a file. Some government agencies (usually not ones concerned directly with security)  
25173 find this limitation too confining. This volume of POSIX.1-2024 uses *may* to permit secure  
25174 implementations while not disallowing System V.

25175 System III and System V allow the owner of a file to change the group ID to anything. Version 7  
25176 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to  
25177 change the group ID of a file to its effective group ID or to any of the groups in the list of  
25178 supplementary group IDs, but to no others.

25179 The POSIX.1-1990 standard requires that the *chown()* function invoked by a non-appropriate  
25180 privileged process clear the S\_ISGID and the S\_ISUID bits for regular files, and permits them to  
25181 be cleared for other types of files. This is so that changes in accessibility do not accidentally  
25182 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-  
25183 executable data files also clears the mandatory file locking bit (shared with S\_ISGID), which is  
25184 an extension on many implementations (it first appeared in System V). These bits should only be  
25185 required to be cleared on regular files that have one or more of their execute bits set.

25186 The purpose of the *fchownat()* function is to enable changing ownership of files in directories  
25187 other than the current working directory without exposure to race conditions. Any part of the  
25188 path of a file could be changed in parallel to a call to *chown()* or *lchown()*, resulting in  
25189 unspecified behavior. By opening a file descriptor for the target directory and using the  
25190 *fchownat()* function it can be guaranteed that the changed file is located relative to the desired  
25191 directory.

#### 25192 FUTURE DIRECTIONS

25193 None.

#### 25194 SEE ALSO

25195 [\*chmod\(\)\*](#), [\*fpathconf\(\)\*](#), [\*lchown\(\)\*](#)

25196 XBD [\*<fcntl.h>\*](#), [\*<sys/types.h>\*](#), [\*<unistd.h>\*](#)



25197 **CHANGE HISTORY**

25198 First released in Issue 1. Derived from Issue 1 of the SVID.

25199 **Issue 6**

25200 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 25201 • The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is  
25202 restored.
- 25203 • The `[EPERM]` error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`.  
25204 This is since its operand is a pathname and applications should be aware that the error  
25205 may not occur for that pathname if the file system does not support  
25206 `_POSIX_CHOWN_RESTRICTED`.

25207 The following new requirements on POSIX implementations derive from alignment with the  
25208 Single UNIX Specification:

- 25209 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
25210 required for conforming implementations of previous POSIX specifications, it was not  
25211 required for UNIX applications.
- 25212 • The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the  
25213 group ID only. A corresponding change is made for group.
- 25214 • The `[ELOOP]` mandatory error condition is added.
- 25215 • The `[EIO]` and `[EINTR]` optional error conditions are added.
- 25216 • A second `[ENAMETOOLONG]` is added as an optional error condition.

25217 The following changes were made to align with the IEEE P1003.1a draft standard:

- 25218 • Clarification is added that the `S_ISUID` and `S_ISGID` bits do not need to be cleared when  
25219 the process has appropriate privileges.
- 25220 • The `[ELOOP]` optional error condition is added.

25221 **Issue 7**

25222 Austin Group Interpretation 1003.1-2001 #143 is applied.

25223 The `fchownat()` function is added from The Open Group Technical Standard, 2006, Extended API  
25224 Set Part 2.

25225 Changes are made related to support for finegrained timestamps.

25226 Changes are made to allow a directory to be opened for searching.

25227 The `[ENOTDIR]` error condition is clarified to cover the condition where the last component of a  
25228 pathname exists but is not a directory or a symbolic link to a directory.25229 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0053 [461], XSH/TC1-2008/0054 [324],  
25230 XSH/TC1-2008/0055 [278], and XSH/TC1-2008/0056 [278] are applied.25231 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0062 [873], XSH/TC2-2008/0063 [591],  
25232 XSH/TC2-2008/0064 [485], XSH/TC2-2008/0065 [817], and XSH/TC2-2008/0066 [817] are  
25233 applied.

25234 **NAME**

25235 cimag, cimagf, cimagl — complex imaginary functions

25236 **SYNOPSIS**

```
25237 #include <complex.h>
25238 double cimag(double complex z);
25239 float cimagf(float complex z);
25240 long double cimagl(long double complex z);
```

25241 **DESCRIPTION**

25242 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
25243 conflict between the requirements described here and the ISO C standard is unintentional. This  
25244 volume of POSIX.1-2024 defers to the ISO C standard.

25245 These functions shall compute the imaginary part of *z*.25246 **RETURN VALUE**

25247 These functions shall return the imaginary part value (as a real).

25248 **ERRORS**

25249 No errors are defined.

25250 **EXAMPLES**

25251 None.

25252 **APPLICATION USAGE**25253 For a variable *z* of complex type:25254 `z == creal(z) + cimag(z)*I`25255 **RATIONALE**

25256 None.

25257 **FUTURE DIRECTIONS**

25258 None.

25259 **SEE ALSO**25260 [carg\(\)](#), [conj\(\)](#), [cproj\(\)](#), [creal\(\)](#)25261 XBD [<complex.h>](#)25262 **CHANGE HISTORY**

25263 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

25264 **NAME**

25265 clearerr — clear indicators on a stream

25266 **SYNOPSIS**

25267 #include &lt;stdio.h&gt;

25268 void clearerr(FILE \*stream);

25269 **DESCRIPTION**

25270 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
25271 conflict between the requirements described here and the ISO C standard is unintentional. This  
25272 volume of POSIX.1-2024 defers to the ISO C standard.

25273 The `clearerr()` function shall clear the end-of-file and error indicators for the stream to which  
25274 `stream` points.

25275 CX The `clearerr()` function shall not change the setting of `errno` if `stream` is valid.

25276 **RETURN VALUE**25277 The `clearerr()` function shall not return a value.25278 **ERRORS**

25279 No errors are defined.

25280 **EXAMPLES**

25281 None.

25282 **APPLICATION USAGE**

25283 None.

25284 **RATIONALE**

25285 None.

25286 **FUTURE DIRECTIONS**

25287 None.

25288 **SEE ALSO**

25289 XBD &lt;stdio.h&gt;

25290 **CHANGE HISTORY**

25291 First released in Issue 1. Derived from Issue 1 of the SVID.

25292 **Issue 7**

25293 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0057 [401] is applied.

25294 **NAME**

25295 clock — report CPU time used

25296 **SYNOPSIS**

25297 #include &lt;time.h&gt;

25298 clock\_t clock(void);

25299 **DESCRIPTION**

25300 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 25301 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25302 volume of POSIX.1-2024 defers to the ISO C standard.

25303 The *clock()* function shall return the implementation's best approximation to the processor time  
 25304 used by the process since the beginning of an implementation-defined era related only to the  
 25305 process invocation.

25306 **RETURN VALUE**

25307 To determine the time in seconds, the value returned by *clock()* should be divided by the value  
 25308 XSI of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.  
 25309 If the processor time used is not available or its value cannot be represented, the function shall  
 25310 return the value `(clock_t)-1`.

25311 **ERRORS**25312 The *clock()* function shall fail if:25313 CX `[EOVERFLOW]` The processor time used cannot be represented in an object of type `clock_t`.25314 **EXAMPLES**

25315 None.

25316 **APPLICATION USAGE**

25317 In programming environments where `clock_t` is a 32-bit integer type and `CLOCKS_PER_SEC` is  
 25318 one million, *clock()* will start failing in less than 36 minutes of processor time for signed `clock_t`,  
 25319 or 72 minutes for unsigned `clock_t`. Applications intended to be portable to such environments  
 25320 should use *times()* instead (or *clock\_gettime()* with `CLOCK_PROCESS_CPUTIME_ID`, if  
 25321 supported).

25322 In order to measure the time spent in a program, *clock()* should be called at the start of the  
 25323 program and its return value subtracted from the value returned by subsequent calls. The value  
 25324 returned by *clock()* is defined for compatibility across systems that have clocks with different  
 25325 resolutions. The resolution on any particular system need not be to microsecond accuracy.

25326 **RATIONALE**

25327 None.

25328 **FUTURE DIRECTIONS**

25329 None.

25330 **SEE ALSO**

25331 *asctime()*, *clock\_getres()*, *ctime()*, *difftime()*, *futimens()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*,  
 25332 *strptime()*, *time()*, *times()*

25333 XBD `<time.h>`25334 **CHANGE HISTORY**

25335 First released in Issue 1. Derived from Issue 1 of the SVID.

25336 **Issue 7**

25337 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0067 [686] is applied.

25338 **Issue 8**

25339 Austin Group Defect 703 is applied, adding the [Eoverflow] error.

25340 **NAME**25341 clock\_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)25342 **SYNOPSIS**

```
25343 CPT #include <time.h>
25344 int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

25345 **DESCRIPTION**

25346 The *clock\_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process  
 25347 specified by *pid*. If the process described by *pid* exists and the calling process has permission, the  
 25348 clock ID of this clock shall be returned in *clock\_id*.

25349 If *pid* is zero, the *clock\_getcpuclockid()* function shall return the clock ID of the CPU-time clock of  
 25350 the process making the call, in *clock\_id*.

25351 The conditions under which one process has permission to obtain the CPU-time clock ID of  
 25352 other processes are implementation-defined.

25353 **RETURN VALUE**

25354 Upon successful completion, *clock\_getcpuclockid()* shall return zero; otherwise, an error number  
 25355 shall be returned to indicate the error.

25356 **ERRORS**

25357 The *clock\_getcpuclockid()* function shall fail if:

25358 [EPERM] The requesting process does not have permission to access the CPU-time clock  
 25359 for the process.

25360 The *clock\_getcpuclockid()* function may fail if:

25361 [ESRCH] No process can be found corresponding to the process specified by *pid*.

25362 **EXAMPLES**

25363 None.

25364 **APPLICATION USAGE**

25365 The *clock\_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not be  
 25366 provided on all implementations.

25367 **RATIONALE**

25368 None.

25369 **FUTURE DIRECTIONS**

25370 None.

25371 **SEE ALSO**

25372 [clock\\_getres\(\)](#), [timer\\_create\(\)](#)

25373 XBD [<time.h>](#)

25374 **CHANGE HISTORY**

25375 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

25376 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

25377 **NAME**

25378 clock\_getres, clock\_gettime, clock\_settime — clock and timer functions

25379 **SYNOPSIS**

```
25380 CX #include <time.h>
25381 int clock_getres(clockid_t clock_id, struct timespec *res);
25382 int clock_gettime(clockid_t clock_id, struct timespec *tp);
25383 int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

25384 **DESCRIPTION**

25385 The *clock\_getres()* function shall return the resolution of any clock. Clock resolutions are  
 25386 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the  
 25387 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,  
 25388 the clock resolution is not returned. If the *time* argument of *clock\_settime()* is not a multiple of *res*,  
 25389 then the value is truncated to a multiple of *res*.

25390 The *clock\_gettime()* function shall return the current value *tp* for the specified clock, *clock\_id*.

25391 The *clock\_settime()* function shall set the specified clock, *clock\_id*, to the value specified by *tp*.  
 25392 Time values that are between two consecutive non-negative integer multiples of the resolution of  
 25393 the specified clock shall be truncated down to the smaller multiple of the resolution.

25394 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that  
 25395 is meaningful only within a process). All implementations shall support a *clock\_id* of  
 25396 CLOCK\_REALTIME as defined in **<time.h>**. This clock represents the clock measuring real time  
 25397 for the system. For this clock, the values returned by *clock\_gettime()* and specified by  
 25398 *clock\_settime()* represent the amount of time (in seconds and nanoseconds) since the Epoch. An  
 25399 implementation may also support additional clocks. The interpretation of time values for these  
 25400 clocks is unspecified.

25401 If the value of the CLOCK\_REALTIME clock is set via *clock\_settime()*, the new value of the clock  
 25402 shall be used to determine the time of expiration for absolute time services based upon the  
 25403 CLOCK\_REALTIME clock. This applies to the time at which armed absolute timers expire. If the  
 25404 absolute time requested at the invocation of such a time service is before the new value of the  
 25405 clock, the time service shall expire immediately as if the clock had reached the requested time  
 25406 normally.

25407 Setting the value of the CLOCK\_REALTIME clock via *clock\_settime()* shall have no effect on  
 25408 threads that are blocked waiting for a relative time service based upon this clock, including the  
 25409 *nanosleep()* and *thrd\_sleep()* functions; nor on the expiration of relative timers based upon this  
 25410 clock. Consequently, these time services shall expire when the requested relative interval  
 25411 elapses, independently of the new or old value of the clock.

25412 All implementations shall support a *clock\_id* of CLOCK\_MONOTONIC defined in **<time.h>**.  
 25413 This clock represents the monotonic clock for the system. For this clock, the value returned by  
 25414 *clock\_gettime()* represents the amount of time (in seconds and nanoseconds) since an unspecified  
 25415 point in the past (for example, system start-up time, or the Epoch). This point does not change  
 25416 after system start-up time. The value of the CLOCK\_MONOTONIC clock cannot be set via  
 25417 *clock\_settime()*. This function shall fail if it is invoked with a *clock\_id* argument of  
 25418 CLOCK\_MONOTONIC.

25419 The effect of setting a clock via *clock\_settime()* on armed per-process timers associated with a  
 25420 clock other than CLOCK\_REALTIME is implementation-defined.

25421 If the value of the CLOCK\_REALTIME clock is set via *clock\_settime()*, the new value of the clock  
 25422 shall be used to determine the time at which the system shall awaken a thread blocked on an

25423 absolute *clock\_nanosleep()* call based upon the CLOCK\_REALTIME clock. If the absolute time  
 25424 requested at the invocation of such a time service is before the new value of the clock, the call  
 25425 shall return immediately as if the clock had reached the requested time normally.

25426 Setting the value of the CLOCK\_REALTIME clock via *clock\_settime()* shall have no effect on any  
 25427 thread that is blocked on a relative *clock\_nanosleep()* call. Consequently, the call shall return  
 25428 when the requested relative interval elapses, independently of the new or old value of the clock.

25429 Appropriate privileges to set a particular clock are implementation-defined.

25430 CPT If `_POSIX_CPUTIME` is defined, implementations shall support clock ID values obtained by  
 25431 invoking *clock\_getcpuclockid()*, which represent the CPU-time clock of a given process.  
 25432 Implementations shall also support the special `clockid_t` value  
 25433 `CLOCK_PROCESS_CPUTIME_ID`, which represents the CPU-time clock of the calling process  
 25434 when invoking one of the *clock\_\**() or *timer\_\**() functions. For these clock IDs, the values  
 25435 returned by *clock\_gettime()* and specified by *clock\_settime()* represent the amount of execution  
 25436 time of the process associated with the clock. Changing the value of a CPU-time clock via  
 25437 *clock\_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see  
 25438 [Scheduling Policies](#)).

25439 TCT If `_POSIX_THREAD_CPUTIME` is defined, implementations shall support clock ID values  
 25440 obtained by invoking *pthread\_getcpuclockid()*, which represent the CPU-time clock of a given  
 25441 thread. Implementations shall also support the special `clockid_t` value  
 25442 `CLOCK_THREAD_CPUTIME_ID`, which represents the CPU-time clock of the calling thread  
 25443 when invoking one of the *clock\_\**() or *timer\_\**() functions. For these clock IDs, the values  
 25444 returned by *clock\_gettime()* and specified by *clock\_settime()* shall represent the amount of  
 25445 execution time of the thread associated with the clock. Changing the value of a CPU-time clock  
 25446 via *clock\_settime()* shall have no effect on the behavior of the sporadic server scheduling policy  
 25447 (see [Scheduling Policies](#)).

#### 25448 RETURN VALUE

25449 A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that  
 25450 an error occurred, and *errno* shall be set to indicate the error.

#### 25451 ERRORS

25452 The *clock\_getres()*, *clock\_gettime()*, and *clock\_settime()* functions shall fail if:

25453 [EINVAL] The *clock\_id* argument does not specify a known clock.

25454 The *clock\_gettime()* function shall fail if:

25455 [EOVERFLOW] The number of seconds will not fit in an object of type `time_t`.

25456 The *clock\_settime()* function shall fail if:

25457 [EINVAL] The *tp* argument to *clock\_settime()* is outside the range for the given clock ID.

25458 [EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than or  
 25459 equal to 1 000 million.

25460 [EINVAL] The value of the *clock\_id* argument is `CLOCK_MONOTONIC`.

25461 The *clock\_settime()* function may fail if:

25462 [EPERM] The requesting process does not have appropriate privileges to set the  
 25463 specified clock.



25464 **EXAMPLES**

25465 None.

25466 **APPLICATION USAGE**

25467 Note that the absolute value of the monotonic clock is meaningless (because its origin is  
 25468 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the  
 25469 fact that the value of this clock is never set and, therefore, that time intervals measured with this  
 25470 clock will not be affected by calls to *clock\_settime()*.

25471 **RATIONALE**

25472 None.

25473 **FUTURE DIRECTIONS**

25474 None.

25475 **SEE ALSO**

25476 [Scheduling Policies](#) (on page 531), *clock\_getcpuclockid()*, *clock\_nanosleep()*, *ctime()*, *mq\_receive()*,  
 25477 *mq\_send()*, *nanosleep()*, *pthread\_mutex\_clocklock()*, *sem\_clockwait()*, *thrd\_sleep()*, *time()*,  
 25478 *timer\_create()*, *timer\_getovertun()*

25479 XBD <[time.h](#)>25480 **CHANGE HISTORY**

25481 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

25482 **Issue 6**

25483 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 25484 implementation does not support the Timers option.

25485 The APPLICATION USAGE section is added.

25486 The following changes were made to align with the IEEE P1003.1a draft standard:

- 25487 • Clarification is added of the effect of resetting the clock resolution.

25488 CPU-time clocks and the *clock\_getcpuclockid()* function are added for alignment with IEEE Std  
 25489 1003.1d-1999.

25490 The following changes are added for alignment with IEEE Std 1003.1j-2000:

- 25491 • The DESCRIPTION is updated as follows:
  - 25492 — The value returned by *clock\_gettime()* for CLOCK\_MONOTONIC is specified.
  - 25493 — The *clock\_settime()* function failing for CLOCK\_MONOTONIC is specified.
  - 25494 — The effects of *clock\_settime()* on the *clock\_nanosleep()* function with respect to  
 25495 CLOCK\_REALTIME are specified.
- 25496 • An [EINVAL] error is added to the ERRORS section, indicating that *clock\_settime()* fails for  
 25497 CLOCK\_MONOTONIC.
- 25498 • The APPLICATION USAGE section notes that the CLOCK\_MONOTONIC clock need not  
 25499 and shall not be set by *clock\_settime()* since the absolute value of the  
 25500 CLOCK\_MONOTONIC clock is meaningless.
- 25501 • The *clock\_nanosleep()*, *mq\_timedreceive()*, *mq\_timedsend()*, *pthread\_mutex\_timedlock()*,  
 25502 *sem\_timedwait()*, *timer\_create()*, and *timer\_settime()* functions are added to the SEE ALSO  
 25503 section.

25504 **Issue 7**

25505           Functionality relating to the Clock Selection option is moved to the Base.

25506           The *clock\_getres()*, *clock\_gettime()*, and *clock\_settime()* functions are moved from the Timers  
25507           option to the Base.

25508           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0058 [106] is applied.

25509 **Issue 8**

25510           Austin Group Defect 1302 is applied, changing “the *nanosleep()* function” to “the *nanosleep()* and  
25511           *thrd\_sleep()* functions”.

25512           Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

25513 **NAME**

25514 clock\_nanosleep — high resolution sleep with specifiable clock

25515 **SYNOPSIS**

```
25516 CX #include <time.h>
25517 int clock_nanosleep(clockid_t clock_id, int flags,
25518     const struct timespec *rqtp, struct timespec *rmtp);
```

25519 **DESCRIPTION**

25520 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the `clock_nanosleep()` function shall  
 25521 cause the current thread to be suspended from execution until either the time interval specified  
 25522 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to  
 25523 invoke a signal-catching function, or the process is terminated. The clock used to measure the  
 25524 time shall be the clock specified by *clock\_id*.

25525 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the `clock_nanosleep()` function shall  
 25526 cause the current thread to be suspended from execution until either the time value of the clock  
 25527 specified by *clock\_id* reaches the absolute time specified by the *rqtp* argument, or a signal is  
 25528 delivered to the calling thread and its action is to invoke a signal-catching function, or the  
 25529 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or  
 25530 equal to the time value of the specified clock, then `clock_nanosleep()` shall return immediately  
 25531 and the calling process shall not be suspended.

25532 The suspension time caused by this function may be longer than requested because the  
 25533 argument value is rounded up to an integer multiple of the sleep resolution, or because of the  
 25534 scheduling of other activity by the system. But, except for the case of being interrupted by a  
 25535 signal, the suspension time for the relative `clock_nanosleep()` function (that is, with the  
 25536 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as  
 25537 measured by the corresponding clock. The suspension for the absolute `clock_nanosleep()` function  
 25538 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the  
 25539 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being  
 25540 interrupted by a signal.

25541 The use of the `clock_nanosleep()` function shall have no effect on the action or blockage of any  
 25542 signal.

25543 The `clock_nanosleep()` function shall fail if the *clock\_id* argument refers to the CPU-time clock of  
 25544 the calling thread. It is unspecified whether *clock\_id* values of other CPU-time clocks are allowed.

25545 **RETURN VALUE**

25546 If the `clock_nanosleep()` function returns because the requested time has elapsed, its return value  
 25547 shall be zero.

25548 If the `clock_nanosleep()` function returns because it has been interrupted by a signal, it shall  
 25549 return the corresponding error value. For the relative `clock_nanosleep()` function, if the *rmtp*  
 25550 argument is non-NULL, the `timespec` structure referenced by it shall be updated to contain the  
 25551 amount of time remaining in the interval (the requested time minus the time actually slept). The  
 25552 *rqtp* and *rmtp* arguments can point to the same object. If the *rmtp* argument is NULL, the  
 25553 remaining time is not returned. The absolute `clock_nanosleep()` function has no effect on the  
 25554 structure referenced by *rmtp*.

25555 If `clock_nanosleep()` fails, it shall return the corresponding error value.

25556 **ERRORS**25557 The *clock\_nanosleep()* function shall fail if:25558 [EINTR] The *clock\_nanosleep()* function was interrupted by a signal.25559 [EINVAL] The *rqt* argument specified a nanosecond value less than zero or greater than  
25560 or equal to 1 000 million; or the `TIMER_ABSTIME` flag was specified in *flags*  
25561 and the *rqt* argument is outside the range for the clock specified by *clock\_id*;  
25562 or the *clock\_id* argument does not specify a known clock, or specifies the CPU-  
25563 time clock of the calling thread.25564 [ENOTSUP] The *clock\_id* argument specifies a clock for which *clock\_nanosleep()* is not  
25565 supported, such as a CPU-time clock.25566 **EXAMPLES**

25567 None.

25568 **APPLICATION USAGE**25569 Calling *clock\_nanosleep()* with the value `TIMER_ABSTIME` not set in the *flags* argument and with  
25570 a *clock\_id* of `CLOCK_REALTIME` is equivalent to calling *nanosleep()* with the same *rqt* and *rmt*  
25571 arguments.25572 **RATIONALE**25573 The *nanosleep()* function specifies that the system-wide clock `CLOCK_REALTIME` is used to  
25574 measure the elapsed time for this time service. However, with the introduction of the monotonic  
25575 clock `CLOCK_MONOTONIC` a new relative sleep function is needed to allow an application to  
25576 take advantage of the special characteristics of this clock.25577 There are many applications in which a process needs to be suspended and then activated  
25578 multiple times in a periodic way; for example, to poll the status of a non-interrupting device or  
25579 to refresh a display device. For these cases, it is known that precise periodic activation cannot be  
25580 achieved with a relative *sleep()* or *nanosleep()* function call. Suppose, for example, a periodic  
25581 process that is activated at time  $T_0$ , executes for a while, and then wants to suspend itself until  
25582 time  $T_0+T$ , the period being  $T$ . If this process wants to use the *nanosleep()* function, it must first  
25583 call *clock\_gettime()* to get the current time, then calculate the difference between the current time  
25584 and  $T_0+T$  and, finally, call *nanosleep()* using the computed interval. However, the process could  
25585 be preempted by a different process between the two function calls, and in this case the interval  
25586 computed would be wrong; the process would wake up later than desired. This problem would  
25587 not occur with the absolute *clock\_nanosleep()* function, since only one function call would be  
25588 necessary to suspend the process until the desired time. In other cases, however, a relative sleep  
25589 is needed, and that is why both functionalities are required.25590 Although it is possible to implement periodic processes using the timers interface, this  
25591 implementation would require the use of signals, and the reservation of some signal numbers. In  
25592 this regard, the reasons for including an absolute version of the *clock\_nanosleep()* function in  
25593 POSIX.1-2024 are the same as for the inclusion of the relative *nanosleep()*.25594 It is also possible to implement precise periodic processes using *pthread\_cond\_timedwait()*, in  
25595 which an absolute timeout is specified that takes effect if the condition variable involved is never  
25596 signaled. However, the use of this interface is unnatural, and involves performing other  
25597 operations on mutexes and condition variables that imply an unnecessary overhead.  
25598 Furthermore, *pthread\_cond\_timedwait()* is not available in implementations that do not support  
25599 threads.25600 Although the interface of the relative and absolute versions of the new high resolution sleep  
25601 service is the same *clock\_nanosleep()* function, the *rmt* argument is only used in the relative  
25602 sleep. This argument is needed in the relative *clock\_nanosleep()* function to reissue the function

25603 call if it is interrupted by a signal, but it is not needed in the absolute *clock\_nanosleep()* function  
25604 call; if the call is interrupted by a signal, the absolute *clock\_nanosleep()* function can be invoked  
25605 again with the same *rqtp* argument used in the interrupted call.

25606 **FUTURE DIRECTIONS**

25607 None.

25608 **SEE ALSO**

25609 *clock\_getres()*, *nanosleep()*, *pthread\_cond\_clockwait()*, *sleep()*

25610 XBD <**time.h**>

25611 **CHANGE HISTORY**

25612 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

25613 **Issue 7**

25614 The *clock\_nanosleep()* function is moved from the Clock Selection option to the Base.

25615 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0068 [909] is applied.

25616 **NAME**

25617 clock\_settime — clock and timer functions

25618 **SYNOPSIS**

```
25619 CX #include <time.h>  
25620 int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

25621 **DESCRIPTION**

25622 Refer to [clock\\_getres\(\)](#).

25623 **NAME**

25624 clog, clogf, clogl — complex natural logarithm functions

25625 **SYNOPSIS**

```
25626 #include <complex.h>
25627 double complex clog(double complex z);
25628 float complex clogf(float complex z);
25629 long double complex clogl(long double complex z);
```

25630 **DESCRIPTION**

25631 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 25632 conflict between the requirements described here and the ISO C standard is unintentional. This  
 25633 volume of POSIX.1-2024 defers to the ISO C standard.

25634 These functions shall compute the complex natural (base  $e$ ) logarithm of  $z$ , with a branch cut  
 25635 along the negative real axis.

25636 **RETURN VALUE**

25637 These functions shall return the complex natural logarithm value, in the range of a strip  
 25638 mathematically unbounded along the real axis and in the interval  $[-i\pi, +i\pi]$  along the imaginary  
 25639 axis.

25640 MXC  $clog(conj(z))$ ,  $clogf(conjf(z))$ , and  $clogl(conjl(z))$  shall return exactly the same value as  $conj(clog(z))$ ,  
 25641  $conjf(clogf(z))$ , and  $conjl(clogl(z))$ , respectively, including for the special values of  $z$  below.

25642 If  $z$  is  $-0 + i0$ ,  $-\text{Inf} + i\pi$  shall be returned and the divide-by-zero floating-point exception shall be  
 25643 raised.

25644 If  $z$  is  $+0 + i0$ ,  $-\text{Inf} + i0$  shall be returned and the divide-by-zero floating-point exception shall be  
 25645 raised.

25646 If  $z$  is  $x + i\text{Inf}$  where  $x$  is finite,  $+\text{Inf} + i\pi/2$  shall be returned.

25647 If  $z$  is  $x + i\text{NaN}$  where  $x$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
 25648 exception may be raised.

25649 If  $z$  is  $-\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $+\text{Inf} + i\pi$  shall be returned.

25650 If  $z$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $+\text{Inf} + i0$  shall be returned.

25651 If  $z$  is  $-\text{Inf} + i\text{Inf}$ ,  $+\text{Inf} + i3\pi/4$  shall be returned.

25652 If  $z$  is  $+\text{Inf} + i\text{Inf}$ ,  $+\text{Inf} + i\pi/4$  shall be returned.

25653 If  $z$  is  $\pm\text{Inf} + i\text{NaN}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.

25654 If  $z$  is  $\text{NaN} + iy$  where  $y$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
 25655 exception may be raised.

25656 If  $z$  is  $\text{NaN} + i\text{Inf}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.

25657 If  $z$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.

25658 **ERRORS**

25659 No errors are defined.

25660 **EXAMPLES**

25661 None.

25662 **APPLICATION USAGE**

25663 None.

25664 **RATIONALE**

25665 None.

25666 **FUTURE DIRECTIONS**

25667 None.

25668 **SEE ALSO**25669 [cexp\(\)](#)25670 XBD [<complex.h>](#)25671 **CHANGE HISTORY**

25672 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

25673 **Issue 8**25674 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
25675 standard.



25676 **NAME**

25677 close, posix\_close — close a file descriptor

25678 **SYNOPSIS**

```
25679 #include <unistd.h>
25680 int close(int fd);
25681 int posix_close(int fd, int flag);
```

25682 **DESCRIPTION**

25683 The `close()` function shall deallocate the file descriptor indicated by *fd*. To deallocate means to  
 25684 make the file descriptor available for return by subsequent calls to `open()` or other functions that  
 25685 allocate file descriptors. All process-owned file locks that the calling process owns on the file  
 25686 associated with the file descriptor shall be unlocked.

25687 If `close()` is interrupted by a signal that is to be caught, then it is unspecified whether it returns  
 25688 `-1` with *errno* set to `[EINTR]` and *fd* remaining open, or returns `-1` with *errno* set to  
 25689 `[EINPROGRESS]` and *fd* being closed, or returns `0` to indicate successful completion; except  
 25690 that if `POSIX_CLOSE_RESTART` is defined as `0`, then the option of returning `-1` with *errno* set to  
 25691 `[EINTR]` and *fd* remaining open shall not occur. If `close()` returns `-1` with *errno* set to `[EINTR]`,  
 25692 it is unspecified whether *fd* can subsequently be passed to any function except `close()` or  
 25693 `posix_close()` without error. For all other error situations (except for `[EBADF]` where *fd* was  
 25694 invalid), *fd* shall be closed. If *fd* was closed even though the close operation is incomplete,  
 25695 the close operation shall continue asynchronously and the process shall have no further ability  
 25696 to track the completion or final status of the close operation.

25697 When all file descriptors associated with a pipe or FIFO special file are closed, any data  
 25698 remaining in the pipe or FIFO shall be discarded.

25699 When all file descriptors associated with an open file description have been closed, the open file  
 25700 description shall be freed.

25701 If the link count of the file is `0`, when all file descriptors associated with the file are closed, the  
 25702 space occupied by the file shall be freed and the file shall no longer be accessible.

25703 XSI If *fd* refers to the manager side of a pseudo-terminal, and this is the last close, a `SIGHUP`  
 25704 signal shall be sent to the controlling process, if any, for which the subsidiary side of the pseudo-  
 25705 terminal is the controlling terminal. It is unspecified whether closing the manager side of the  
 25706 pseudo-terminal flushes all queued input and output.

25707 When there is an outstanding cancelable asynchronous I/O operation against *fd* when `close()`  
 25708 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes  
 25709 as if the `close()` operation had not yet occurred. All operations that are not canceled shall  
 25710 complete as if the `close()` blocked until the operations completed. The `close()` operation itself  
 25711 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and  
 25712 which I/O operation may be canceled upon `close()`, is implementation-defined.

25713 SHM If a memory mapped file or a shared memory object remains referenced at the last close (that is,  
 25714 a process has it mapped), then the entire contents of the memory object shall persist until the  
 25715 SHM memory object becomes unreferenced. If this is the last close of a memory mapped file or a  
 25716 shared memory object and the close results in the memory object becoming unreferenced, and  
 25717 the memory object has been unlinked, then the memory object shall be removed.

25718 When all file descriptors associated with a socket have been closed, the socket shall be  
 25719 destroyed. If the socket is in connection-mode, and the `SO_LINGER` option is set for the socket  
 25720 with non-zero linger time, and the socket has untransmitted data, then `close()` shall block for up  
 25721 to the current linger interval until all data is transmitted.

25722 The `posix_close()` function shall be equivalent to the `close()` function, except with the  
 25723 modifications based on the `flag` argument as described below. If `flag` is 0, then `posix_close()` shall  
 25724 not return `-1` with `errno` set to `[EINTR]`, which implies that `fildev` will always be closed (except for  
 25725 `[EBADF]`, where `fildev` was invalid). If `flag` includes `POSIX_CLOSE_RESTART` and  
 25726 `POSIX_CLOSE_RESTART` is defined as a non-zero value, and `posix_close()` is interrupted by a  
 25727 signal that is to be caught, then `posix_close()` may return `-1` with `errno` set to `[EINTR]`, in which  
 25728 case `fildev` shall be left open; however, it is unspecified whether `fildev` can subsequently be passed  
 25729 to any function except `close()` or `posix_close()` without error. If `flag` is invalid, `posix_close()` may  
 25730 fail with `errno` set to `[EINVAL]`, but shall otherwise behave as if `flag` had been 0 and close `fildev`.

#### 25731 RETURN VALUE

25732 Upon successful completion, 0 shall be returned; otherwise, `-1` shall be returned and `errno` set to  
 25733 indicate the error.

#### 25734 ERRORS

25735 The `close()` and `posix_close()` functions shall fail if:

25736 `[EBADF]` The `fildev` argument is not a open file descriptor.

25737 `[EINPROGRESS]` The function was interrupted by a signal and `fildev` was closed but the close  
 25738 operation is continuing asynchronously.

25739 The `close()` and `posix_close()` functions may fail if:

25740 `[EINTR]` The function was interrupted by a signal, `POSIX_CLOSE_RESTART` is defined  
 25741 as non-zero, and (in the case of `posix_close()`) the `flag` argument included  
 25742 `POSIX_CLOSE_RESTART`, in which case `fildev` is still open.

25743 `[EIO]` An I/O error occurred while reading from or writing to the file system.

25744 The `posix_close()` function may fail if:

25745 `[EINVAL]` The value of the `flag` argument is invalid.

25746 The `close()` and `posix_close()` functions shall not return an `[EAGAIN]` or `[EWOULDBLOCK]`  
 25747 error. If `POSIX_CLOSE_RESTART` is zero, the `close()` function shall not return an `[EINTR]` error.  
 25748 The `posix_close()` function shall not return an `[EINTR]` error unless `flag` includes a non-zero  
 25749 `POSIX_CLOSE_RESTART`.

#### 25750 EXAMPLES

##### 25751 Reassigning a File Descriptor

25752 The following example closes the file descriptor associated with standard output for the current  
 25753 process, re-assigns standard output to a new file descriptor, and closes the original file descriptor  
 25754 to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard  
 25755 input) is not closed.

```
25756 #include <unistd.h>
25757 ...
25758 int pfd;
25759 ...
25760 close(1);
25761 dup(pfd);
25762 close(pfd);
25763 ...
```

25764 Incidentally, this is exactly what could be achieved using:

```
25765     dup2 (pfd, 1);
25766     close (pfd);
```

### 25767 Closing a File Descriptor

25768 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is  
25769 made to associate that file descriptor with a stream.

```
25770     #include <stdio.h>
25771     #include <unistd.h>
25772     #include <stdlib.h>

25773     #define LOCKFILE "/etc/ptmp"
25774     ...
25775     int pfd;
25776     FILE *fpfd;
25777     ...
25778     if ((fpfd = fdopen (pfd, "w")) == NULL) {
25779         close (pfd);
25780         unlink (LOCKFILE);
25781         exit (1);
25782     }
25783     ...
```

### 25784 APPLICATION USAGE

25785 An application that had used the *stdio* routine *fopen()* to open a file should use the  
25786 corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no  
25787 longer exists, since the integer corresponding to it no longer refers to a file.

25788 Implementations may use file descriptors that must be inherited into child processes for the  
25789 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,  
25790 an application that calls *close()* on an arbitrary integer risks non-conforming behavior, and  
25791 *close()* can only portably be used on file descriptor values that the application has obtained  
25792 through explicit actions, as well as the three file descriptors corresponding to the standard file  
25793 streams. In multi-threaded parent applications, the practice of calling *close()* in a loop after *fork()*  
25794 and before an *exec* call in order to avoid a race condition of leaking an unintended file descriptor  
25795 into a child process, is therefore unsafe, and the race should instead be combatted by opening all  
25796 file descriptors with the *FD\_CLOEXEC* bit set unless the file descriptor is intended to be  
25797 inherited across *exec*.

25798 Usage of *close()* on file descriptors *STDIN\_FILENO*, *STDOUT\_FILENO*, or *STDERR\_FILENO*  
25799 should immediately be followed by an operation to reopen these file descriptors. Unexpected  
25800 behavior will result if any of these file descriptors is left in a closed state (for example, an  
25801 [EBADF] error from *perror()*) or if an unrelated *open()* or similar call later in the application  
25802 accidentally allocates a file to one of these well-known file descriptors. Furthermore, a *close()*  
25803 followed by a reopen operation (e.g., *open()*, *dup()*, etc.) is not atomic; *dup2()* should be used to  
25804 change standard file descriptors.

### 25805 RATIONALE

25806 The use of interruptible device close routines should be discouraged to avoid problems with the  
25807 implicit closes of file descriptors, such as by *exec*, process termination, or *dup2()*. This volume of  
25808 POSIX.1-2024 only intends to permit such behavior by specifying the [EINTR] error condition  
25809 for *close()* and *posix\_close()* with non-zero *POSIX\_CLOSE\_RESTART*, to allow applications a  
25810 portable way to resume waiting for an event associated with the close operation (for example, a  
25811 tape drive rewinding) after receiving an interrupt. This standard also permits implementations

25812 to define `POSIX_CLOSE_RESTART` to 0 if they do not choose to provide a way to restart an  
25813 interrupted close action. Although the file descriptor is left open on `[EINTR]`, it might no longer  
25814 be usable; that is, passing it to any function except `close()` or `posix_close()` might result in an error  
25815 such as `[EIO]`. If an application must guarantee that data will not be lost, it is recommended that  
25816 the application use `fsync()` or `fdatasync()` prior to the close operation, rather than leaving the  
25817 close operation to deal with pending I/O and risk an interrupt.

25818 Earlier versions of this standard left the state of *files* unspecified after errors such as `[EINTR]`  
25819 and `[EIO]`; and implementations differed on whether `close()` left *files* open after `[EINTR]`. This  
25820 was unsatisfactory once threads were introduced, since multi-threaded applications need to  
25821 know whether *files* has been closed. Applications cannot blindly call `close()` again, because if  
25822 *files* was closed by the first call another thread could have been allocated a file descriptor with  
25823 the same value as *files*, which must not be closed by the first thread. On the other hand, the  
25824 alternative of never retrying `close()` would lead to a file descriptor leak in implementations  
25825 where `close()` did not close *files*, although such a leak may be harmless if the process is about to  
25826 exit or the file descriptor is marked `FD_CLOEXEC` and the process is about to be replaced by  
25827 `exec`. This standard introduced `posix_close()` with a *flag* argument that allows a choice between  
25828 the two possible error behaviors, and leaves it unspecified which of the two behaviors is  
25829 implemented by `close()` (although it is guaranteed to be one of the two behaviors of `posix_close()`,  
25830 rather than leaving things completely unspecified as in earlier versions of the standard).

25831 Note that the standard requires that `close()` and `posix_close()` must leave *files* open after `[EINTR]`  
25832 (in the cases where `[EINTR]` is permitted) and must close the file descriptor regardless of all  
25833 other errors (except `[EBADF]`, where *files* is already invalid). In general, portable applications  
25834 should only retry a `close()` after checking for `[EINTR]` (and on implementations where  
25835 `POSIX_CLOSE_RESTART` is defined to be zero, this retry loop will be dead code), and risk a file  
25836 descriptor leak if a retry loop is not attempted. It should also be noted that `[EINTR]` is only  
25837 possible if `close()` can be interrupted; if no signal handlers are installed, then `close()` will not be  
25838 interrupted. Conversely, if a single-threaded application can guarantee that no file descriptors  
25839 are opened or closed in signal handlers, then a retry loop without checking for `[EINTR]` will be  
25840 harmless (since the retry will fail with `[EBADF]`), but guaranteeing that no external libraries  
25841 introduce the use of threading can be difficult. There are additional guarantees for applications  
25842 which will only ever be used on systems where `POSIX_CLOSE_RESTART` is defined as 0. These  
25843 observations should help in determining whether an application needs to have its `close()` calls  
25844 audited for replacement with `posix_close()`.

25845 It should also be noted that the requirement for `posix_close()` with a *flag* of 0 to always close *files*,  
25846 even if an error is reported, is similar to the requirements on `fclose()` to always release the  
25847 stream, even if an error is encountered while flushing data.

25848 Implementations that previously always closed *files* can meet the new requirements by  
25849 translating `[EINTR]` to `[EINPROGRESS]` in `close()`; and may define `POSIX_CLOSE_RESTART` to  
25850 0 rather than having to add restart semantics. On the other hand, implementations that  
25851 previously left *files* open on `[EINTR]` can map that to `posix_close()` with  
25852 `POSIX_CLOSE_RESTART`, and must add the semantics of `posix_close()` when *flag* is 0; one  
25853 possibility is by calling the original `close()` implementation, checking for failure, and on `[EINTR]`,  
25854 using actions similar to `dup2()` to replace the incomplete close operation with another file  
25855 descriptor that can be closed immediately by another call to the original `close()`, all before  
25856 returning to the application. Either way, `close()` should always map to one of the two behaviors  
25857 of `posix_close()`, and implementations are encouraged to keep the behavior of `close()` unchanged  
25858 so as not to break implementation-specific expectations of older applications that were relying  
25859 on behavior not specified by older versions of this standard.

25860 The standard developers considered introducing a thread-local variable that `close()` would set to

25861 indicate whether it had closed *filides* when returning `-1`. However, this was rejected in favor of  
 25862 the simpler solution of tightening `close()` to guarantee that *filides* is closed except for [EINTR],  
 25863 and exposing a choice of whether to expect [EINTR] by adding `posix_close()`. Additionally, while  
 25864 the name `posix_close()` is new to this standard, it is reminiscent of at least one implementation  
 25865 that introduced an alternate system call named `close_nocancel()` in order to allow an application  
 25866 to choose whether restart semantics were desired.

25867 Another consideration was whether implementations might return [EAGAIN] as an extension  
 25868 and whether `close()` should be required to leave the file descriptor open in this case, since  
 25869 [EAGAIN] normally implies an operation should be retried. It seemed very unlikely that any  
 25870 implementation would have a legitimate reason to return [EAGAIN] or [EWOULDBLOCK], and  
 25871 therefore this requirement would mean applications have to include code for an error case that  
 25872 will never be used. Therefore `close()` is now forbidden from returning [EAGAIN] and  
 25873 [EWOULDBLOCK] errors.

25874 Note that the requirement for `close()` on a socket to block for up to the current linger interval is  
 25875 not conditional on the `O_NONBLOCK` setting.

25876 The standard developers rejected a proposal to add `closefrom()` to the standard. Because the  
 25877 standard permits implementations to use inherited file descriptors as a means of providing a  
 25878 conforming environment for the child process, it is not possible to standardize an interface that  
 25879 closes arbitrary file descriptors above a certain value while still guaranteeing a conforming  
 25880 environment.

#### 25881 FUTURE DIRECTIONS

25882 None.

#### 25883 SEE ALSO

25884 `dup()`, `exec`, `exit()`, `fclose()`, `fopen()`, `fork()`, `open()`, `perror()`, `unlink()`

25885 XBD `<unistd.h>`

#### 25886 CHANGE HISTORY

25887 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 25888 Issue 5

25889 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

#### 25890 Issue 6

25891 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of  
 25892 the XSI STREAMS Option Group.

25893 The following new requirements on POSIX implementations derive from alignment with the  
 25894 Single UNIX Specification:

- 25895 • The [EIO] error condition is added as an optional error.
- 25896 • The DESCRIPTION is updated to describe the state of the *filides* file descriptor as  
 25897 unspecified if an I/O error occurs and an [EIO] error condition is returned.

25898 Text referring to sockets is added to the DESCRIPTION.

25899 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 25900 shared memory objects and memory mapped files (and not typed memory objects) are the types  
 25901 of memory objects to which the paragraph on last closes applies.

25902 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded  
 25903 text relating to the manager side of a pseudo-terminal. The reason for the change is that the  
 25904 behavior of pseudo-terminals and regular terminals should be as much alike as possible in this  
 25905 case; the change achieves that and matches historical behavior.

25906 **Issue 7**

25907

Functionality relating to the XSI STREAMS option is marked obsolescent.

25908

25909

Functionality relating to the Asynchronous Input and Output and Memory Mapped Files options is moved to the Base.

25910

25911

25912

Austin Group Interpretation 1003.1-2001 #139 is applied, clarifying that the requirement for *close()* on a socket to block for up to the current linger interval is not conditional on the `O_NONBLOCK` setting.

25913

25914

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0059 [419], XSH/TC1-2008/0060 [149], and XSH/TC1-2008/0061 [149] are applied.

25915

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0069 [555] is applied.

25916 **Issue 8**

25917

25918

Austin Group Defect 529 is applied, adding the *posix\_close()* function and changing requirements for the *close()* function relating to `[EINTR]`.

25919

Austin Group Defect 768 is applied, adding OFD-owned file locks.

25920

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

25921

25922

Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal devices.

25923

25924

Austin Group Defect 1525 is applied, clarifying that a socket is not destroyed until all file descriptors associated with it have been closed.

25925 **NAME**

25926 closedir — close a directory stream

25927 **SYNOPSIS**

25928 #include &lt;dirent.h&gt;

25929 int closedir(DIR \*dirp);

25930 **DESCRIPTION**

25931 The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon  
 25932 return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If there is a  
 25933 file descriptor associated with the stream (whether opened by *opendir()* or *dirfd()*, or passed to  
 25934 *fdopendir()* when creating the stream), that file descriptor shall be closed by *closedir()*.

25935 **RETURN VALUE**

25936 Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*  
 25937 set to indicate the error.

25938 **ERRORS**25939 The *closedir()* function may fail if:25940 [EBADF] The *dirp* argument does not refer to an open directory stream.25941 [EINTR] The *closedir()* function was interrupted by a signal.25942 **EXAMPLES**25943 **Closing a Directory Stream**25944 The following program fragment demonstrates how the *closedir()* function is used.

```

25945 ...
25946     DIR *dir;
25947     struct dirent *dp;
25948 ...
25949     if ((dir = opendir(".")) == NULL) {
25950 ...
25951     }
25952
25953     while ((dp = readdir(dir)) != NULL) {
25954 ...
25955     }
25956     closedir(dir);
25957 ...

```

25957 **APPLICATION USAGE**

25958 None.

25959 **RATIONALE**

25960 None.

25961 **FUTURE DIRECTIONS**

25962 None.

25963 **SEE ALSO**25964 [dirfd\(\)](#), [fdopendir\(\)](#)25965 XBD [<dirent.h>](#)

25966 **CHANGE HISTORY**

25967 First released in Issue 2.

25968 **Issue 6**25969 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.25970 The following new requirements on POSIX implementations derive from alignment with the  
25971 Single UNIX Specification:25972 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
25973 required for conforming implementations of previous POSIX specifications, it was not  
25974 required for UNIX applications.

25975 • The [EINTR] error condition is added as an optional error condition.

25976 **Issue 8**25977 Austin Group Defect 1360 is applied, clarifying that type **DIR** always has the ability to store a  
25978 file descriptor; what is optional is whether one is opened by `opendir()`.



25979 **NAME**

25980 closelog, openlog, setlogmask, syslog — control system log

25981 **SYNOPSIS**

```
25982 XSI #include <syslog.h>
25983 void closelog(void);
25984 void openlog(const char *ident, int logopt, int facility);
25985 int setlogmask(int maskpri);
25986 void syslog(int priority, const char *message, ... /* arguments */);
```

25987 **DESCRIPTION**

25988 The *syslog()* function shall send a message to an implementation-defined logging facility, which  
 25989 may log it in an implementation-defined system log, write it to the system console, forward it to  
 25990 a list of users, or forward it to the logging facility on another host over the network. The logged  
 25991 message shall include a message header and a message body. The message header contains at  
 25992 least a timestamp and a tag string.

25993 The message body is generated from the *message* and following arguments in the same manner  
 25994 as if these were arguments to *printf()*, except that the additional conversion specification *%m*  
 25995 shall be recognized; it shall convert no arguments, shall cause the output of the error message  
 25996 string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument  
 25997 specifications of the "*%n\$*" form. If a complete conversion specification with the *m* conversion  
 25998 specifier character is not just *%m*, the behavior is undefined. A trailing <newline> may be added  
 25999 if needed.

26000 Values of the *priority* argument are formed by OR'ing together a severity-level value and an  
 26001 optional facility value. If no facility value is specified, the current default facility value is used.

26002 Possible values of severity level include:

26003	LOG_EMERG	A panic condition.
26004	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
26005		
26006	LOG_CRIT	Critical conditions, such as hard device errors.
26007	LOG_ERR	Errors.
26008	LOG_WARNING	
26009		Warning messages.
26010	LOG_NOTICE	Conditions that are not error conditions, but that may require special handling.
26011		
26012	LOG_INFO	Informational messages.
26013	LOG_DEBUG	Messages that contain information normally of use only when debugging a program.
26014		

26015 The facility indicates the application or system component generating the message. Possible  
 26016 facility values include:

26017	LOG_USER	Messages generated by arbitrary processes. This is the default facility identifier if none is specified.
26018		
26019	LOG_LOCAL0	Reserved for local use.

26020	LOG_LOCAL1	Reserved for local use.
26021	LOG_LOCAL2	Reserved for local use.
26022	LOG_LOCAL3	Reserved for local use.
26023	LOG_LOCAL4	Reserved for local use.
26024	LOG_LOCAL5	Reserved for local use.
26025	LOG_LOCAL6	Reserved for local use.
26026	LOG_LOCAL7	Reserved for local use.
26027	The <i>openlog()</i> function shall set process attributes that affect subsequent calls to <i>syslog()</i> . The	
26028	<i>ident</i> argument is a pointer to a null-terminated identifier that shall be prepended (without the	
26029	null terminator) to every message. The application shall ensure that the string pointed to by <i>ident</i>	
26030	remains valid during the <i>syslog()</i> calls that will prepend this identifier; however, it is unspecified	
26031	whether changes made to the string will change the identifier prepended by later <i>syslog()</i> calls.	
26032	The <i>logopt</i> argument indicates logging options. Values for <i>logopt</i> are constructed by a bitwise-	
26033	inclusive OR of zero or more of the following:	
26034	LOG_PID	Log the process ID with each message. This is useful for identifying specific
26035		processes.
26036	LOG_CONS	Write messages to the system console if they cannot be sent to the logging
26037		facility. The <i>syslog()</i> function ensures that the process does not acquire the
26038		console as a controlling terminal in the process of writing the message.
26039	LOG_NDELAY	Open the connection to the logging facility immediately. Normally the open is
26040		delayed until the first message is logged. This is useful for programs that need
26041		to manage the order in which file descriptors are allocated.
26042	LOG_ODELAY	Delay open until <i>syslog()</i> is called.
26043	LOG_NOWAIT	Do not wait for child processes that may have been created during the course
26044		of logging the message. This option should be used by processes that enable
26045		notification of child termination using SIGCHLD, since <i>syslog()</i> may
26046		otherwise block waiting for a child whose exit status has already been
26047		collected.
26048	The <i>facility</i> argument encodes a default facility to be assigned to all messages that do not have an	
26049	explicit facility already encoded. The initial default facility is LOG_USER.	
26050	The <i>openlog()</i> and <i>syslog()</i> functions may allocate a file descriptor. It is not necessary to call	
26051	<i>openlog()</i> prior to calling <i>syslog()</i> . If a file descriptor is allocated, the FD_CLOEXEC flag shall be	
26052	set; see <fcntl.h>.	
26053	The <i>closelog()</i> function shall close any open file descriptors allocated by previous calls to	
26054	<i>openlog()</i> or <i>syslog()</i> .	
26055	The <i>setlogmask()</i> function shall set the log priority mask for the current process to <i>maskpri</i> and	
26056	return the previous mask. If the <i>maskpri</i> argument is 0, the current log mask is not modified.	
26057	Calls by the current process to <i>syslog()</i> with a priority not set in <i>maskpri</i> shall be rejected. The	
26058	default log mask allows all priorities to be logged. A call to <i>openlog()</i> is not required prior to	
26059	calling <i>setlogmask()</i> .	
26060	The LOG_MASK( <i>pri</i> ) and LOG_UPTO( <i>pri</i> ) macros can be used to ensure a value or range of	
26061	severity levels is properly encoded for the <i>setlogmask()</i> <i>maskpri</i> argument in a portable manner.	
26062	The masks produced by these macros can be OR'ed or AND'ed with other priority masks (for	

26063 example,  
 26064 LOG\_UPTO (LOG\_WARNING) | LOG\_MASK (LOG\_DEBUG)  
 26065 and  
 26066 LOG\_UPTO (LOG\_DEBUG) & ~ ( (LOG\_MASK (LOG\_NOTICE) | LOG\_MASK (LOG\_INFO) )  
 26067 would produce the same priority mask).  
 26068 Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are  
 26069 defined in the `<syslog.h>` header.

#### 26070 RETURN VALUE

26071 The *setlogmask()* function shall return the previous log priority mask. The *closelog()*, *openlog()*,  
 26072 and *syslog()* functions shall not return a value.

#### 26073 ERRORS

26074 The *openlog()* and *syslog()* functions may fail if:

- 26075 [EMFILE] All file descriptors available to the process are currently open.
- 26076 [ENFILE] The maximum allowable number of files is currently open in the system.

#### 26077 EXAMPLES

##### 26078 Using openlog()

26079 The following example causes subsequent calls to *syslog()* to log the process ID with each  
 26080 message, and to write messages to the system console if they cannot be sent to the logging  
 26081 facility.

```
26082 #include <syslog.h>
26083 char *ident = "Process demo";
26084 int logopt = LOG_PID | LOG_CONS;
26085 int facility = LOG_USER;
26086 ...
26087 openlog(ident, logopt, facility);
```

##### 26088 Using setlogmask()

26089 The following example causes subsequent calls to *syslog()* to accept error messages, and to reject  
 26090 all other messages.

```
26091 #include <syslog.h>
26092 int result;
26093 int mask = LOG_MASK (LOG_ERR);
26094 ...
26095 result = setlogmask(mask);
```

**26096 Using syslog**

26097 The following example sends the message "This is a message" to the default logging  
26098 facility, marking the message as an error message generated by random processes.

```
26099 #include <syslog.h>
26100 char *message = "This is a message";
26101 int priority = LOG_ERR | LOG_USER;
26102 ...
26103 syslog(priority, message);
```

**26104 APPLICATION USAGE**

26105 None.

**26106 RATIONALE**

26107 None.

**26108 FUTURE DIRECTIONS**

26109 None.

**26110 SEE ALSO**

26111 [fprintf\(\)](#)

26112 XBD [<fcntl.h>](#), [<syslog.h>](#)

**26113 CHANGE HISTORY**

26114 First released in Issue 4, Version 2.

**26115 Issue 5**

26116 Moved from X/OPEN UNIX extension to BASE.

**26117 Issue 6**

26118 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES  
26119 section.

**26120 Issue 8**

26121 Austin Group Defect 368 is applied, adding a requirement for FD\_CLOEXEC to be set if a file  
26122 descriptor is allocated, and adding the [EMFILE] and [ENFILE] errors.

26123 Austin Group Defect 1033 is applied, adding the LOG\_UPTO macro.

26124 Austin Group Defect 1244 is applied, clarifying the handling of the *ident* argument.

26125 **NAME**

26126 cnd\_broadcast, cnd\_signal — broadcast or signal a condition

26127 **SYNOPSIS**

```
26128 #include <threads.h>
26129 int cnd_broadcast(cnd_t *cond);
26130 int cnd_signal(cnd_t *cond);
```

26131 **DESCRIPTION**

26132 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26133 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26134 volume of POSIX.1-2024 defers to the ISO C standard.

26135 CX The *cnd\_broadcast()* function shall, as a single atomic operation, determine which threads, if  
 26136 any, are blocked on the condition variable pointed to by *cond* and unblock all of these threads.

26137 CX The *cnd\_signal()* function shall, as a single atomic operation, determine which threads, if any,  
 26138 are blocked on the condition variable pointed to by *cond* and unblock at least one of these  
 26139 threads.

26140 If these functions determine that there are no threads blocked on the condition variable pointed  
 26141 to by *cond*, they shall have no effect and shall return *thrd\_success*.

26142 CX If more than one thread is blocked on a condition variable, the scheduling policy shall determine  
 26143 the order in which threads are unblocked. When each thread unblocked as a result of a  
 26144 *cnd\_broadcast()* or *cnd\_signal()* returns from its call to *cnd\_wait()* or *cnd\_timedwait()*, the thread  
 26145 shall own the mutex with which it called *cnd\_wait()* or *cnd\_timedwait()*. The thread(s) that are  
 26146 unblocked shall contend for the mutex according to the scheduling policy (if applicable), and as  
 26147 if each had called *mtx\_lock()*.

26148 The *cnd\_broadcast()* and *cnd\_signal()* functions can be called by a thread whether or not it  
 26149 currently owns the mutex that threads calling *cnd\_wait()* or *cnd\_timedwait()* have associated  
 26150 with the condition variable during their waits; however, if predictable scheduling behavior is  
 26151 required, then that mutex shall be locked by the thread calling *cnd\_broadcast()* or *cnd\_signal()*.

26152 These functions shall not be affected if the calling thread executes a signal handler during the  
 26153 call.

26154 The behavior is undefined if the value specified by the *cond* argument to *cnd\_broadcast()* or  
 26155 *cnd\_signal()* does not refer to an initialized condition variable.

26156 **RETURN VALUE**

26157 These functions shall return *thrd\_success* on success, or *thrd\_error* if the request could  
 26158 not be honored.

26159 **ERRORS**

26160 No errors are defined.

26161 **EXAMPLES**

26162 None.

26163 **APPLICATION USAGE**

26164 See the APPLICATION USAGE section for *pthread\_cond\_broadcast()*, substituting *cnd\_broadcast()*  
 26165 for *pthread\_cond\_broadcast()* and *cnd\_signal()* for *pthread\_cond\_signal()*.

**26166 RATIONALE**

26167 As for *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()*, spurious wakeups may occur with  
26168 *cnd\_broadcast()* and *cnd\_signal()*, necessitating that applications code a predicate-testing-loop  
26169 around the condition wait. (See the RATIONALE section for *pthread\_cond\_broadcast()*.)

26170 These functions are not affected by signal handlers for the reasons stated in XRAT [Section B.2.3](#)  
26171 (on page 3742).

**26172 FUTURE DIRECTIONS**

26173 None.

**26174 SEE ALSO**

26175 [\*cnd\\_destroy\(\)\*](#), [\*cnd\\_timedwait\(\)\*](#), [\*pthread\\_cond\\_broadcast\(\)\*](#)

26176 [XBD Section 4.15.2](#), [<threads.h>](#)

**26177 CHANGE HISTORY**

26178 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

26179 **NAME**

26180 cnd\_destroy, cnd\_init — destroy and initialize condition variables

26181 **SYNOPSIS**

```
26182 #include <threads.h>
26183 void cnd_destroy(cnd_t *cond);
26184 int cnd_init(cnd_t *cond);
```

26185 **DESCRIPTION**

26186 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26187 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26188 volume of POSIX.1-2024 defers to the ISO C standard.

26189 The *cnd\_destroy()* function shall release all resources used by the condition variable pointed to  
 26190 by *cond*. It shall be safe to destroy an initialized condition variable upon which no threads are  
 26191 currently blocked. Attempting to destroy a condition variable upon which other threads are  
 26192 currently blocked results in undefined behavior. A destroyed condition variable object can be  
 26193 reinitialized using *cnd\_init()*; the results of otherwise referencing the object after it has been  
 26194 destroyed are undefined. The behavior is undefined if the value specified by the *cond* argument  
 26195 to *cnd\_destroy()* does not refer to an initialized condition variable.

26196 The *cnd\_init()* function shall initialize a condition variable. If it succeeds it shall set the variable  
 26197 pointed to by *cond* to a value that uniquely identifies the newly initialized condition variable.  
 26198 Attempting to initialize an already initialized condition variable results in undefined behavior. A  
 26199 thread that calls *cnd\_wait()* on a newly initialized condition variable shall block.

26200 CX See [Section 2.9.9](#) (on page 548) for further requirements.

26201 These functions shall not be affected if the calling thread executes a signal handler during the  
 26202 call.

26203 **RETURN VALUE**

26204 The *cnd\_destroy()* function shall not return a value.

26205 The *cnd\_init()* function shall return *thrd\_success* on success, or *thrd\_nomem* if no memory  
 26206 could be allocated for the newly created condition variable, or *thrd\_error* if the request could  
 26207 not be honored.

26208 **ERRORS**

26209 See RETURN VALUE.

26210 **EXAMPLES**

26211 None.

26212 **APPLICATION USAGE**

26213 None.

26214 **RATIONALE**

26215 These functions are not affected by signal handlers for the reasons stated in XRAT [Section B.2.3](#)  
 26216 (on page 3742).

26217 **FUTURE DIRECTIONS**

26218 None.

26219 **SEE ALSO**

26220 [cnd\\_broadcast\(\)](#), [cnd\\_timedwait\(\)](#)

26221 XBD [<threads.h>](#)

26222 **CHANGE HISTORY**

26223 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.



26224 **NAME**

26225 cnd\_timedwait, cnd\_wait — wait on a condition

26226 **SYNOPSIS**

```
26227 #include <threads.h>
26228 int cnd_timedwait(cnd_t * restrict cond, mtx_t * restrict mtx,
26229                 const struct timespec * restrict ts);
26230 int cnd_wait(cnd_t *cond, mtx_t *mtx);
```

26231 **DESCRIPTION**

26232 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26233 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26234 volume of POSIX.1-2024 defers to the ISO C standard.

26235 The *cnd\_timedwait()* function shall atomically unlock the mutex pointed to by *mtx* and block  
 26236 until the condition variable pointed to by *cond* is signaled by a call to *cnd\_signal()* or to  
 26237 *cnd\_broadcast()*, or until after the TIME\_UTC-based calendar time pointed to by *ts*, or until it is  
 26238 unblocked due to an unspecified reason.

26239 The *cnd\_wait()* function shall atomically unlock the mutex pointed to by *mtx* and block until the  
 26240 condition variable pointed to by *cond* is signaled by a call to *cnd\_signal()* or to *cnd\_broadcast()*, or  
 26241 until it is unblocked due to an unspecified reason.

26242 CX Atomically here means “atomically with respect to access by another thread to the mutex and  
 26243 then the condition variable”. That is, if another thread is able to acquire the mutex after the  
 26244 about-to-block thread has released it, then a subsequent call to *cnd\_broadcast()* or *cnd\_signal()* in  
 26245 that thread shall behave as if it were issued after the about-to-block thread has blocked.

26246 When the calling thread becomes unblocked, these functions shall lock the mutex pointed to by  
 26247 *mtx* before they return. The application shall ensure that the mutex pointed to by *mtx* is locked  
 26248 by the calling thread before it calls these functions.

26249 When using condition variables there is always a Boolean predicate involving shared variables  
 26250 associated with each condition wait that is true if the thread should proceed. Spurious wakeups  
 26251 from the *cnd\_timedwait()* and *cnd\_wait()* functions may occur. Since the return from  
 26252 *cnd\_timedwait()* or *cnd\_wait()* does not imply anything about the value of this predicate, the  
 26253 predicate should be re-evaluated upon such return.

26254 When a thread waits on a condition variable, having specified a particular mutex to either the  
 26255 *cnd\_timedwait()* or the *cnd\_wait()* operation, a dynamic binding is formed between that mutex  
 26256 and condition variable that remains in effect as long as at least one thread is blocked on the  
 26257 condition variable. During this time, the effect of an attempt by any thread to wait on that  
 26258 condition variable using a different mutex is undefined. Once all waiting threads have been  
 26259 unblocked (as by the *cnd\_broadcast()* operation), the next wait operation on that condition  
 26260 variable shall form a new dynamic binding with the mutex specified by that wait operation.  
 26261 Even though the dynamic binding between condition variable and mutex might be removed or  
 26262 replaced between the time a thread is unblocked from a wait on the condition variable and the  
 26263 time that it returns to the caller or begins cancellation cleanup, the unblocked thread shall  
 26264 always re-acquire the mutex specified in the condition wait operation call from which it is  
 26265 returning.

26266 CX A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a  
 26267 thread is set to PTHREAD\_CANCEL\_DEFERRED, a side-effect of acting upon a cancellation  
 26268 request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first  
 26269 cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up  
 26270 to the point of returning from the call to *cnd\_timedwait()* or *cnd\_wait()*, but at that point notices

26271 the cancellation request and instead of returning to the caller of *cnd\_timedwait()* or *cnd\_wait()*,  
 26272 starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

26273 A thread that has been unblocked because it has been canceled while blocked in a call to  
 26274 *cnd\_timedwait()* or *cnd\_wait()* shall not consume any condition signal that may be directed  
 26275 concurrently at the condition variable if there are other threads blocked on the condition  
 26276 variable.

26277 When *cnd\_timedwait()* times out, it shall nonetheless release and re-acquire the mutex referenced  
 26278 by *mtx*, and may consume a condition signal directed concurrently at the condition variable.

26279 CX These functions shall not be affected if the calling thread executes a signal handler during the  
 26280 call, except that if a signal is delivered to a thread waiting for a condition variable, upon return  
 26281 from the signal handler either the thread shall resume waiting for the condition variable as if it  
 26282 was not interrupted, or it shall return *thrd\_success* due to spurious wakeup.

26283 The behavior is undefined if the value specified by the *cond* or *mtx* argument to these functions  
 26284 does not refer to an initialized condition variable or an initialized mutex object, respectively.

#### 26285 RETURN VALUE

26286 The *cnd\_timedwait()* function shall return *thrd\_success* upon success, or *thrd\_timedout* if  
 26287 the time specified in the call was reached without acquiring the requested resource, or  
 26288 *thrd\_error* if the request could not be honored.

26289 The *cnd\_wait()* function shall return *thrd\_success* upon success or *thrd\_error* if the  
 26290 request could not be honored.

#### 26291 ERRORS

26292 See RETURN VALUE.

#### 26293 EXAMPLES

26294 None.

#### 26295 APPLICATION USAGE

26296 None.

#### 26297 RATIONALE

26298 These functions are not affected by signal handlers (except as stated in the DESCRIPTION) for  
 26299 the reasons stated in XRAT Section B.2.3 (on page 3742).

#### 26300 FUTURE DIRECTIONS

26301 None.

#### 26302 SEE ALSO

26303 *cnd\_broadcast()*, *cnd\_destroy()*, *timespec\_get()*

26304 XBD Section 4.15.2, <threads.h>

#### 26305 CHANGE HISTORY

26306 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

26307 **NAME**

26308 confstr — get configurable variables

26309 **SYNOPSIS**

26310 #include &lt;unistd.h&gt;

26311 size\_t confstr(int name, char \*buf, size\_t len);

26312 **DESCRIPTION**26313 The *confstr()* function shall return configuration-defined string values. Its use and purpose are  
26314 similar to *sysconf()*, but it is used where string values rather than numeric values are returned.26315 The *name* argument represents the system variable to be queried. The implementation shall  
26316 support the following name values, defined in <unistd.h>. It may support others:

26317 \_CS\_PATH  
 26318 \_CS\_POSIX\_V8\_ILP32\_OFF32\_CFLAGS  
 26319 \_CS\_POSIX\_V8\_ILP32\_OFF32\_LDFLAGS  
 26320 \_CS\_POSIX\_V8\_ILP32\_OFF32\_LIBS  
 26321 \_CS\_POSIX\_V8\_ILP32\_OFFBIG\_CFLAGS  
 26322 \_CS\_POSIX\_V8\_ILP32\_OFFBIG\_LDFLAGS  
 26323 \_CS\_POSIX\_V8\_ILP32\_OFFBIG\_LIBS  
 26324 \_CS\_POSIX\_V8\_LP64\_OFF64\_CFLAGS  
 26325 \_CS\_POSIX\_V8\_LP64\_OFF64\_LDFLAGS  
 26326 \_CS\_POSIX\_V8\_LP64\_OFF64\_LIBS  
 26327 \_CS\_POSIX\_V8\_LPBIG\_OFFBIG\_CFLAGS  
 26328 \_CS\_POSIX\_V8\_LPBIG\_OFFBIG\_LDFLAGS  
 26329 \_CS\_POSIX\_V8\_LPBIG\_OFFBIG\_LIBS  
 26330 \_CS\_POSIX\_V8\_THREADS\_CFLAGS  
 26331 \_CS\_POSIX\_V8\_THREADS\_LDFLAGS  
 26332 \_CS\_POSIX\_V8\_WIDTH\_RESTRICTED\_ENVS  
 26333 \_CS\_V8\_ENV  
 26334 OB \_CS\_POSIX\_V7\_ILP32\_OFF32\_CFLAGS  
 26335 \_CS\_POSIX\_V7\_ILP32\_OFF32\_LDFLAGS  
 26336 \_CS\_POSIX\_V7\_ILP32\_OFF32\_LIBS  
 26337 \_CS\_POSIX\_V7\_ILP32\_OFFBIG\_CFLAGS  
 26338 \_CS\_POSIX\_V7\_ILP32\_OFFBIG\_LDFLAGS  
 26339 \_CS\_POSIX\_V7\_ILP32\_OFFBIG\_LIBS  
 26340 \_CS\_POSIX\_V7\_LP64\_OFF64\_CFLAGS  
 26341 \_CS\_POSIX\_V7\_LP64\_OFF64\_LDFLAGS  
 26342 \_CS\_POSIX\_V7\_LP64\_OFF64\_LIBS  
 26343 \_CS\_POSIX\_V7\_LPBIG\_OFFBIG\_CFLAGS  
 26344 \_CS\_POSIX\_V7\_LPBIG\_OFFBIG\_LDFLAGS  
 26345 \_CS\_POSIX\_V7\_LPBIG\_OFFBIG\_LIBS  
 26346 \_CS\_POSIX\_V7\_THREADS\_CFLAGS  
 26347 \_CS\_POSIX\_V7\_THREADS\_LDFLAGS  
 26348 \_CS\_POSIX\_V7\_WIDTH\_RESTRICTED\_ENVS  
 26349 \_CS\_V7\_ENV

26350 If *len* is not 0, and if *name* has a configuration-defined value, *confstr()* shall copy that value into  
 26351 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,  
 26352 including the terminating null, then *confstr()* shall truncate the string to *len*-1 bytes and null-  
 26353 terminate the result. The application can detect that the string was truncated by comparing the  
 26354 value returned by *confstr()* with *len*.

26355 If *len* is 0 and *buf* is a null pointer, then *confstr()* shall still return the integer value as defined  
 26356 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is  
 26357 unspecified.

26358 After a call to:

```
26359 confstr(_CS_V8_ENV, buf, sizeof(buf))
```

26360 the string stored in *buf* shall contain a <space>-separated list of the variable=value environment  
 26361 variable pairs an implementation requires as part of specifying a conforming environment, as  
 26362 described in the implementations' conformance documentation.

26363 If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

```
26364 confstr(_CS_PATH, buf, sizeof(buf))
```

26365 can be used as a value of the *PATH* environment variable that accesses all of the standard  
 26366 utilities of POSIX.1-2024, that are provided in a manner accessible via the *exec* family of  
 26367 functions, if the return value is less than or equal to *sizeof(buf)*.

#### 26368 RETURN VALUE

26369 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be  
 26370 needed to hold the entire configuration-defined value including the terminating null. If this  
 26371 return value is greater than *len*, the string returned in *buf* is truncated.

26372 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

26373 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*  
 26374 unchanged.

#### 26375 ERRORS

26376 The *confstr()* function shall fail if:

26377 [EINVAL] The value of the *name* argument is invalid.

#### 26378 EXAMPLES

26379 None.

#### 26380 APPLICATION USAGE

26381 An application can distinguish between an invalid *name* parameter value and one that  
 26382 corresponds to a configurable variable that has no configuration-defined value by checking if  
 26383 *errno* is modified. This mirrors the behavior of *sysconf()*.

26384 The original need for this function was to provide a way of finding the configuration-defined  
 26385 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to  
 26386 include directories that could contain utilities replacing the standard utilities in the Shell and  
 26387 Utilities volume of POSIX.1-2024, applications need a way to determine the system-supplied  
 26388 *PATH* environment variable value that contains the correct search path for the standard utilities.

26389 An application could use:

```
26390 confstr(name, (char *)NULL, (size_t)0)
```

26391 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to  
 26392 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,  
 26393 static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use  
 26394 *malloc()* to allocate a larger buffer if it finds that this is too small.

26395 **RATIONALE**

26396 Application developers can normally determine any configuration variable by means of reading  
26397 from the stream opened by a call to:

```
26398 popen("command -p getconf variable", "r");
```

26399 The *confstr()* function with a *name* argument of `_CS_PATH` returns a string that can be used as a  
26400 *PATH* environment variable setting that will reference the standard shell and utilities as  
26401 described in the Shell and Utilities volume of POSIX.1-2024.

26402 The *confstr()* function copies the returned string into a buffer supplied by the application instead  
26403 of returning a pointer to a string. This allows a cleaner function in some implementations (such  
26404 as those with lightweight threads) and resolves questions about when the application must copy  
26405 the string returned.

26406 **FUTURE DIRECTIONS**

26407 None.

26408 **SEE ALSO**

26409 *exec*, *fpathconf()*, *sysconf()*

26410 XBD [<unistd.h>](#)

26411 XCU *c17*

26412 **CHANGE HISTORY**

26413 First released in Issue 4. Derived from the ISO POSIX-2 standard.

26414 **Issue 5**

26415 A table indicating the permissible values of *name* is added to the DESCRIPTION. All those  
26416 marked EX are new in this version.

26417 **Issue 6**

26418 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the  
26419 size of the buffer now explicitly states that this includes the terminating null.

26420 The following new requirements on POSIX implementations derive from alignment with the  
26421 Single UNIX Specification:

- 26422 • The DESCRIPTION is updated with new arguments which can be used to determine  
26423 configuration strings for C compiler flags, linker/loader flags, and libraries for each  
26424 different supported programming environment. This is a change to support data size  
26425 neutrality.

26426 The following changes were made to align with the IEEE P1003.1a draft standard:

- 26427 • The DESCRIPTION is updated to include text describing how `_CS_PATH` can be used to  
26428 obtain a *PATH* to access the standard utilities.

26429 The macros associated with the *c89* programming models are marked LEGACY and new  
26430 equivalent macros associated with *c99* are introduced.

26431 **Issue 7**

26432 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the `_CS_V7_ENV` variable.

26433 Austin Group Interpretations 1003.1-2001 #166 is applied to permit an additional compiler flag  
26434 to enable threads.

26435 The V6 variables for the supported programming environments are marked obsolescent.

26436 The variables for the supported programming environments are updated to be V7.

- 26437 The LEGACY variables and obsolescent values are removed.
- 26438 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0070 [810] and XSH/TC2-2008/0071  
26439 [911] are applied.
- 26440 **Issue 8**
- 26441 Austin Group Defect 1330 is applied, changing ``\_V7\_'' to ``\_V8\_'' and ``\_V6\_'' to ``\_V7\_''.

26442 **NAME**

26443 conj, conjf, conjl — complex conjugate functions

26444 **SYNOPSIS**

26445 #include &lt;complex.h&gt;

26446 double complex conj(double complex z);

26447 float complex conjf(float complex z);

26448 long double complex conjl(long double complex z);

26449 **DESCRIPTION**

26450 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
26451 conflict between the requirements described here and the ISO C standard is unintentional. This  
26452 volume of POSIX.1-2024 defers to the ISO C standard.

26453 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary  
26454 part.

26455 **RETURN VALUE**

26456 These functions return the complex conjugate value.

26457 **ERRORS**

26458 No errors are defined.

26459 **EXAMPLES**

26460 None.

26461 **APPLICATION USAGE**

26462 None.

26463 **RATIONALE**

26464 None.

26465 **FUTURE DIRECTIONS**

26466 None.

26467 **SEE ALSO**26468 [carg\(\)](#), [cimag\(\)](#), [cproj\(\)](#), [creal\(\)](#)26469 XBD [<complex.h>](#)26470 **CHANGE HISTORY**

26471 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26472 **NAME**

26473 connect — connect a socket

26474 **SYNOPSIS**

26475 #include &lt;sys/socket.h&gt;

```
26476 int connect(int socket, const struct sockaddr *address,
26477            socklen_t address_len);
```

26478 **DESCRIPTION**

26479 The *connect()* function shall attempt to make a connection on a connection-mode socket or to set  
 26480 or reset the peer address of a connectionless-mode socket. The function takes the following  
 26481 arguments:

26482	<i>socket</i>	Specifies the file descriptor associated with the socket.
26483	<i>address</i>	Points to a <b>sockaddr</b> structure containing the peer address. The length and format of the address depend on the address family of the socket.
26485	<i>address_len</i>	Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>address</i> argument.

26487 If the address family of the socket is AF\_UNIX, the application shall ensure that a null terminator after the pathname is included in the *sun\_path* member of *address* as a **sockaddr\_un** structure, and that *address\_len* is at least `offsetof(struct sockaddr_un, sun_path) + 1` plus the length of the pathname.

26491 If the socket has not already been bound to a local address, *connect()* shall bind it to an address which, unless the socket's address family is AF\_UNIX, is an unused local address.

26493 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address, and no connection is made. For SOCK\_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent *recv()* functions. If the *sa\_family* member of *address* is AF\_UNSPEC, the socket's peer address shall be reset. Note that despite no connection being made, the term "connected" is used to describe a connectionless-mode socket for which a peer address has been set.

26499 If the initiating socket is connection-mode, then *connect()* shall attempt to establish a connection to the address specified by the *address* argument. If the connection cannot be established immediately and O\_NONBLOCK is not set for the file descriptor for the socket, *connect()* shall block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, *connect()* shall fail and the connection attempt shall be aborted. If *connect()* is interrupted by a signal that is caught while blocked waiting to establish a connection, *connect()* shall fail and set *errno* to [EINTR], but the connection request shall not be aborted, and the connection shall be established asynchronously.

26507 If the connection cannot be established immediately and O\_NONBLOCK is set for the file descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection request shall not be aborted, and the connection shall be established asynchronously. Subsequent calls to *connect()* for the same socket, before the connection is established, shall fail and set *errno* to [EALREADY].

26512 When the connection has been established asynchronously, *pselect()*, *select()*, *poll()*, and *ppoll()* shall indicate that the file descriptor for the socket is ready for writing.

26514 The socket in use may require the process to have appropriate privileges to use the *connect()* function.



26516 **RETURN VALUE**

26517 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*  
 26518 set to indicate the error.

26519 **ERRORS**

26520 The *connect()* function shall fail if:

26521 [EADDRNOTAVAIL]

26522 The specified address is not available from the local machine.

26523 [EAFNOSUPPORT]

26524 The specified address is not a valid address for the address family of the  
 26525 specified socket.

26526 [EALREADY] A connection request is already in progress for the specified socket.

26527 [EBADF] The *socket* argument is not a valid file descriptor.

26528 [ECONNREFUSED]

26529 The target address was not listening for connections or refused the connection  
 26530 request.

26531 [EINPROGRESS] O\_NONBLOCK is set for the file descriptor for the socket and the connection  
 26532 cannot be immediately established; the connection shall be established  
 26533 asynchronously.

26534 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal  
 26535 that was caught; the connection shall be established asynchronously.

26536 [EISCONN] The specified socket is connection-mode and is already connected.

26537 [ENETUNREACH]

26538 No route to the network is present.

26539 [ENOTSOCK] The *socket* argument does not refer to a socket.

26540 [EPROTOTYPE] The specified address has a different type than the socket bound to the  
 26541 specified peer address.

26542 [ETIMEDOUT] The attempt to connect timed out before a connection was made.

26543 If the address family of the socket is AF\_UNIX, then *connect()* shall fail if:

26544 [EIO] An I/O error occurred while reading from or writing to the file system.

26545 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname  
 26546 in *address*.

26547 [ENAMETOOLONG]

26548 The length of a component of a pathname is longer than {NAME\_MAX}.

26549 [ENOENT] A component of the pathname does not name an existing file or the pathname  
 26550 is an empty string.

26551 [ENOTDIR]

26552 A component of the path prefix of the pathname in *address* names an existing  
 26553 file that is neither a directory nor a symbolic link to a directory, or the  
 26554 pathname in *address* contains at least one non-*</i>slash*>* character and ends with  
 26555 one or more trailing *</i>slash*>* characters and the last pathname component  
 26556 names an existing file that is neither a directory nor a symbolic link to a  
 directory.**

26557 The *connect()* function may fail if:

26558	[EACCES]	Search permission is denied for a component of the path prefix; or write access to the named socket is denied.
26559		
26560	[EADDRINUSE]	Attempt to establish a connection that uses addresses that are already in use.
26561	[ECONNRESET]	Remote host reset the connection request.
26562	[EHOSTUNREACH]	
26563		The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
26564		
26565	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family; or invalid address family in the <b>sockaddr</b> structure.
26566		
26567	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in <i>address</i> .
26568		
26569	[ENAMETOOLONG]	
26570		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
26571		
26572		
26573	[ENETDOWN]	The local network interface used to reach the destination is down.
26574	[ENOBUFS]	No buffer space is available.
26575	[EOPNOTSUPP]	The socket is listening and cannot be connected.

**EXAMPLES**

26576 None.

**APPLICATION USAGE**

26578 If *connect()* fails, the state of the socket is unspecified. Conforming applications should close the file descriptor and create a new socket before attempting to reconnect.

26581 For AF\_UNIX sockets, some implementations support an extension where *address\_len* does not have to include a null terminator for the pathname stored in *sun\_path*, which in turn allows a pathname to be one byte longer. However, such usage is not portable, and carries a risk of accessing beyond the intended bounds of the pathname length.

**RATIONALE**

26585 None.

**FUTURE DIRECTIONS**

26587 None.

**SEE ALSO**

26590 *accept()*, *bind()*, *close()*, *getsockname()*, *poll()*, *pselect()*, *send()*, *shutdown()*, *socket()*

26591 XBD <[sys/socket.h](#)>

**CHANGE HISTORY**

26592 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

26594 The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

**Issue 7**

26597 Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected sockets.

26599 Austin Group Interpretation 1003.1-2001 #143 is applied.

- 26600 Austin Group Interpretation 1003.1-2001 #188 is applied, changing the method used to reset a peer address for a datagram socket.
- 26601
- 26602 SD5-XSH-ERN-185 is applied.
- 26603 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0062 [324] is applied.
- 26604 **Issue 8**
- 26605 Austin Group Defect 561 is applied, changing the requirements for the *sun\_path* member of the **sockaddr\_un** structure.
- 26606
- 26607 Austin Group Defect 1263 is applied, adding *ppoll()*.

26608 **NAME**

26609 copysign, copysignf, copysignl — number manipulation function

26610 **SYNOPSIS**

```
26611 #include <math.h>
26612 double copysign(double x, double y);
26613 float copysignf(float x, float y);
26614 long double copysignl(long double x, long double y);
```

26615 **DESCRIPTION**

26616 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26617 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26618 volume of POSIX.1-2024 defers to the ISO C standard.

26619 These functions shall produce a value with the magnitude of  $x$  and the sign of  $y$ . On  
 26620 implementations that represent a signed zero but do not treat negative zero consistently in  
 26621 arithmetic operations, these functions regard the sign of zero as positive.

26622 **RETURN VALUE**

26623 Upon successful completion, these functions shall return a value with the magnitude of  $x$  and  
 26624 the sign of  $y$ .

26625 MX The returned value shall be exact and shall be independent of the current rounding direction  
 26626 mode.

26627 **ERRORS**

26628 No errors are defined.

26629 **EXAMPLES**

26630 None.

26631 **APPLICATION USAGE**

26632 None.

26633 **RATIONALE**

26634 None.

26635 **FUTURE DIRECTIONS**

26636 None.

26637 **SEE ALSO**26638 [signbit\(\)](#)26639 XBD [<math.h>](#)26640 **CHANGE HISTORY**

26641 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26642 **Issue 8**

26643 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
 26644 standard.

26645 **NAME**

26646 cos, cosf, cosl — cosine function

26647 **SYNOPSIS**

```
26648 #include <math.h>
26649 double cos(double x);
26650 float cosf(float x);
26651 long double cosl(long double x);
```

26652 **DESCRIPTION**

26653 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26654 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26655 volume of POSIX.1-2024 defers to the ISO C standard.

26656 These functions shall compute the cosine of their argument  $x$ , measured in radians.

26657 An application wishing to check for error situations should set *errno* to zero and call  
 26658 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 26659 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 26660 zero, an error has occurred.

26661 **RETURN VALUE**

26662 Upon successful completion, these functions shall return the cosine of  $x$ .

26663 MX If  $x$  is NaN, a NaN shall be returned.

26664 If  $x$  is  $\pm 0$ , the value 1.0 shall be returned.

26665 If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and a NaN shall be returned.

26666 **ERRORS**

26667 These functions shall fail if:

26668 MX **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ .

26669 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 26670 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 26671 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 26672 shall be raised.

26673 **EXAMPLES**26674 **Taking the Cosine of a 45-Degree Angle**

```
26675 #include <math.h>
26676 ...
26677 double radians = 45 * M_PI / 180;
26678 double result;
26679 ...
26680 result = cos(radians);
```

26681 **APPLICATION USAGE**

26682 These functions may lose accuracy when their argument is near an odd multiple of  $\pi/2$  or is far  
 26683 from 0.

26684 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 26685 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

26686 **RATIONALE**

26687 None.

26688 **FUTURE DIRECTIONS**

26689 None.

26690 **SEE ALSO**26691 *acos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, *tan()*26692 XBD Section 4.23 (on page 109), **<math.h>**26693 **CHANGE HISTORY**

26694 First released in Issue 1. Derived from Issue 1 of the SVID.

26695 **Issue 5**26696 The DESCRIPTION is updated to indicate how an application should check for an error. This  
26697 text was previously published in the APPLICATION USAGE section.26698 **Issue 6**26699 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.26700 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
26701 revised to align with the ISO/IEC 9899:1999 standard.26702 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
26703 marked.26704 **Issue 7**

26705 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0063 [320] is applied.

26706 **NAME**

26707 cosh, coshf, coshl — hyperbolic cosine functions

26708 **SYNOPSIS**

```
26709 #include <math.h>
26710 double cosh(double x);
26711 float coshf(float x);
26712 long double coshl(long double x);
```

26713 **DESCRIPTION**

26714 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26715 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26716 volume of POSIX.1-2024 defers to the ISO C standard.

26717 These functions shall compute the hyperbolic cosine of their argument  $x$ .

26718 An application wishing to check for error situations should set *errno* to zero and call  
 26719 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 26720 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 26721 zero, an error has occurred.

26722 **RETURN VALUE**

26723 Upon successful completion, these functions shall return the hyperbolic cosine of  $x$ .

26724 If the correct value would cause overflow, a range error shall occur and *cosh()*, *coshf()*, and  
 26725 *coshl()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 26726 respectively.

26727 MX If  $x$  is NaN, a NaN shall be returned.

26728 If  $x$  is  $\pm 0$ , the value 1.0 shall be returned.

26729 If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

26730 **ERRORS**

26731 These functions shall fail if:

26732 Range Error The result would cause an overflow.

26733 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 26734 then *errno* shall be set to [ERANGE]. If the integer expression  
 26735 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 26736 floating-point exception shall be raised.

26737 **EXAMPLES**

26738 None.

26739 **APPLICATION USAGE**

26740 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 26741 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

26742 **RATIONALE**

26743 None.

26744 **FUTURE DIRECTIONS**

26745 None.

26746 **SEE ALSO**26747 *acosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sinh()*, *tanh()*26748 XBD Section 4.23 (on page 109), **<math.h>**26749 **CHANGE HISTORY**

26750 First released in Issue 1. Derived from Issue 1 of the SVID.

26751 **Issue 5**26752 The DESCRIPTION is updated to indicate how an application should check for an error. This  
26753 text was previously published in the APPLICATION USAGE section.26754 **Issue 6**26755 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.26756 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
26757 revised to align with the ISO/IEC 9899:1999 standard.26758 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
26759 marked.26760 **Issue 7**

26761 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0072 [630] is applied.



26762 **NAME**

26763           cosl — cosine function

26764 **SYNOPSIS**

26765           #include <math.h>

26766           long double cosl(long double *x*);

26767 **DESCRIPTION**

26768           Refer to *cos()*.

26769 **NAME**

26770 cpow, cpowf, cpowl — complex power functions

26771 **SYNOPSIS**

26772 #include &lt;complex.h&gt;

26773 double complex cpow(double complex *x*, double complex *y*);26774 float complex cpowf(float complex *x*, float complex *y*);26775 long double complex cpowl(long double complex *x*,26776 long double complex *y*);26777 **DESCRIPTION**

26778 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26779 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26780 volume of POSIX.1-2024 defers to the ISO C standard.

26781 These functions shall compute the complex power function  $x^y$ , with a branch cut for the first  
 26782 parameter along the negative real axis.

26783 MXC These functions shall raise floating-point exceptions if appropriate for the calculation of the parts  
 26784 of the result, and may also raise spurious floating-point exceptions.

26785 **RETURN VALUE**

26786 These functions shall return the complex power function value.

26787 **ERRORS**

26788 No errors are defined.

26789 **EXAMPLES**

26790 None.

26791 **APPLICATION USAGE**

26792 None.

26793 **RATIONALE**

26794 Permitting spurious floating-point exceptions allows  $cpow(z, c)$  to be implemented as  
 26795  $cexp(c \log(z))$  without precluding implementations that treat special cases more carefully.

26796 **FUTURE DIRECTIONS**

26797 None.

26798 **SEE ALSO**26799 [cabs\(\)](#), [csqrt\(\)](#)26800 XBD [<complex.h>](#)26801 **CHANGE HISTORY**

26802 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26803 **Issue 8**

26804 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
 26805 standard.

26806 **NAME**

26807 cproj, cprojf, cprojl — complex projection functions

26808 **SYNOPSIS**

```
26809 #include <complex.h>
26810 double complex cproj(double complex z);
26811 float complex cprojf(float complex z);
26812 long double complex cprojl(long double complex z);
```

26813 **DESCRIPTION**

26814 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 26815 conflict between the requirements described here and the ISO C standard is unintentional. This  
 26816 volume of POSIX.1-2024 defers to the ISO C standard.

26817 These functions shall compute a projection of  $z$  onto the Riemann sphere:  $z$  projects to  $z$ , except  
 26818 that all complex infinities (even those with one infinite part and one NaN part) project to  
 26819 positive infinity on the real axis. If  $z$  has an infinite part, then  $cproj(z)$  shall be equivalent to:

```
26820 INFINITY + I * copysign(0.0, cimag(z))
```

26821 **RETURN VALUE**

26822 These functions shall return the value of the projection onto the Riemann sphere.

26823 **ERRORS**

26824 No errors are defined.

26825 **EXAMPLES**

26826 None.

26827 **APPLICATION USAGE**

26828 None.

26829 **RATIONALE**

26830 Two topologies are commonly used in complex mathematics: the complex plane with its  
 26831 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is  
 26832 better suited for transcendental functions, the Riemann sphere for algebraic functions. The  
 26833 complex types with their multiplicity of infinities provide a useful (though imperfect) model for  
 26834 the complex plane. The  $cproj()$  function helps model the Riemann sphere by mapping all  
 26835 infinities to one, and should be used just before any operation, especially comparisons, that  
 26836 might give spurious results for any of the other infinities. Note that a complex value with one  
 26837 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is  
 26838 infinite, the complex value is infinite independent of the value of the other part. For the same  
 26839 reason,  $cabs()$  returns an infinity if its argument has an infinite part and a NaN part.

26840 **FUTURE DIRECTIONS**

26841 None.

26842 **SEE ALSO**26843 *carg()*, *cimag()*, *conj()*, *creal()*26844 XBD [<complex.h>](#)26845 **CHANGE HISTORY**

26846 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26847 **NAME**

26848 creal, crealf, creall — complex real functions

26849 **SYNOPSIS**

26850 #include &lt;complex.h&gt;

26851 double creal(double complex z);

26852 float crealf(float complex z);

26853 long double creall(long double complex z);

26854 **DESCRIPTION**

26855 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
26856 conflict between the requirements described here and the ISO C standard is unintentional. This  
26857 volume of POSIX.1-2024 defers to the ISO C standard.

26858 These functions shall compute the real part of *z*.26859 **RETURN VALUE**

26860 These functions shall return the real part value.

26861 **ERRORS**

26862 No errors are defined.

26863 **EXAMPLES**

26864 None.

26865 **APPLICATION USAGE**26866 For a variable *z* of type **complex**:26867  $z == \text{creal}(z) + \text{cimag}(z) * I$ 26868 **RATIONALE**

26869 None.

26870 **FUTURE DIRECTIONS**

26871 None.

26872 **SEE ALSO**26873 *carg()*, *cimag()*, *conj()*, *cproj()*

26874 XBD &lt;complex.h&gt;

26875 **CHANGE HISTORY**

26876 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

26877 **NAME**

26878 creat — create a new file or rewrite an existing one

26879 **SYNOPSIS**

```
26880 OH #include <sys/stat.h>
26881 #include <fcntl.h>
26882 int creat(const char *path, mode_t mode);
```

26883 **DESCRIPTION**26884 The *creat()* function shall behave as if it is implemented as follows:

```
26885 int creat(const char *path, mode_t mode)
26886 {
26887     return open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);
26888 }
```

26889 **RETURN VALUE**26890 Refer to *open()*.26891 **ERRORS**26892 Refer to *open()*.26893 **EXAMPLES**26894 **Creating a File**

26895 The following example creates the file **/tmp/file** with read and write permissions for the file  
26896 owner and read permission for group and others. The resulting file descriptor is assigned to the  
26897 *fd* variable.

```
26898 #include <fcntl.h>
26899 ...
26900 int fd;
26901 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
26902 char *pathname = "/tmp/file";
26903 ...
26904 fd = creat(pathname, mode);
26905 ...
```

26906 **APPLICATION USAGE**

26907 In multi-threaded applications, the *creat()* function can leak file descriptors into child processes.  
26908 Applications should instead use *open()* with the *O\_CLOEXEC* and *O\_CLOFORK* flags to avoid  
26909 the leak.

26910 **RATIONALE**

26911 The *creat()* function is redundant. Its services are also provided by the *open()* function. It has  
26912 been included primarily for historical purposes since many existing applications depend on it. It  
26913 is best considered a part of the C binding rather than a function that should be provided in other  
26914 languages.

26915 **FUTURE DIRECTIONS**

26916 None.

26917 **SEE ALSO**26918 *mknod()*, *open()*26919 XBD *<fcntl.h>*, *<sys/stat.h>*, *<sys/types.h>*

26920 **CHANGE HISTORY**

26921 First released in Issue 1. Derived from Issue 1 of the SVID.

26922 **Issue 6**

26923 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

26924 The following new requirements on POSIX implementations derive from alignment with the  
26925 Single UNIX Specification:

- 26926 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
26927 required for conforming implementations of previous POSIX specifications, it was not  
26928 required for UNIX applications.

26929 **Issue 7**

26930 SD5-XSH-ERN-186 is applied.

26931 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0064 [291] is applied.

26932 **Issue 8**

26933 Austin Group Defects 411 and 1318 are applied, changing the APPLICATION USAGE section.

26934 **NAME**26935 crypt — password hashing function (**CRYPT**)26936 **SYNOPSIS**

```
26937 XSI      #include <unistd.h>
26938 char *crypt(const char *key, const char *salt);
```

26939 **DESCRIPTION**

26940 The *crypt()* function hashes a password for storage in the user database. The algorithm is  
 26941 implementation-defined.

26942 The *key* argument points to a password to be hashed. The *salt* argument shall be a string of at  
 26943 least two bytes in length not including the null character chosen from the set:

```
26944 a b c d e f g h i j k l m n o p q r s t u v w x y z
26945 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
26946 0 1 2 3 4 5 6 7 8 9 . /
```

26947 The first two bytes of this string may be used to perturb the encoding algorithm.

26948 The return value of *crypt()* points to static data that is overwritten by each call.

26949 The *crypt()* function need not be thread-safe.

26950 **RETURN VALUE**

26951 Upon successful completion, *crypt()* shall return a pointer to the hashed password; the first two  
 26952 bytes of the returned value shall be those of the *salt* argument. Otherwise, it shall return a null  
 26953 pointer and set *errno* to indicate the error.

26954 **ERRORS**

26955 The *crypt()* function shall fail if:

26956 [ENOSYS] The functionality is not supported on this implementation.

26957 **EXAMPLES**26958 **Encoding Passwords**

26959 The following example finds a user database entry matching a particular user name and changes  
 26960 the current password to a new password. The *crypt()* function generates an encoded version of  
 26961 each password. The first call to *crypt()* produces an encoded version of the old password; that  
 26962 encoded password is then compared to the password stored in the user database. The second  
 26963 call to *crypt()* encodes the new password before it is stored.

26964 The *putpwent()* function, used in the following example, is not part of POSIX.1-2024.

```
26965 #include <unistd.h>
26966 #include <pwd.h>
26967 #include <string.h>
26968 #include <stdio.h>
26969 ...
26970 int valid_change;
26971 int pfd; /* Integer for file descriptor returned by open(). */
26972 FILE *fpfd; /* File pointer for use in putpwent(). */
26973 struct passwd *p;
26974 char user[100];
26975 char oldpasswd[100];
26976 char newpasswd[100];
```

```

26977     char savepasswd[100];
26978     ...
26979     valid_change = 0;
26980     while ((p = getpwent()) != NULL) {
26981         /* Change entry if found. */
26982         if (strcmp(p->pw_name, user) == 0) {
26983             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
26984                 strcpy(savepasswd, crypt(newpasswd, user));
26985                 p->pw_passwd = savepasswd;
26986                 valid_change = 1;
26987             }
26988             else {
26989                 fprintf(stderr, "Old password is not valid\n");
26990             }
26991         }
26992         /* Put passwd entry into ptmp. */
26993         putpwent(p, fpfd);
26994     }

```

#### 26995 APPLICATION USAGE

26996 The values returned by this function need not be portable among XSI-conformant systems.

26997 Several implementations offer extensions via characters outside of the set specified for the *salt*  
 26998 argument for specifying alternative algorithms; while not portable, these extensions may offer  
 26999 better security. The use of *crypt()* for anything other than password hashing is not  
 27000 recommended.

#### 27001 RATIONALE

27002 None.

#### 27003 FUTURE DIRECTIONS

27004 None.

#### 27005 SEE ALSO

27006 [encrypt\(\)](#), [setkey\(\)](#)

27007 XBD [<unistd.h>](#)

#### 27008 CHANGE HISTORY

27009 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 27010 Issue 5

27011 Normative text previously in the APPLICATION USAGE section is moved to the  
 27012 DESCRIPTION.

#### 27013 Issue 7

27014 Austin Group Interpretation 1003.1-2001 #156 is applied.

27015 SD5-XSH-ERN-178 is applied.

27016 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0073 [899] is applied.

#### 27017 Issue 8

27018 Austin Group Defect 1192 is applied, clarifying that *crypt()* is intended for password hashing,  
 27019 not for general string encoding.



27020 **NAME**

27021 csin, csinf, csinl — complex sine functions

27022 **SYNOPSIS**

27023 #include &lt;complex.h&gt;

27024 double complex csin(double complex z);

27025 float complex csinf(float complex z);

27026 long double complex csinl(long double complex z);

27027 **DESCRIPTION**

27028 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27029 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27030 volume of POSIX.1-2024 defers to the ISO C standard.

27031 These functions shall compute the complex sine of  $z$ .27032 **RETURN VALUE**

27033 These functions shall return the complex sine value.

27034 MXC  $csin(conj(iz))$ ,  $csinf(conj(iz))$ , and  $csinl(conj(iz))$  shall return exactly the same value as  
 27035  $conj(csin(iz))$ ,  $conj(csinf(iz))$ , and  $conj(csinl(iz))$ , respectively, and  $csin(-iz)$ ,  $csinf(-iz)$ , and  
 27036  $csinl(-iz)$  shall return exactly the same value as  $-csin(iz)$ ,  $-csinf(iz)$ , and  $-csinl(iz)$ , respectively,  
 27037 including for the special values of  $iz$  below.

27038 If  $iz$  is  $+0 + i0$ ,  $-i (+0 + i0)$  shall be returned.

27039 If  $iz$  is  $+0 + iInf$ ,  $-i (\pm 0 + iNaN)$  shall be returned and the invalid floating-point exception shall be  
 27040 raised; the sign of the imaginary part of the result is unspecified.

27041 If  $iz$  is  $+0 + iNaN$ ,  $-i (\pm 0 + iNaN)$  shall be returned; the sign of the imaginary part of the result is  
 27042 unspecified.

27043 If  $iz$  is  $x + iInf$  where  $x$  is positive and finite,  $-i (NaN + iNaN)$  shall be returned and the invalid  
 27044 floating-point exception shall be raised.

27045 If  $iz$  is  $x + iNaN$  where  $x$  is non-zero and finite,  $-i (NaN + iNaN)$  shall be returned and the  
 27046 invalid floating-point exception may be raised.

27047 If  $iz$  is  $+Inf + i0$ ,  $-i (+Inf + i0)$  shall be returned.27048 If  $iz$  is  $+Inf + iy$  where  $y$  is positive and finite,  $-iInf (\cos(y) + i \sin(y))$  shall be returned.

27049 If  $iz$  is  $+Inf + iInf$ ,  $-i (\pm Inf + iNaN)$  shall be returned and the invalid floating-point exception  
 27050 shall be raised; the sign of the imaginary part of the result is unspecified.

27051 If  $iz$  is  $+Inf + iNaN$ ,  $-i (\pm Inf + iNaN)$  shall be returned; the sign of the imaginary part of the  
 27052 result is unspecified.

27053 If  $iz$  is  $NaN + i0$ ,  $-i (NaN + i0)$  shall be returned.

27054 If  $iz$  is  $NaN + iy$  where  $y$  is any non-zero number,  $-i (NaN + iNaN)$  shall be returned and the  
 27055 invalid floating-point exception may be raised.

27056 If  $iz$  is  $NaN + iNaN$ ,  $-i (NaN + iNaN)$  shall be returned.27057 **ERRORS**

27058 No errors are defined.

27059 **EXAMPLES**

27060 None.

27061 **APPLICATION USAGE**

27062 None.

27063 **RATIONALE**

27064 The MXC special cases for *csin()* are derived from those for *csinh()* by applying the formula  
27065  $csin(z) = -i csinh(iz)$ .

27066 **FUTURE DIRECTIONS**

27067 None.

27068 **SEE ALSO**27069 *casin()*, *csinh()*27070 XBD <**complex.h**>27071 **CHANGE HISTORY**

27072 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27073 **Issue 8**

27074 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
27075 standard.

27076 **NAME**

27077           csinh, csinhf, csinhl — complex hyperbolic sine functions

27078 **SYNOPSIS**

```
27079       #include <complex.h>
27080       double complex csinh(double complex z);
27081       float complex csinhf(float complex z);
27082       long double complex csinhl(long double complex z);
```

27083 **DESCRIPTION**

27084 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 27085           conflict between the requirements described here and the ISO C standard is unintentional. This  
 27086           volume of POSIX.1-2024 defers to the ISO C standard.

27087           These functions shall compute the complex hyperbolic sine of  $z$ .27088 **RETURN VALUE**

27089           These functions shall return the complex hyperbolic sine value.

27090 MXC        $csinh(conj(z))$ ,  $csinhf(conjf(z))$ , and  $csinhl(conjl(z))$  shall return exactly the same value as  
 27091            $conj(csinh(z))$ ,  $conjf(csinhf(z))$ , and  $conjl(csinhl(z))$ , respectively, and  $csinh(-z)$ ,  $csinhf(-z)$ , and  
 27092            $csinhl(-z)$  shall return exactly the same value as  $-csinh(z)$ ,  $-csinhf(z)$ , and  $-csinhl(z)$ , respectively,  
 27093           including for the special values of  $z$  below.

27094           If  $z$  is  $+0 + i0$ ,  $+0 + i0$  shall be returned.

27095           If  $z$  is  $+0 + iInf$ ,  $\pm 0 + iNaN$  shall be returned and the invalid floating-point exception shall be  
 27096           raised; the sign of the real part of the result is unspecified.

27097           If  $z$  is  $+0 + iNaN$ ,  $\pm 0 + iNaN$  shall be returned; the sign of the real part of the result is  
 27098           unspecified.

27099           If  $z$  is  $x + iInf$  where  $x$  is positive and finite,  $NaN + iNaN$  shall be returned and the invalid  
 27100           floating-point exception shall be raised.

27101           If  $z$  is  $x + iNaN$  where  $x$  is non-zero and finite,  $NaN + iNaN$  shall be returned and the invalid  
 27102           floating-point exception may be raised.

27103           If  $z$  is  $+Inf + i0$ ,  $+Inf + i0$  shall be returned.27104           If  $z$  is  $+Inf + iy$  where  $y$  is positive and finite,  $+Inf (\cos(y) + i \sin(y))$  shall be returned.

27105           If  $z$  is  $+Inf + iInf$ ,  $\pm Inf + iNaN$  shall be returned and the invalid floating-point exception shall be  
 27106           raised; the sign of the real part of the result is unspecified.

27107           If  $z$  is  $+Inf + iNaN$ ,  $\pm Inf + iNaN$  shall be returned; the sign of the real part of the result is  
 27108           unspecified.

27109           If  $z$  is  $NaN + i0$ ,  $NaN + i0$  shall be returned.

27110           If  $z$  is  $NaN + iy$  where  $y$  is any non-zero number,  $NaN + iNaN$  shall be returned and the invalid  
 27111           floating-point exception may be raised.

27112           If  $z$  is  $NaN + iNaN$ ,  $NaN + iNaN$  shall be returned.27113 **ERRORS**

27114           No errors are defined.

27115 **EXAMPLES**

27116 None.

27117 **APPLICATION USAGE**

27118 None.

27119 **RATIONALE**

27120 None.

27121 **FUTURE DIRECTIONS**

27122 None.

27123 **SEE ALSO**27124 [casinh\(\)](#)27125 XBD [<complex.h>](#)27126 **CHANGE HISTORY**

27127 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27128 **Issue 8**27129 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
27130 standard.

27131 **NAME**

27132           csinl — complex sine functions

27133 **SYNOPSIS**

27134           #include &lt;complex.h&gt;

27135           long double complex csinl(long double complex z);

27136 **DESCRIPTION**27137           Refer to *csin()*.

27138 **NAME**

27139 csqrt, csqrtf, csqrtl — complex square root functions

27140 **SYNOPSIS**

```
27141 #include <complex.h>
27142 double complex csqrt(double complex z);
27143 float complex csqrtf(float complex z);
27144 long double complex csqrtl(long double complex z);
```

27145 **DESCRIPTION**

27146 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27147 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27148 volume of POSIX.1-2024 defers to the ISO C standard.

27149 These functions shall compute the complex square root of  $z$ , with a branch cut along the negative  
 27150 real axis.

27151 **RETURN VALUE**

27152 These functions shall return the complex square root value, in the range of the right half-plane  
 27153 (including the imaginary axis).

27154 MXC *csqrt(conj(z))*, *csqrtf(conjf(z))*, and *csqrtl(conjl(z))* shall return exactly the same value as  
 27155 *conj(csqrt(z))*, *conjf(csqrtf(z))*, and *conjl(csqrtl(z))*, respectively, including for the special values of  
 27156  $z$  below.

27157 If  $z$  is  $\pm 0 + i0$ ,  $+0 + i0$  shall be returned.

27158 If the imaginary part of  $z$  is  $\text{Inf}$ ,  $+\text{Inf} + i\text{Inf}$  shall be returned.

27159 If  $z$  is  $x + i\text{NaN}$  where  $x$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
 27160 exception may be raised.

27161 If  $z$  is  $-\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $+0 + i\text{Inf}$  shall be returned.

27162 If  $z$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $+\text{Inf} + i0$  shall be returned.

27163 If  $z$  is  $-\text{Inf} + i\text{NaN}$ ,  $\text{NaN} \pm i\text{Inf}$  shall be returned; the sign of the imaginary part of the result is  
 27164 unspecified.

27165 If  $z$  is  $+\text{Inf} + i\text{NaN}$ ,  $+\text{Inf} + i\text{NaN}$  shall be returned.

27166 If  $z$  is  $\text{NaN} + iy$  where  $y$  is finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid floating-point  
 27167 exception may be raised.

27168 If  $z$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.

27169 **ERRORS**

27170 No errors are defined.

27171 **EXAMPLES**

27172 None.

27173 **APPLICATION USAGE**

27174 None.

27175 **RATIONALE**

27176 None.

27177 **FUTURE DIRECTIONS**

27178 None.

27179 **SEE ALSO**27180 *fabs()*, *cpow()*27181 XBD **<complex.h>**27182 **CHANGE HISTORY**

27183 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27184 **Issue 8**27185 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
27186 standard.

27187 **NAME**

27188 ctan, ctanf, ctanl — complex tangent functions

27189 **SYNOPSIS**

27190 #include &lt;complex.h&gt;

27191 double complex ctan(double complex z);

27192 float complex ctanf(float complex z);

27193 long double complex ctanl(long double complex z);

27194 **DESCRIPTION**

27195 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27196 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27197 volume of POSIX.1-2024 defers to the ISO C standard.

27198 These functions shall compute the complex tangent of  $z$ .27199 **RETURN VALUE**

27200 These functions shall return the complex tangent value.

27201 MXC  $ctan(conj(iz))$ ,  $ctanf(conjf(iz))$ , and  $ctanl(conjl(iz))$  shall return exactly the same value as  
 27202  $conj(ctan(iz))$ ,  $conjf(ctanf(iz))$ , and  $conjl(ctanl(iz))$ , respectively, and  $ctan(-iz)$ ,  $ctanf(-iz)$ , and  
 27203  $ctanl(-iz)$  shall return exactly the same value as  $-ctan(iz)$ ,  $-ctanf(iz)$ , and  $-ctanl(iz)$ , respectively,  
 27204 including for the special values of  $iz$  below.

27205 If  $iz$  is  $+0 + i0$ ,  $-i (+0 + i0)$  shall be returned.

27206 If  $iz$  is  $0 + i\text{Inf}$ ,  $-i (0 + i\text{NaN})$  shall be returned and the invalid floating-point exception shall be  
 27207 raised.

27208 If  $iz$  is  $x + i\text{Inf}$  where  $x$  is non-zero and finite,  $-i (\text{NaN} + i\text{NaN})$  shall be returned and the invalid  
 27209 floating-point exception shall be raised.

27210 If  $iz$  is  $0 + i\text{NaN}$ ,  $-i (0 + i\text{NaN})$  shall be returned.

27211 If  $iz$  is  $x + i\text{NaN}$  where  $x$  is non-zero and finite,  $-i (\text{NaN} + i\text{NaN})$  shall be returned and the  
 27212 invalid floating-point exception may be raised.

27213 If  $iz$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $-i (1 + i0 \sin(2y))$  shall be returned.27214 If  $iz$  is  $+\text{Inf} + i\text{Inf}$ ,  $-i (1 \pm i0)$  shall be returned; the sign of the real part of the result is unspecified.

27215 If  $iz$  is  $+\text{Inf} + i\text{NaN}$ ,  $-i (1 \pm i0)$  shall be returned; the sign of the real part of the result is  
 27216 unspecified.

27217 If  $iz$  is  $\text{NaN} + i0$ ,  $-i (\text{NaN} + i0)$  shall be returned.

27218 If  $iz$  is  $\text{NaN} + iy$  where  $y$  is any non-zero number,  $-i (\text{NaN} + i\text{NaN})$  shall be returned and the  
 27219 invalid floating-point exception may be raised.

27220 If  $iz$  is  $\text{NaN} + i\text{NaN}$ ,  $-i (\text{NaN} + i\text{NaN})$  shall be returned.27221 **ERRORS**

27222 No errors are defined.



27223 **EXAMPLES**

27224 None.

27225 **APPLICATION USAGE**

27226 None.

27227 **RATIONALE**

27228 The MXC special cases for *ctan()* are derived from those for *ctanh()* by applying the formula  
27229  $ctan(z) = -i ctanh(iz)$ .

27230 **FUTURE DIRECTIONS**

27231 None.

27232 **SEE ALSO**27233 [catan\(\)](#), [ctanh\(\)](#)27234 XBD [<complex.h>](#)27235 **CHANGE HISTORY**

27236 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27237 **Issue 8**

27238 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
27239 standard.

27240 **NAME**

27241 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

27242 **SYNOPSIS**

```
27243 #include <complex.h>
27244 double complex ctanh(double complex z);
27245 float complex ctanhf(float complex z);
27246 long double complex ctanhl(long double complex z);
```

27247 **DESCRIPTION**

27248 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27249 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27250 volume of POSIX.1-2024 defers to the ISO C standard.

27251 These functions shall compute the complex hyperbolic tangent of  $z$ .27252 **RETURN VALUE**

27253 These functions shall return the complex hyperbolic tangent value.

27254 MXC  $ctanh(conj(z))$ ,  $ctanhf(conjf(z))$ , and  $ctanhl(conjl(z))$  shall return exactly the same value as  
 27255  $conj(ctanh(z))$ ,  $conjf(ctanhf(z))$ , and  $conjl(ctanhl(z))$ , respectively, and  $ctanh(-z)$ ,  $ctanhf(-z)$ , and  
 27256  $ctanhl(-z)$  shall return exactly the same value as  $-ctanh(z)$ ,  $-ctanhf(z)$ , and  $-ctanhl(z)$ ,  
 27257 respectively, including for the special values of  $z$  below.

27258 If  $z$  is  $+0 + i0$ ,  $+0 + i0$  shall be returned.27259 If  $z$  is  $0 + i\text{Inf}$ ,  $0 + i\text{NaN}$  shall be returned and the invalid floating-point exception shall be raised.

27260 If  $z$  is  $x + i\text{Inf}$  where  $x$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 27261 floating-point exception shall be raised.

27262 If  $z$  is  $0 + i\text{NaN}$ ,  $0 + i\text{NaN}$  shall be returned.

27263 If  $z$  is  $x + i\text{NaN}$  where  $x$  is non-zero and finite,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 27264 floating-point exception may be raised.

27265 If  $z$  is  $+\text{Inf} + iy$  where  $y$  is positive-signed and finite,  $1 + i0 \sin(2y)$  shall be returned.

27266 If  $z$  is  $+\text{Inf} + i\text{Inf}$ ,  $1 \pm i0$  shall be returned; the sign of the imaginary part of the result is  
 27267 unspecified.

27268 If  $z$  is  $+\text{Inf} + i\text{NaN}$ ,  $1 \pm i0$  shall be returned; the sign of the imaginary part of the result is  
 27269 unspecified.

27270 If  $z$  is  $\text{NaN} + i0$ ,  $\text{NaN} + i0$  shall be returned.

27271 If  $z$  is  $\text{NaN} + iy$  where  $y$  is any non-zero number,  $\text{NaN} + i\text{NaN}$  shall be returned and the invalid  
 27272 floating-point exception may be raised.

27273 If  $z$  is  $\text{NaN} + i\text{NaN}$ ,  $\text{NaN} + i\text{NaN}$  shall be returned.27274 **ERRORS**

27275 No errors are defined.

27276 **EXAMPLES**

27277 None.

27278 **APPLICATION USAGE**

27279 None.

27280 **RATIONALE**

27281 None.

27282 **FUTURE DIRECTIONS**

27283 None.

27284 **SEE ALSO**27285 [catanh\(\)](#)27286 XBD [<complex.h>](#)27287 **CHANGE HISTORY**

27288 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27289 **Issue 8**27290 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
27291 standard.

27292 **NAME**

27293           ctanl — complex tangent functions

27294 **SYNOPSIS**

27295           #include &lt;complex.h&gt;

27296           long double complex ctanl(long double complex z);

27297 **DESCRIPTION**27298           Refer to *ctan()*.

27299 **NAME**

27300 ctermid — generate a pathname for the controlling terminal

27301 **SYNOPSIS**

```
27302 CX #include <stdio.h>
27303 char *ctermid(char *s);
```

27304 **DESCRIPTION**

27305 The *ctermid()* function shall generate a string that, when used as a pathname, refers to the  
 27306 current controlling terminal for the current process. If *ctermid()* returns a pathname, access to the  
 27307 file is not guaranteed.

27308 The *ctermid()* function need not be thread-safe if called with a NULL parameter.

27309 **RETURN VALUE**

27310 If *s* is a null pointer, the string shall be generated in an area that may be static, the address of  
 27311 which shall be returned. The application shall not modify the string returned. The returned  
 27312 pointer might be invalidated or the string content might be overwritten by a subsequent call to  
 27313 *ctermid()*. The returned pointer might also be invalidated if the calling thread is terminated. If *s*  
 27314 is not a null pointer, *s* is assumed to point to a character array of at least `L_ctermid` bytes; the  
 27315 string is placed in this array and the value of *s* shall be returned. The symbolic constant  
 27316 `L_ctermid` is defined in `<stdio.h>`, and shall have a value greater than 0.

27317 The *ctermid()* function shall return an empty string if the pathname that would refer to the  
 27318 controlling terminal cannot be determined, or if the function is unsuccessful.

27319 **ERRORS**

27320 No errors are defined.

27321 **EXAMPLES**27322 **Determining the Controlling Terminal for the Current Process**

27323 The following example returns a pointer to a string that identifies the controlling terminal for the  
 27324 current process. The pathname for the terminal is stored in the array pointed to by the *ptr*  
 27325 argument, which has a size of `L_ctermid` bytes, as indicated by the *term* argument.

```
27326 #include <stdio.h>
27327 ...
27328 char term[L_ctermid];
27329 char *ptr;
27330 ptr = ctermid(term);
```

27331 **APPLICATION USAGE**

27332 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be handed a file  
 27333 descriptor and return a path of the terminal associated with that file descriptor, while *ctermid()*  
 27334 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a  
 27335 pathname.

27336 **RATIONALE**

27337 `L_ctermid` must be defined appropriately for a given implementation and must be greater than  
 27338 zero so that array declarations using it are accepted by the compiler. The value includes the  
 27339 terminating null byte.

27340 Conforming applications that use multiple threads cannot call *ctermid()* with NULL as the  
 27341 parameter. If *s* is not NULL, the *ctermid()* function generates a string that, when used as a

27342 pathname, refers to the current controlling terminal for the current process. If *s* is NULL, the  
27343 return value of *ctermid()* is undefined.

27344 There is no additional burden on the programmer—changing to use a hypothetical thread-safe  
27345 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a  
27346 buffer. Application code should not assume that the returned string is short, as some  
27347 implementations have more than two pathname components before reaching a logical device  
27348 name.

#### 27349 **FUTURE DIRECTIONS**

27350 None.

#### 27351 **SEE ALSO**

27352 [ttyname\(\)](#)

27353 XBD [<stdio.h>](#)

#### 27354 **CHANGE HISTORY**

27355 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 27356 **Issue 5**

27357 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 27358 **Issue 6**

27359 The normative text is updated to avoid use of the term “must” for application requirements.

#### 27360 **Issue 7**

27361 Austin Group Interpretation 1003.1-2001 #148 is applied, updating the RATIONALE.

27362 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0065 [75,428] is applied.

27363 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0074 [656] is applied.

27364 **NAME**

27365 ctime — convert a time value to a date and time string

27366 **SYNOPSIS**

```
27367 OB #include <time.h>
27368 char *ctime(const time_t *clock);
```

27369 **DESCRIPTION**

27370 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 27371 conflict between the requirements described here and the ISO C standard is unintentional. This  
 27372 volume of POSIX.1-2024 defers to the ISO C standard.

27373 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds  
 27374 since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
27375 asctime(localtime(clock))
```

27376 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static  
 27377 objects: a broken-down time structure and an array of **char**. Execution of any of the functions  
 27378 that return a pointer to one of these object types may overwrite the information in any object of  
 27379 the same type pointed to by the value returned from any previous call to any of them.

27380 The *ctime()* function need not be thread-safe; however, *ctime()* shall avoid data races with all  
 27381 functions other than itself, *asctime()*, *gmtime()*, and *localtime()*.

27382 **RETURN VALUE**

27383 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time  
 27384 as an argument.

27385 **ERRORS**

27386 No errors are defined.

27387 **EXAMPLES**

27388 None.

27389 **APPLICATION USAGE**

27390 This function is included only for compatibility with older implementations. It has undefined  
 27391 behavior if the resulting string would be too long, so the use of this function should be  
 27392 discouraged. On implementations that do not detect output string length overflow, it is possible  
 27393 to overflow the output buffer in such a way as to cause applications to fail, or possible system  
 27394 security violations. Also, this function does not support localized date and time formats. To  
 27395 avoid these problems, applications should use *strftime()* to generate strings from broken-down  
 27396 times.

27397 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

27398 Attempts to use *ctime()* for times before the Epoch or for times beyond the year 9999 produce  
 27399 undefined results. Refer to *asctime()* (on page 617).

27400 **RATIONALE**

27401 The standard developers decided to mark the *ctime()* function obsolescent even though it is in  
 27402 the ISO C standard due to the possibility of buffer overflow. The ISO C standard also provides  
 27403 the *strftime()* function which can be used to avoid these problems.

27404 **FUTURE DIRECTIONS**

27405 This function may be removed in a future version, but not until after it has been removed from  
27406 the ISO C standard.

27407 **SEE ALSO**

27408 *asctime()*, *clock()*, *difftime()*, *futimens()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*,  
27409 *time()*

27410 XBD <[time.h](#)>

27411 **CHANGE HISTORY**

27412 First released in Issue 1. Derived from Issue 1 of the SVID.

27413 **Issue 5**

27414 Normative text previously in the APPLICATION USAGE section is moved to the  
27415 DESCRIPTION.

27416 The *ctime\_r()* function is included for alignment with the POSIX Threads Extension.

27417 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.

27418 **Issue 6**

27419 Extensions beyond the ISO C standard are marked.

27420 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

27421 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
27422 its avoidance of possibly using a static data area.

27423 **Issue 7**

27424 Austin Group Interpretation 1003.1-2001 #156 is applied.

27425 SD5-XSH-ERN-25 is applied, updating the APPLICATION USAGE.

27426 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

27427 The *ctime\_r()* function is moved from the Thread-Safe Functions option to the Base.

27428 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0066 [321,428] is applied.

27429 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0075 [664] is applied.

27430 **Issue 8**

27431 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
27432 standard.

27433 Austin Group Defect 1330 is applied, changing the FUTURE DIRECTIONS section.

27434 Austin Group Defect 1376 is applied, removing CX shading from some text derived from the  
27435 ISO C standard and updating it to match the ISO C standard.

27436 Austin Group Defect 1410 is applied, removing the *ctime\_r()* function.



27437 **NAME**

27438 daylight — daylight saving time flag

27439 **SYNOPSIS**

```
27440 XSI #include <time.h>  
27441 extern int daylight;
```

27442 **DESCRIPTION**27443 Refer to [tzset\(\)](#).

27444 **NAME**

27445 dbm\_clearerr, dbm\_close, dbm\_delete, dbm\_error, dbm\_fetch, dbm\_firstkey, dbm\_nextkey,  
27446 dbm\_open, dbm\_store — database functions

27447 **SYNOPSIS**

```
27448 XSI #include <ndbm.h>
27449
27449 int dbm_clearerr(DBM *db);
27450 void dbm_close(DBM *db);
27451 int dbm_delete(DBM *db, datum key);
27452 int dbm_error(DBM *db);
27453 datum dbm_fetch(DBM *db, datum key);
27454 datum dbm_firstkey(DBM *db);
27455 datum dbm_nextkey(DBM *db);
27456 DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
27457 int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

27458 **DESCRIPTION**

27459 These functions create, access, and modify a database.

27460 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object  
27461 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in  
27462 the object pointed to by *dptr*.

27463 A database shall be stored in one or two files. When one file is used, the name of the database  
27464 file shall be formed by appending the suffix **.db** to the *file* argument given to *dbm\_open()*. When  
27465 two files are used, the names of the database files shall be formed by appending the suffixes **.dir**  
27466 and **.pag** respectively to the *file* argument.

27467 The *dbm\_open()* function shall open a database. The *file* argument to the function is the  
27468 pathname of the database. Values for the *open\_flags* argument are constructed by a bitwise-  
27469 inclusive OR of flags from the following list, defined in **<fcntl.h>**. Applications shall specify  
27470 exactly one of the first three values (file access modes) below in the value of *open\_flags*:

27471 **O\_RDONLY** Open the database, and the underlying file(s) used to store the database,  
27472 for reading only.

27473 **O\_RDWR** Open the database, and the underlying file(s) used to store the database,  
27474 for reading and writing.

27475 **O\_WRONLY** Open the database for writing only. The underlying file(s) used to store  
27476 the database shall be opened for reading and writing.

27477 Any combination of the following can be used:

27478 **O\_CLOEXEC** If set, the **FD\_CLOEXEC** flag for the new file descriptor(s) used to open  
27479 the underlying file(s) shall be set.

27480 **O\_CREAT** If the database exists, this flag has no effect except as noted under  
27481 **O\_EXCL** below. Otherwise, the database shall be created; the user ID of  
27482 the underlying file(s) shall be set to the effective user ID of the process;  
27483 the group ID of the underlying file(s) shall be set to the group ID of the  
27484 file's parent directory or to the effective group ID of the process.  
27485 Implementations shall provide a way to initialize the group ID to the  
27486 group ID of the parent directory. Implementations may, but need not,  
27487 provide an implementation-defined way to initialize the group ID to the  
27488 effective group ID of the calling process.

27489	SIO	O_DSYNC	Write I/O operations on the file(s) used to store the database shall complete as defined by synchronized I/O data integrity completion.
27490			
27491		O_EXCL	If O_CREAT and O_EXCL are set, <i>dbm_open()</i> shall fail if the database exists according to the following criteria. If the database is stored in a file with a <b>.db</b> suffix, a check for the existence of the file and the creation of the file if it does not exist shall be performed as a single atomic operation. If the database is stored in files with <b>.pag</b> and <b>.dir</b> suffixes, this atomic existence check and creation operation shall be performed for each file, but it is unspecified which file is first. If the first file is successfully created and the second file is found to exist, the first file should be removed before <i>dbm_open()</i> returns.
27492			
27493			
27494			
27495			
27496			
27497			
27498			
27499			
27500			If O_EXCL is set and O_CREAT is not set, the result is undefined.
27501	SIO	O_RSYNC	Read I/O operations on the file(s) used to store the database shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in oflag, all I/O operations on the file(s) shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file(s) shall complete as defined by synchronized I/O file integrity completion.
27502			
27503			
27504			
27505			
27506			
27507			
27508		O_SYNC	Write I/O operations on the file(s) used to store the database shall complete as defined by synchronized I/O file integrity completion.
27509			
27510		O_TRUNC	If the database exists and is successfully opened O_RDWR or O_WRONLY, it shall be emptied, and the mode and owner of the underlying file(s) shall be unchanged. The result of using O_TRUNC without either O_RDWR or O_WRONLY is undefined.
27511			
27512			
27513			
27514			The behaviour of other flags described for the <i>flags</i> argument of <i>open()</i> is unspecified.
27515			The <i>file_mode</i> argument has the same meaning as the third argument of <i>open()</i> and shall apply to the underlying file(s) used to store the database.
27516			
27517			The <i>dbm_open()</i> function need not accept pathnames longer than {PATH_MAX}-4 bytes (including the terminating null), or pathnames with a last component longer than {NAME_MAX}-4 bytes (excluding the terminating null).
27518			
27519			
27520			The <i>dbm_close()</i> function shall close a database. The application shall ensure that argument <i>db</i> is a pointer to a <b>dbm</b> structure that has been returned from a call to <i>dbm_open()</i> .
27521			
27522			These database functions shall support an internal block size large enough to support key/content pairs of at least 1 023 bytes.
27523			
27524			The <i>dbm_fetch()</i> function shall read a record from a database. The argument <i>db</i> is a pointer to a database structure that has been returned from a call to <i>dbm_open()</i> . The argument <i>key</i> is a <b>datum</b> that has been initialized by the application to the value of the key that matches the key of the record the program is fetching.
27525			
27526			
27527			
27528			The <i>dbm_store()</i> function shall write a record to a database. The argument <i>db</i> is a pointer to a database structure that has been returned from a call to <i>dbm_open()</i> . The argument <i>key</i> is a <b>datum</b> that has been initialized by the application to the value of the key that identifies (for subsequent reading, writing, or deleting) the record the application is writing. The argument <i>content</i> is a <b>datum</b> that has been initialized by the application to the value of the record the program is writing. The argument <i>store_mode</i> controls whether <i>dbm_store()</i> replaces any pre-existing record that has the same key that is specified by the <i>key</i> argument. The application shall
27529			
27530			
27531			
27532			
27533			
27534			

27535 set *store\_mode* to either DBM\_INSERT or DBM\_REPLACE. If the database contains a record that  
 27536 matches the *key* argument and *store\_mode* is DBM\_REPLACE, the existing record shall be  
 27537 replaced with the new record. If the database contains a record that matches the *key* argument  
 27538 and *store\_mode* is DBM\_INSERT, the existing record shall be left unchanged and the new record  
 27539 ignored. If the database does not contain a record that matches the *key* argument and *store\_mode*  
 27540 is either DBM\_INSERT or DBM\_REPLACE, the new record shall be inserted in the database.

27541 If the sum of a key/content pair exceeds the internal block size, the result is unspecified.  
 27542 Moreover, the application shall ensure that all key/content pairs that hash together fit on a  
 27543 single block. The *dbm\_store()* function shall return an error in the event that a disk block fills  
 27544 with inseparable data.

27545 The *dbm\_delete()* function shall delete a record and its key from the database. The argument *db* is  
 27546 a pointer to a database structure that has been returned from a call to *dbm\_open()*. The argument  
 27547 *key* is a **datum** that has been initialized by the application to the value of the key that identifies  
 27548 the record the program is deleting.

27549 The *dbm\_firstkey()* function shall return the first key in the database. The argument *db* is a  
 27550 pointer to a database structure that has been returned from a call to *dbm\_open()*.

27551 The *dbm\_nextkey()* function shall return the next key in the database. The argument *db* is a  
 27552 pointer to a database structure that has been returned from a call to *dbm\_open()*. The application  
 27553 shall ensure that the *dbm\_firstkey()* function is called before calling *dbm\_nextkey()*. Subsequent  
 27554 calls to *dbm\_nextkey()* return the next key until all of the keys in the database have been  
 27555 returned.

27556 The *dbm\_error()* function shall return the error condition of the database. The argument *db* is a  
 27557 pointer to a database structure that has been returned from a call to *dbm\_open()*.

27558 The *dbm\_clearerr()* function shall clear the error condition of the database. The argument *db* is a  
 27559 pointer to a database structure that has been returned from a call to *dbm\_open()*.

27560 The *dptr* pointers returned by these functions may point into static storage that may be changed  
 27561 by subsequent calls.

27562 These functions need not be thread-safe.

#### 27563 RETURN VALUE

27564 The *dbm\_store()* and *dbm\_delete()* functions shall return 0 when they succeed and a negative  
 27565 value when they fail.

27566 The *dbm\_store()* function shall return 1 if it is called with a *flags* value of DBM\_INSERT and the  
 27567 function finds an existing record with the same key.

27568 The *dbm\_error()* function shall return 0 if the error condition is not set and return a non-zero  
 27569 value if the error condition is set.

27570 The return value of *dbm\_clearerr()* is unspecified.

27571 The *dbm\_firstkey()* and *dbm\_nextkey()* functions shall return a key **datum**. When the end of the  
 27572 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*  
 27573 member of the key shall be a null pointer and the error condition of the database shall be set.

27574 The *dbm\_fetch()* function shall return a content **datum**. If no record in the database matches the  
 27575 key or if an error condition has been detected in the database, the *dptr* member of the content  
 27576 shall be a null pointer.

27577 The *dbm\_open()* function shall return a pointer to a database structure. If an error is detected  
 27578 during the operation, *dbm\_open()* shall return a (DBM \*)0.

27579 **ERRORS**

27580 No errors are defined.

27581 **EXAMPLES**

27582 None.

27583 **APPLICATION USAGE**

27584 The following code can be used to traverse the database:

27585 

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

27586 The *dbm\_\** functions provided in this library should not be confused in any way with those of a  
 27587 general-purpose database management system. These functions do not provide for multiple  
 27588 search keys per entry, they do not protect against multi-user access (in other words they do not  
 27589 lock records or files), and they do not provide the many other useful database functions that are  
 27590 found in more robust database management systems. Creating and updating databases by use of  
 27591 these functions is relatively slow because of data copies that occur upon hash collisions. These  
 27592 functions are useful for applications requiring fast lookup of relatively static information that is  
 27593 to be indexed by a single key.

27594 Note that a strictly conforming application is extremely limited by these functions: since there is  
 27595 no way to determine that the keys in use do not all hash to the same value (although that would  
 27596 be rare), a strictly conforming application cannot be guaranteed that it can store more than one  
 27597 block's worth of data in the database. As long as a key collision does not occur, additional data  
 27598 may be stored, but because there is no way to determine whether an error is due to a key  
 27599 collision or some other error condition (*dbm\_error()* being effectively a Boolean), once an error is  
 27600 detected, the application is effectively limited to guessing what the error might be if it wishes to  
 27601 continue using these functions.

27602 The *dbm\_delete()* function need not physically reclaim file space, although it does make it  
 27603 available for reuse by the database.

27604 After calling *dbm\_store()* or *dbm\_delete()* during a pass through the keys by *dbm\_firstkey()* and  
 27605 *dbm\_nextkey()*, the application should reset the database by calling *dbm\_firstkey()* before again  
 27606 calling *dbm\_nextkey()*. The contents of these files are unspecified and may not be portable.

27607 Applications should take care that database pathname arguments specified to *dbm\_open()* are  
 27608 not prefixes of unrelated files. This might be done, for example, by placing databases in a  
 27609 separate directory.

27610 Since some implementations use three characters for a suffix and others use four characters for a  
 27611 suffix, applications should ensure that the maximum portable pathname length passed to  
 27612 *dbm\_open()* is no greater than {PATH\_MAX}-4 bytes, with the last component of the pathname  
 27613 no greater than {NAME\_MAX}-4 bytes.

27614 **RATIONALE**

27615 Previously the standard required the database to be stored in two files, one file being a directory  
 27616 containing a bitmap of keys and having *.dir* as its suffix. The second file containing all data and  
 27617 having *.pag* as its suffix. This has been changed not to specify the use of the files and to allow  
 27618 newer implementations of the Berkeley DB interface using a single file that have evolved while  
 27619 remaining compatible with the application programming interface. The standard developers  
 27620 considered removing the specific suffixes altogether but decided to retain them so as not to  
 27621 pollute the application file name space more than necessary and to allow for portable backups of  
 27622 the database.

27623 **FUTURE DIRECTIONS**

27624 A future revision of this standard may mandate the file removal that is currently recommended  
27625 (by the use of ``should'') in the description of O\_EXCL.

27626 **SEE ALSO**

27627 *open()*

27628 XBD <fcntl.h>, <ndbm.h>

27629 **CHANGE HISTORY**

27630 First released in Issue 4, Version 2.

27631 **Issue 5**

27632 Moved from X/OPEN UNIX extension to BASE.

27633 Normative text previously in the APPLICATION USAGE section is moved to the  
27634 DESCRIPTION.

27635 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

27636 **Issue 6**

27637 The normative text is updated to avoid use of the term ``must'' for application requirements.

27638 **Issue 7**

27639 Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits  
27640 newer implementations of the Berkeley DB interface.

27641 Austin Group Interpretation 1003.1-2001 #156 is applied.

27642 **Issue 8**

27643 Austin Group Defect 1057 is applied, clarifying how the O\_ flags defined for use with *open()*  
27644 apply to *dbm\_open()*.

27645 **NAME**

27646 dcgettext, dcgettext\_l, dcngettext, dcngettext\_l, dgettext, dgettext\_l — message handling  
27647 functions

27648 **SYNOPSIS**

```
27649 #include <libintl.h>
27650 char *dcgettext(const char *domainname, const char *msgid,
27651               int category);
27652 char *dcgettext_l(const char *domainname, const char *msgid,
27653                 int category, locale_t locale);
27654 char *dcngettext(const char *domainname, const char *msgid,
27655                 const char *msgid_plural, unsigned long int n,
27656                 int category);
27657 char *dcngettext_l(const char *domainname, const char *msgid,
27658                  const char *msgid_plural, unsigned long int n,
27659                  int category, locale_t locale);
27660 char *dgettext(const char *domainname, const char *msgid);
27661 char *dgettext_l(const char *domainname, const char *msgid,
27662                 locale_t locale);
```

27663 **DESCRIPTION**

27664 Refer to [gettext](#).

27665 **NAME**

27666 difftime — compute the difference between two calendar time values

27667 **SYNOPSIS**

27668 #include &lt;time.h&gt;

27669 double difftime(time\_t time1, time\_t time0);

27670 **DESCRIPTION**

27671 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
27672 conflict between the requirements described here and the ISO C standard is unintentional. This  
27673 volume of POSIX.1-2024 defers to the ISO C standard.

27674 The *difftime()* function shall compute the difference between two calendar times (as returned by  
27675 *time()*):  $time1 - time0$ .

27676 **RETURN VALUE**27677 The *difftime()* function shall return the difference expressed in seconds as a type **double**.27678 **ERRORS**

27679 No errors are defined.

27680 **EXAMPLES**

27681 None.

27682 **APPLICATION USAGE**

27683 None.

27684 **RATIONALE**

27685 None.

27686 **FUTURE DIRECTIONS**

27687 None.

27688 **SEE ALSO**27689 [asctime\(\)](#), [clock\(\)](#), [ctime\(\)](#), [futimens\(\)](#), [gmtime\(\)](#), [localtime\(\)](#), [mktime\(\)](#), [strftime\(\)](#), [strptime\(\)](#), [time\(\)](#)27690 XBD [<time.h>](#)27691 **CHANGE HISTORY**

27692 First released in Issue 4. Derived from the ISO C standard.



27693 **NAME**27694 `dirfd` — extract the file descriptor used by a DIR stream27695 **SYNOPSIS**

```
27696 #include <dirent.h>
27697 int dirfd(DIR *dirp);
```

27698 **DESCRIPTION**

27699 If the directory stream referenced by *dirp* has an associated file descriptor, *dirfd()* shall return  
 27700 that file descriptor. Otherwise, *dirfd()* shall open a new file description referring to the directory  
 27701 associated with the directory stream as if by calling:

```
27702 open(DirectoryName, O_RDONLY | O_DIRECTORY | O_CLOEXEC);
```

27703 except that no pathname for use as *DirectoryName* need exist or be accessible. It shall then  
 27704 associate the new file descriptor with the directory stream, and return that file descriptor.

27705 Upon successful return from *dirfd()*, the file descriptor is under the control of the system, and if  
 27706 any attempt is made to close the file descriptor, or to modify the state of the associated  
 27707 XSI description, other than by means of *closedir()*, *readdir()*, *readdir\_r()*, *rewinddir()*, or *seekdir()*, the  
 27708 behavior is undefined. Upon calling *closedir()* the file descriptor shall be closed.

27709 **RETURN VALUE**

27710 Upon successful completion, the *dirfd()* function shall return an integer which contains a file  
 27711 descriptor for the stream pointed to by *dirp*. Otherwise, it shall return `-1` and shall set *errno* to  
 27712 indicate the error.

27713 **ERRORS**

27714 The *dirfd()* function shall fail if:

27715 [EMFILE] A new file descriptor is required and all file descriptors available to the  
 27716 process are currently open.

27717 [ENFILE] A new file descriptor is required and the maximum allowable number of files  
 27718 is currently open in the system.

27719 The *dirfd()* function may fail if:

27720 [EINVAL] The *dirp* argument does not refer to a valid directory stream.

27721 **EXAMPLES**

27722 None.

27723 **APPLICATION USAGE**

27724 The *dirfd()* function is intended to be a mechanism by which an application may obtain a file  
 27725 descriptor to use for the *fchdir()* function.

27726 **RATIONALE**

27727 This interface was introduced because the Base Definitions volume of POSIX.1-2024 does not  
 27728 make public the **DIR** data structure. Applications tend to use the *fchdir()* function on the file  
 27729 descriptor returned by this interface, and this has proven useful for security reasons; in  
 27730 particular, it is a better technique than others where directory names might change.

27731 On an implementation where reading from a directory stream does not use a file descriptor,  
 27732 *opendir()* need not allocate one to be returned by *dirfd()*. The implementation can instead delay  
 27733 the allocation of a suitable file descriptor until the first time *dirfd()* is called for the stream. A file  
 27734 descriptor allocated by *dirfd()* must be closed by *closedir()*.

27735 **FUTURE DIRECTIONS**

27736 None.

27737 **SEE ALSO**27738 *closedir()*, *fchdir()*, *fdopendir()*, *fileno()*, *open()*, *readdir()*27739 XBD <**dirent.h**>27740 **CHANGE HISTORY**

27741 First released in Issue 7.

27742 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0067 [422] is applied.

27743 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0076 [572] is applied.

27744 **Issue 8**27745 Austin Group Defects 391 and 1359 are applied, changing the DESCRIPTION and RATIONALE  
27746 sections, and adding the [EMFILE] and [ENFILE] errors.

27747 **NAME**

27748           dirname — report the parent directory name of a file pathname

27749 **SYNOPSIS**

```
27750 XSI      #include <libgen.h>
27751      char *dirname(char *path);
```

27752 **DESCRIPTION**

27753       The *dirname()* function shall take a pointer to a character string that contains a pathname, and  
 27754       return a pointer to a string that is a pathname of the directory containing the entry of the final  
 27755       pathname component. The *dirname()* function shall not perform pathname resolution; the result  
 27756       shall not be affected by whether or not *path* exists or by its file type. Trailing '/' characters in  
 27757       the pathname that are not also leading '/' characters shall not be counted as part of the  
 27758       pathname.

27759       If the pathname does not contain a '/', then *dirname()* shall return a pointer to the string ".".  
 27760       If *path* is a null pointer or points to an empty string, *dirname()* shall return a pointer to the string  
 27761       ".".

27762       It is unspecified whether redundant '/' characters and '.' pathname components in *path* are  
 27763       removed after determining the pathname to output. However, ".." pathname components  
 27764       occurring prior to the final component shall not be removed.

27765       The *dirname()* function may modify the string pointed to by *path*, and may return a pointer into  
 27766       the input string. The returned pointer might be invalidated if the input string is subsequently  
 27767       modified or freed. If *path* does not contain a '/', is a null pointer, or points to an empty string  
 27768       the returned pointer may point to constant data that cannot be modified.

27769 **RETURN VALUE**27770       The *dirname()* function shall return a pointer to a string as described above.

27771       The *dirname()* function shall always be successful and no return value is reserved to indicate an  
 27772       error.

27773 **ERRORS**

27774       No errors are defined.

27775 **EXAMPLES**

27776       The following code fragment reads a pathname, changes the current working directory to the  
 27777       parent directory, and opens the file.

```
27778      char *path = NULL, *pathcopy;
27779      size_t buflen = 0;
27780      ssize_t linelen = 0;
27781      int fd;

27782      linelen = getline(&path, &buflen, stdin);

27783      path[linelen-1] = 0;
27784      pathcopy = strdup(path);
27785      if (chdir(dirname(pathcopy)) < 0) {
27786          ...
27787      }
27788      if ((fd = open(basename(path), O_RDONLY)) >= 0) {
27789          ...
27790          close (fd);
27791      }
```

```

27792     ...
27793     free (pathcopy);
27794     free (path);

```

27795 The EXAMPLES section of the *basename()* function (see *basename()*) includes a table showing  
 27796 examples of the results of processing several sample pathnames by the *basename()* and *dirname()*  
 27797 functions and by the *basename* and *dirname* utilities.

#### 27798 APPLICATION USAGE

27799 The *dirname()* and *basename()* functions together yield a complete pathname. The expression  
 27800 *dirname(path)* obtains the pathname of the directory where *basename(path)* is found.

27801 Since the meaning of the leading `"/"` is implementation-defined, *dirname("//foo")* may return  
 27802 either `"/"` or `"/"` (but nothing else).

27803 Note that in some circumstances, the returned pointer might point into constant data. Therefore,  
 27804 if the application needs to modify the returned data, it should be copied first.

#### 27805 RATIONALE

27806 An implementation should prefer the shortest output possible; however, this is not required, in  
 27807 part because earlier versions of the standard did not mention whether elision of redundant  
 27808 `<slash>` characters or dot (`"."`) components was permitted. Removal of the dot-dot (`".."`)  
 27809 pathname component is not permitted, because eliding it correctly would require performing  
 27810 pathname resolution to ensure the resulting string would still point to the correct pathname if  
 27811 the original string resolved as a pathname. On implementations where pathname `"/"` has an  
 27812 implementation-defined meaning distinct from the pathname `"/"`, the *dirname* of `"/"` will be  
 27813 `"/"`.

27814 Earlier versions of this standard seemed to allow thread-safe and non-thread-safe  
 27815 implementations of *basename()* and *dirname()*, but did not allow implementations to return a  
 27816 null pointer and require that *errno* be set when that happened. The standard now requires  
 27817 thread-safe behavior for both of these functions and clearly states that they are always  
 27818 successful.

#### 27819 FUTURE DIRECTIONS

27820 None.

#### 27821 SEE ALSO

27822 *basename()*

27823 XBD `<libgen.h>`

27824 XCU *basename*, *dirname*

#### 27825 CHANGE HISTORY

27826 First released in Issue 4, Version 2.

#### 27827 Issue 5

27828 Moved from X/OPEN UNIX extension to BASE.

27829 Normative text previously in the APPLICATION USAGE section is moved to the  
 27830 DESCRIPTION.

27831 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

#### 27832 Issue 7

27833 Austin Group Interpretation 1003.1-2001 #156 is applied.

27834 The EXAMPLES section is revised.

- 27835 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0068 [75] is applied.
- 27836 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0077 [830], XSH/TC2-2008/0078 [612],  
27837 XSH/TC2-2008/0079 [830], XSH/TC2-2008/0080 [656], and XSH/TC2-2008/0081 [612] are  
27838 applied.
- 27839 **Issue 8**
- 27840 Austin Group Defect 1064 is applied, requiring *dirname()* to be thread-safe and allowing it to  
27841 return a pointer to constant data under certain conditions.
- 27842 Austin Group Defect 1073 is applied, changing “parent directory of that file” to “directory  
27843 containing the entry of the final pathname component” and clarifying that redundant '/'  
27844 characters and '.' pathname components may be removed after determining the pathname to  
27845 output.

27846 **NAME**

27847 div — compute the quotient and remainder of an integer division

27848 **SYNOPSIS**

27849 #include &lt;stdlib.h&gt;

27850 div\_t div(int *numer*, int *denom*);27851 **DESCRIPTION**

27852 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
27853 conflict between the requirements described here and the ISO C standard is unintentional. This  
27854 volume of POSIX.1-2024 defers to the ISO C standard.

27855 The *div()* function shall compute the quotient and remainder of the division of the numerator  
27856 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer  
27857 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be  
27858 represented, the behavior is undefined; otherwise, *quot\*denom+rem* shall equal *numer*.

27859 **RETURN VALUE**

27860 The *div()* function shall return a structure of type **div\_t**, comprising both the quotient and the  
27861 remainder. The structure includes the following members, in any order:

```
27862 int quot; /* quotient */  
27863 int rem; /* remainder */
```

27864 **ERRORS**

27865 No errors are defined.

27866 **EXAMPLES**

27867 None.

27868 **APPLICATION USAGE**

27869 None.

27870 **RATIONALE**

27871 None.

27872 **FUTURE DIRECTIONS**

27873 None.

27874 **SEE ALSO**27875 [ldiv\(\)](#)27876 XBD [<stdlib.h>](#)27877 **CHANGE HISTORY**

27878 First released in Issue 4. Derived from the ISO C standard.

27879 **NAME**

27880 dladdr — get information relating to an address

27881 **SYNOPSIS**

27882 #include &lt;dlfcn.h&gt;

27883 int dladdr(const void \*restrict addr, Dl\_info\_t \*restrict dlip);

27884 **DESCRIPTION**

27885 The *dladdr()* function shall determine whether the address specified by *addr* is located within the  
 27886 address range occupied by a mapped object. The mapped objects examined shall include any  
 27887 executable object files that have previously been loaded by a call to *dlopen()* and for which  
 27888 *dlclose()* has not subsequently been called, and any shared library files that were loaded as  
 27889 dependencies of the executable file from which the current process image was loaded; they may  
 27890 also include any executable object files that have previously been loaded by a call to *dlopen()*  
 27891 and for which *dlclose()* has subsequently been called, the executable file from which the current  
 27892 process image was loaded, and implementation-defined additional mapped objects (for  
 27893 example, all regular files mapped using *mmap()* might be included). If the specified address is  
 27894 within the mapped address range of one of these mapped objects and the object contains a  
 27895 symbol table, the symbol table shall be searched for a symbol (a function identifier or a data  
 27896 object identifier) that has the largest address less than or equal to the specified address.

27897 If the address specified by *addr* is within the mapped address range of one of the examined  
 27898 mapped objects, the structure pointed to by *dliip* shall be populated as follows:

- 27899 • The value of the *dli\_fname* member shall be set to point to the pathname of the mapped  
 27900 object. (This might no longer resolve to the file that was mapped, for example if it was a  
 27901 link that has subsequently been removed or renamed.)
- 27902 • The value of the *dli\_fbase* member shall be set to the base of the address range occupied by  
 27903 the mapped object.
- 27904 • The value of the *dli\_sname* member shall be set to point to the name of the symbol that has  
 27905 the largest address less than or equal to the specified address, or to a null pointer if no such  
 27906 symbol was found.
- 27907 • If *dli\_sname* is set to a null pointer, the value of the *dli\_saddr* member shall also be set to a  
 27908 null pointer. Otherwise, if *dli\_sname* names a function identifier, *dli\_saddr* shall be set to the  
 27909 address of the function converted from type pointer to function to type pointer to **void**;  
 27910 otherwise, *dli\_saddr* shall be set to the address of the data object named by *dli\_sname*  
 27911 converted from a pointer to the type of the data object to a pointer to **void**.

27912 **RETURN VALUE**

27913 Upon successful completion, a non-zero value shall be returned. If the specified address is not  
 27914 located within the address range occupied by an examined mapped object, or if an error occurs,  
 27915 zero shall be returned. More detailed diagnostic information shall be available through *dlderror()*.

27916 **ERRORS**

27917 No errors are defined.

27918 **EXAMPLES**

27919 None.

27920 **APPLICATION USAGE**

27921 The **DI\_info\_t** members may point to addresses within the mapped object. These pointers can  
27922 become invalid if the object is unmapped (for example, loaded executable objects may be  
27923 unloaded by *dlclose()*).

27924 If *dli\_sname* names a function identifier, the value of *dli\_saddr* can be converted back to type  
27925 pointer to function using a cast in the manner shown in the *dlsym()* **EXAMPLES** section. Note  
27926 that this conversion is not defined by the ISO C standard. This standard requires this conversion  
27927 to work correctly on conforming implementations.

27928 **RATIONALE**

27929 None.

27930 **FUTURE DIRECTIONS**

27931 None.

27932 **SEE ALSO**27933 *dlclose()*, *dLError()*, *dlopen()*, *dlsym()*27934 XBD <**dlfcn.h**>27935 **CHANGE HISTORY**

27936 First released in Issue 8.



27937 **NAME**27938 `dlclose` — close a symbol table handle27939 **SYNOPSIS**

```
27940 #include <dlfcn.h>
27941 int dlclosel(void *handle);
```

27942 **DESCRIPTION**

27943 The `dlclose()` function shall inform the system that the symbol table handle specified by *handle* is  
27944 no longer needed by the application.

27945 An application writer may use `dlclose()` to make a statement of intent on the part of the process,  
27946 but this statement does not create any requirement upon the implementation. When the symbol  
27947 table handle is closed, the implementation may unload the executable object files that were  
27948 loaded by `dlopen()` when the symbol table handle was opened and those that were loaded by  
27949 `dlsym()` when using the symbol table handle identified by *handle*.

27950 Once a symbol table handle has been closed, an application should assume that any symbols  
27951 (function identifiers and data object identifiers) made visible using *handle*, are no longer  
27952 available to the process.

27953 Although a `dlclose()` operation is not required to remove any functions or data objects from the  
27954 address space, neither is an implementation prohibited from doing so. The only restriction on  
27955 such a removal is that no function nor data object shall be removed to which references have  
27956 been relocated, until or unless all such references are removed. For instance, an executable object  
27957 file that had been loaded with a `dlopen()` operation specifying the `RTLD_GLOBAL` flag might  
27958 provide a target for dynamic relocations performed in the processing of other relocatable  
27959 objects—in such environments, an application may assume that no relocation, once made, shall  
27960 be undone or remade unless the executable object file containing the relocated object has itself  
27961 been removed.

27962 **RETURN VALUE**

27963 If the referenced symbol table handle was successfully closed, `dlclose()` shall return 0. If *handle*  
27964 does not refer to an open symbol table handle or if the symbol table handle could not be closed,  
27965 `dlclose()` shall return a non-zero value. More detailed diagnostic information shall be available  
27966 through `dlerror()`.

27967 **ERRORS**

27968 No errors are defined.

27969 **EXAMPLES**27970 The following example illustrates use of `dlopen()` and `dlclose()`:

```
27971 #include <dlfcn.h>
27972 int eret;
27973 void *mylib;
27974 ...
27975 /* Open a dynamic library and then close it ... */
27976 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
27977 ...
27978 eret = dlclosel(mylib);
27979 ...
```

27980 **APPLICATION USAGE**

27981 A conforming application should employ a symbol table handle returned from a *dlopen()*  
27982 invocation only within a given scope bracketed by a *dlopen()* operation and the corresponding  
27983 *dlclose()* operation. Implementations are free to use reference counting or other techniques such  
27984 that multiple calls to *dlopen()* referencing the same executable object file may return a pointer to  
27985 the same data object as the symbol table handle.

27986 Implementations are also free to re-use a handle. For these reasons, the value of a handle must  
27987 be treated as an opaque data type by the application, used only in calls to *dlsym()* and *dlclose()*.

27988 **RATIONALE**

27989 None.

27990 **FUTURE DIRECTIONS**

27991 None.

27992 **SEE ALSO**

27993 *dladdr()*, *dlerror()*, *dlopen()*, *dlsym()*

27994 XBD <dlfcn.h>

27995 **CHANGE HISTORY**

27996 First released in Issue 5.

27997 **Issue 6**

27998 The DESCRIPTION is updated to say that the referenced object is closed “if this is the last  
27999 reference to it”.

28000 **Issue 7**

28001 The *dlopen()* function is moved from the XSI option to Base.

28002 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0069 [74] is applied.

28003 **Issue 8**

28004 Austin Group Defect 993 is applied, adding *dladdr()* to the SEE ALSO section.

28005 **NAME**

28006 dlerror — get diagnostic information

28007 **SYNOPSIS**

```
28008 #include <dlfcn.h>
28009 char *dlerror(void);
```

28010 **DESCRIPTION**

28011 The *dlerror()* function shall return a null-terminated character string (with no trailing  
 28012 <newline>) that describes the last error that occurred during dynamic linking processing. If no  
 28013 dynamic linking errors have occurred since the last invocation of *dlerror()*, *dlerror()* shall return  
 28014 NULL. Thus, invoking *dlerror()* a second time, immediately following a prior invocation, shall  
 28015 result in NULL being returned.

28016 It is implementation-defined whether or not the *dlerror()* function is thread-safe. A thread-safe  
 28017 implementation shall return only errors that occur on the current thread.

28018 **RETURN VALUE**

28019 If successful, *dlerror()* shall return a null-terminated character string; otherwise, NULL shall be  
 28020 returned.

28021 The application shall not modify the string returned. The returned pointer might be invalidated  
 28022 or the string content might be overwritten by a subsequent call to *dlerror()* in the same thread (if  
 28023 *dlerror()* is thread-safe) or in any thread (if *dlerror()* is not thread-safe). The returned pointer  
 28024 might also be invalidated if the calling thread is terminated.

28025 **ERRORS**

28026 No errors are defined.

28027 **EXAMPLES**

28028 The following example prints out the last dynamic linking error:

```
28029 ...
28030 #include <dlfcn.h>
28031 char *errstr;
28032 errstr = dlerror();
28033 if (errstr != NULL)
28034     printf ("A dynamic linking error occurred: (%s)\n", errstr);
28035 ...
```

28036 **APPLICATION USAGE**

28037 Depending on the application environment with respect to asynchronous execution events, such  
 28038 as signals or other asynchronous computation sharing the address space, conforming  
 28039 applications should use a critical section to retrieve the error pointer and buffer.

28040 **RATIONALE**

28041 None.

28042 **FUTURE DIRECTIONS**

28043 None.

28044 **SEE ALSO**28045 [dladdr\(\)](#), [dlclose\(\)](#), [dlopen\(\)](#), [dlsym\(\)](#)28046 XBD [<dlfcn.h>](#)

28047 **CHANGE HISTORY**

28048 First released in Issue 5.

28049 **Issue 6**

28050 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

28051 **Issue 7**

28052 Austin Group Interpretation 1003.1-2001 #156 is applied.

28053 The *dlerror()* function is moved from the XSI option to the Base.28054 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0070 [75], XSH/TC1-2008/0071 [97],  
28055 and XSH/TC1-2008/0072 [133] are applied.

28056 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0082 [656] is applied.

28057 **Issue 8**28058 Austin Group Defect 993 is applied, adding *dladdr()* to the SEE ALSO section.

28059 **NAME**

28060 dlopen — open a symbol table handle

28061 **SYNOPSIS**

28062 #include &lt;dlfcn.h&gt;

28063 void \*dlopen(const char \*file, int mode);

28064 **DESCRIPTION**28065 The *dlopen()* function shall make the symbols (function identifiers and data object identifiers) in  
28066 the executable object file specified by *file* available to the calling program.28067 The class of executable object files eligible for this operation and the manner of their  
28068 construction are implementation-defined, though typically such files are shared libraries or  
28069 programs.28070 Implementations may permit the construction of embedded dependencies in executable object  
28071 files. In such cases, a *dlopen()* operation shall load those dependencies in addition to the  
28072 executable object file specified by *file*. Implementations may also impose specific constraints on  
28073 the construction of programs that can employ *dlopen()* and its related services.28074 A successful *dlopen()* shall return a symbol table handle which the caller may use on subsequent  
28075 calls to *dlsym()* and *dlclose()*.

28076 The value of this symbol table handle should not be interpreted in any way by the caller.

28077 The *file* argument is used to construct a pathname to the executable object file. If *file* contains a  
28078 <slash> character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in  
28079 an implementation-defined manner to yield a pathname.28080 If *file* is a null pointer, *dlopen()* shall return a global symbol table handle for the currently  
28081 running process image. This symbol table handle shall provide access to the symbols from an  
28082 ordered set of executable object files consisting of the original program image file, any  
28083 executable object files loaded at program start-up as specified by that process file (for example,  
28084 shared libraries), and the set of executable object files loaded using *dlopen()* operations with the  
28085 RTLD\_GLOBAL flag. As the latter set of executable object files can change during execution, the  
28086 set of symbols made available by this symbol table handle can also change dynamically.28087 Only a single copy of an executable object file shall be brought into the address space, even if  
28088 *dlopen()* is invoked multiple times in reference to the executable object file, and even if different  
28089 pathnames are used to reference the executable object file.28090 The *mode* parameter describes how *dlopen()* shall operate upon *file* with respect to the processing  
28091 of relocations and the scope of visibility of the symbols provided within *file*. When an  
28092 executable object file is brought into the address space of a process, it may contain references to  
28093 symbols whose addresses are not known until the executable object file is loaded.28094 These references shall be relocated before the symbols can be accessed. The *mode* parameter  
28095 governs when these relocations take place and may have the following values:28096 **RTLD\_LAZY** Relocations shall be performed at an implementation-defined time, ranging  
28097 from the time of the *dlopen()* call until the first reference to a given symbol  
28098 occurs. Specifying **RTLD\_LAZY** should improve performance on  
28099 implementations supporting dynamic symbol binding since a process might  
28100 not reference all of the symbols in an executable object file. And, for systems  
28101 supporting dynamic symbol resolution for normal process execution, this  
28102 behavior mimics the normal handling of process execution.

28103 RTLD\_NOW All necessary relocations shall be performed when the executable object file is  
 28104 first loaded. This may waste some processing if relocations are performed for  
 28105 symbols that are never referenced. This behavior may be useful for  
 28106 applications that need to know that all symbols referenced during execution  
 28107 will be available before *dlopen()* returns.

28108 Any executable object file loaded by *dlopen()* that requires relocations against global symbols  
 28109 can reference the symbols in the original process image file, any executable object files loaded at  
 28110 program start-up, from the initial process image itself, from any other executable object file  
 28111 included in the same *dlopen()* invocation, and any executable object files that were loaded in any  
 28112 *dlopen()* invocation and which specified the RTLD\_GLOBAL flag. To determine the scope of  
 28113 visibility for the symbols loaded with a *dlopen()* invocation, the *mode* parameter should be a  
 28114 bitwise-inclusive OR with one of the following values:

28115 RTLD\_GLOBAL The executable object file's symbols shall be made available for relocation  
 28116 processing of any other executable object file. In addition, symbol lookup  
 28117 using *dlopen(NULL,mode)* and an associated *dlsym()* allows executable object  
 28118 files loaded with this mode to be searched.

28119 RTLD\_LOCAL The executable object file's symbols shall not be made available for relocation  
 28120 processing of any other executable object file.

28121 If neither RTLD\_GLOBAL nor RTLD\_LOCAL is specified, the default behavior is unspecified.

28122 If an executable object file is specified in multiple *dlopen()* invocations, *mode* is interpreted at  
 28123 each invocation.

28124 If RTLD\_NOW has been specified, all relocations shall have been completed rendering further  
 28125 RTLD\_NOW operations redundant and any further RTLD\_LAZY operations irrelevant.

28126 If RTLD\_GLOBAL has been specified, the executable object file shall maintain the  
 28127 RTLD\_GLOBAL status regardless of any previous or future specification of RTLD\_LOCAL, as  
 28128 long as the executable object file remains in the address space (see *dlclose()*). If there was a  
 28129 previous specification of RTLD\_LOCAL, it is unspecified whether relocations after the new  
 28130 specification of RTLD\_GLOBAL are made as if the previous specification had been  
 28131 RTLD\_GLOBAL or as if the executable object file had not previously been loaded.

28132 Symbols introduced into the process image through calls to *dlopen()* may be used in relocation  
 28133 activities. Symbols so introduced may duplicate symbols already defined by the program or  
 28134 previous *dlopen()* operations. To resolve the ambiguities such a situation might present, the  
 28135 resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two  
 28136 such resolution orders are defined: load order and dependency order. Load order establishes an  
 28137 ordering among symbol definitions, such that the first definition loaded (including definitions  
 28138 from the process image file and any dependent executable object files loaded with it) has priority  
 28139 over executable object files added later (by *dlopen()*). Load ordering is used in relocation  
 28140 processing. Dependency ordering uses a breadth-first order starting with a given executable  
 28141 object file, then all of its dependencies, then any dependents of those, iterating until all  
 28142 dependencies are satisfied. With the exception of the global symbol table handle obtained via a  
 28143 *dlopen()* operation with a null pointer as the *file* argument, dependency ordering is used by the  
 28144 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol table  
 28145 handle.

28146 When an executable object file is first made accessible via *dlopen()*, it and its dependent  
 28147 executable object files are added in dependency order. Once all the executable object files are  
 28148 added, relocations are performed using load order. Note that if an executable object file or its  
 28149 dependencies had been previously loaded, the load and dependency orders may yield different

28150 resolutions.

28151 The symbols introduced by *dlopen()* operations and available through *dlsym()* are at a minimum  
28152 those which are exported as identifiers of global scope by the executable object file. Typically,  
28153 such identifiers shall be those that were specified in (for example) C source code as having  
28154 **extern** linkage. The precise manner in which an implementation constructs the set of exported  
28155 symbols for an executable object file is implementation-defined.

28156 **RETURN VALUE**  
28157 Upon successful completion, *dlopen()* shall return a symbol table handle. If *file* cannot be found,  
28158 cannot be opened for reading, is not of an appropriate executable object file format for  
28159 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its  
28160 symbolic references, *dlopen()* shall return a null pointer. More detailed diagnostic information  
28161 shall be available through *dLError()*.

28162 **ERRORS**  
28163 No errors are defined.

28164 **EXAMPLES**  
28165 Refer to *dlsym()*.

28166 **APPLICATION USAGE**  
28167 None.

28168 **RATIONALE**  
28169 None.

28170 **FUTURE DIRECTIONS**  
28171 None.

28172 **SEE ALSO**  
28173 *dladdr()*, *dlclose()*, *dLError()*, *dlsym()*  
28174 XBD <dlfcn.h>

28175 **CHANGE HISTORY**  
28176 First released in Issue 5.

28177 **Issue 6**  
28178 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/21 is applied, changing the default  
28179 behavior in the DESCRIPTION when neither RTLD\_GLOBAL nor RTLD\_LOCAL are specified  
28180 from implementation-defined to unspecified.

28181 **Issue 7**  
28182 The *dlopen()* function is moved from the XSI option to the Base.  
28183 The EXAMPLES section is updated to refer to *dlsym()*.  
28184 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0073 [74] is applied.

28185 **Issue 8**  
28186 Austin Group Defect 982 is applied, clarifying the requirements when changing from  
28187 RTLD\_LOCAL to RTLD\_GLOBAL.  
28188 Austin Group Defect 993 is applied, adding *dladdr()* to the SEE ALSO section.

28189 **NAME**28190 `dlsym` — get the address of a symbol from a symbol table handle28191 **SYNOPSIS**28192 `#include <dlfcn.h>`28193 `void *dlsym(void *restrict handle, const char *restrict name);`28194 **DESCRIPTION**

28195 The `dlsym()` function shall obtain the address of a symbol (a function identifier or a data object  
 28196 identifier) defined in the symbol table identified by the `handle` argument. The `handle` argument is  
 28197 a symbol table handle returned from a call to `dlopen()` (and which has not since been released by  
 28198 a call to `dlclose()`), and `name` is the symbol's name as a character string. The return value from  
 28199 `dlsym()`, converted from type pointer to **void** to a pointer to the type of the named symbol, can  
 28200 be used to call (in the case of a function) or access the contents of (in the case of a data object) the  
 28201 named symbol.

28202 The `dlsym()` function shall search for the named symbol in the symbol table referenced by `handle`.  
 28203 If the symbol table was created with lazy loading (see `RTLD_LAZY` in `dlopen()`), load ordering  
 28204 shall be used in `dlsym()` operations to relocate executable object files needed to resolve the  
 28205 symbol. The symbol resolution algorithm used shall be dependency order as described in  
 28206 `dlopen()`.

28207 The `RTLD_DEFAULT` and `RTLD_NEXT` symbolic constants (which may be defined in  
 28208 `<dlfcn.h>`) are reserved for future use as special values that applications may be allowed to use  
 28209 for `handle`.

28210 **RETURN VALUE**

28211 Upon successful completion, if `name` names a function identifier, `dlsym()` shall return the address  
 28212 of the function converted from type pointer to function to type pointer to **void**; otherwise,  
 28213 `dlsym()` shall return the address of the data object associated with the data object identifier  
 28214 named by `name` converted from a pointer to the type of the data object to a pointer to **void**. If  
 28215 `handle` does not refer to a valid symbol table handle or if the symbol named by `name` cannot be  
 28216 found in the symbol table associated with `handle`, `dlsym()` shall return a null pointer.

28217 More detailed diagnostic information shall be available through `dLError()`.

28218 **ERRORS**

28219 No errors are defined.

28220 **EXAMPLES**

28221 The following example shows how `dlopen()` and `dlsym()` can be used to access either a function  
 28222 or a data object. For simplicity, error checking has been omitted.

```
28223 void *handle;
28224 int (*fptr)(int), *iptr, result;
28225 /* open the needed symbol table */
28226 handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);
28227 /* find the address of the function my_function */
28228 fptr = (int (*)(int))dlsym(handle, "my_function");
28229 /* find the address of the data object my_object */
28230 iptr = (int *)dlsym(handle, "my_OBJ");
28231 /* invoke my_function, passing the value of my_OBJ as the parameter */
28232 result = (*fptr)(*iptr);
```



28233 **APPLICATION USAGE**

28234 The following special purpose values for *handle* are reserved for future use and have the  
28235 indicated meanings:

28236 **RTLD\_DEFAULT** The identifier lookup happens in the normal global scope; that is, a search for  
28237 an identifier using *handle* would find the same definition as a direct use of this  
28238 identifier in the program code.

28239 **RTLD\_NEXT** Specifies the next executable object file after this one that defines *name*. This  
28240 one refers to the executable object file containing the invocation of *dlsym()*.  
28241 The next executable object file is the one found upon the application of a load  
28242 order symbol resolution algorithm (see *dlopen()*). The next symbol is either  
28243 one of global scope (because it was introduced as part of the original process  
28244 image or because it was added with a *dlopen()* operation including the  
28245 **RTLD\_GLOBAL** flag), or is in an executable object file that was included in the  
28246 same *dlopen()* operation that loaded this one.

28247 The **RTLD\_NEXT** flag is useful to navigate an intentionally created hierarchy of multiply-  
28248 defined symbols created through interposition. For example, if a program wished to create an  
28249 implementation of *malloc()* that embedded some statistics gathering about memory allocations,  
28250 such an implementation could use the real *malloc()* definition to perform the memory allocation  
28251 — and itself only embed the necessary logic to implement the statistics gathering function.

28252 Note that conversion from a **void \*** pointer to a function pointer as in:

```
28253 fptr = (int (*)(int))dlsym(handle, "my_function");
```

28254 is not defined by the ISO C standard. This standard requires this conversion to work correctly on  
28255 conforming implementations.

28256 **RATIONALE**

28257 None.

28258 **FUTURE DIRECTIONS**

28259 None.

28260 **SEE ALSO**

28261 [dladdr\(\)](#), [dlclose\(\)](#), [dlerror\(\)](#), [dlopen\(\)](#)

28262 XBD [<dlfcn.h>](#)

28263 **CHANGE HISTORY**

28264 First released in Issue 5.

28265 **Issue 6**

28266 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the  
28267 ISO/IEC 9899:1999 standard.

28268 The **RTLD\_DEFAULT** and **RTLD\_NEXT** flags are reserved for future use.

28269 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/14 is applied, correcting an example, and  
28270 adding text to the **RATIONALE** describing issues related to conversion of pointers to functions  
28271 and back again.

28272 **Issue 7**

28273 The *dlsym()* function is moved from the XSI option to the Base.

28274 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0074 [74] is applied.

28275 **Issue 8**

28276 Austin Group Defect 993 is applied, adding *dladdr()* to the SEE ALSO section.

28277 Austin Group Defect 1644 is applied, clarifying that the return value from *dlsym()* can be  
28278 converted from type pointer to **void** to a pointer to the type of the named symbol using any  
28279 valid conversion method, not just casting.

28280 **NAME**

28281 dngettext, dngettext\_l — message handling functions

28282 **SYNOPSIS**

28283 #include &lt;libintl.h&gt;

```
28284 char *dngettext(const char *domainname, const char *msgid,  
28285                const char *msgid_plural, unsigned long int n);  
28286 char *dngettext_l(const char *domainname, const char *msgid,  
28287                  const char *msgid_plural, unsigned long int n,  
28288                  locale_t locale);
```

28289 **DESCRIPTION**28290 Refer to *gettext*.

28291 **NAME**

28292           dprintf — print formatted output

28293 **SYNOPSIS**

```
28294 CX       #include <stdio.h>  
28295       int dprintf(int filides, const char *restrict format, ...);
```

28296 **DESCRIPTION**

28297       Refer to *fprintf()*.

28298 **NAME**

28299 drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate  
 28300 uniformly distributed pseudo-random numbers

28301 **SYNOPSIS**

```
28302 XSI #include <stdlib.h>
28303 double drand48(void);
28304 double erand48(unsigned short xsubi[3]);
28305 long jrand48(unsigned short xsubi[3]);
28306 void lcong48(unsigned short param[7]);
28307 long lrand48(void);
28308 long mrand48(void);
28309 long nrand48(unsigned short xsubi[3]);
28310 unsigned short *seed48(unsigned short seed16v[3]);
28311 void srand48(long seedval);
```

28312 **DESCRIPTION**

28313 This family of functions shall generate pseudo-random numbers using a linear congruential  
 28314 algorithm and 48-bit integer arithmetic.

28315 The *drand48()* and *erand48()* functions shall return non-negative, double-precision, floating-  
 28316 point values, uniformly distributed over the interval [0.0,1.0).

28317 The *lrnd48()* and *nrnd48()* functions shall return non-negative, long integers, uniformly  
 28318 distributed over the interval  $[0, 2^{31})$ .

28319 The *mrnd48()* and *jrnd48()* functions shall return signed long integers uniformly distributed  
 28320 over the interval  $[-2^{31}, 2^{31})$ .

28321 The *srand48()*, *seed48()*, and *lcong48()* functions are initialization entry points, one of which  
 28322 should be invoked before either *drand48()*, *lrnd48()*, or *mrnd48()* is called. (Although it is not  
 28323 recommended practice, constant default initializer values shall be supplied automatically if  
 28324 *drand48()*, *lrnd48()*, or *mrnd48()* is called without a prior call to an initialization entry point.)  
 28325 The *erand48()*, *nrnd48()*, and *jrnd48()* functions do not require an initialization entry point to  
 28326 be called first.

28327 All the routines work by generating a sequence of 48-bit integer values,  $X_i$ , according to the  
 28328 linear congruential formula:

$$28329 X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

28330 The parameter  $m = 2^{48}$ ; hence 48-bit integer arithmetic is performed. Unless *lcong48()* is invoked,  
 28331 the multiplier value  $a$  and the addend value  $c$  are given by:

$$28332 a = 5\text{DEECE66D}_{16} = 273673163155_8$$

$$28333 c = \text{B}_{16} = 13_8$$

28334 The value returned by any of the *drand48()*, *erand48()*, *jrnd48()*, *lrnd48()*, *mrnd48()*, or  
 28335 *nrnd48()* functions is computed by first generating the next 48-bit  $X_i$  in the sequence.  $X_i$  is then  
 28336 converted to the return value as follows:

- 28337 • For *drand48()* and *erand48()* the value shall be  $2^{-48}$  times  $X_i$ .
- 28338 • For *jrnd48()* and *mrnd48()* the value shall be the largest integer not greater than  $2^{-16}$   
 28339 times  $X_i$ .

28340           • For *lrand48()* and *nrand48()* the value shall be the largest integer not greater than  $2^{-17}$   
28341           times  $X_i$ .

28342           The *drand48()*, *lrand48()*, and *mrnd48()* functions store the last 48-bit  $X_i$  generated in an  
28343           internal buffer; that is why the application shall ensure that these are initialized prior to being  
28344           invoked. The *erand48()*, *nrand48()*, and *jrnd48()* functions require the calling program to  
28345           provide storage for the successive  $X_i$  values in the array specified as an argument when the  
28346           functions are invoked. That is why these routines do not have to be initialized; the calling  
28347           program merely has to place the desired initial value of  $X_i$  into the array and pass it as an  
28348           argument. By using different arguments, *erand48()*, *nrand48()*, and *jrnd48()* allow separate  
28349           modules of a large program to generate several *independent* streams of pseudo-random numbers;  
28350           that is, the sequence of numbers in each stream shall *not* depend upon how many times the  
28351           routines are called to generate numbers for the other streams.

28352           The initializer function *srand48()* sets the high-order 32 bits of  $X_i$  to the low-order 32 bits  
28353           contained in its argument. The low-order 16 bits of  $X_i$  are set to the arbitrary value  $330E_{16}$ .

28354           The initializer function *seed48()* sets the value of  $X_i$  to the 48-bit value specified in the argument  
28355           array. The low-order 16 bits of  $X_i$  are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16  
28356           bits of  $X_i$  are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of  $X_i$  are set to the  
28357           low-order 16 bits of *seed16v*[2]. In addition, the previous value of  $X_i$  is copied into a 48-bit  
28358           internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by  
28359           *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is  
28360           to be restarted from a given point at some future time—use the pointer to get at and store the  
28361           last  $X_i$  value, and then use this value to reinitialize via *seed48()* when the program is restarted.

28362           The initializer function *lcong48()* allows the user to specify the initial  $X_i$ , the multiplier value  $a$ ,  
28363           and the addend value  $c$ . Argument array elements *param*[0-2] specify  $X_i$ , *param*[3-5] specify the  
28364           multiplier  $a$ , and *param*[6] specifies the 16-bit addend  $c$ . After *lcong48()* is called, a subsequent  
28365           call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values,  $a$  and  
28366            $c$ , specified above.

28367           The *drand48()*, *lrand48()*, and *mrnd48()* functions need not be thread-safe.

#### 28368 RETURN VALUE

28369           As described in the DESCRIPTION above.

#### 28370 ERRORS

28371           No errors are defined.

#### 28372 EXAMPLES

28373           The following program tests that the required pseudo-random number generator is used by  
28374           these functions.

```
28375           #include <assert.h>
28376           #include <stdlib.h>
28377           int main() {
28378                {
28379                   unsigned short xsubi[3] = {37174, 64810, 11603};
28380                   double d = erand48(xsubi);
28381                   assert(d >= 0.896);
28382                   assert(d <= 0.897);
28383                   assert(xsubi[0] == 22537);
28384                   assert(xsubi[1] == 47966);
28385                   assert(xsubi[2] == 58735);
28386                   d = erand48(xsubi);
```

```
28387     assert(d >= 0.337);
28388     assert(d <= 0.338);
28389     assert(xsubi[0] == 37344);
28390     assert(xsubi[1] == 32911);
28391     assert(xsubi[2] == 22119);
28392     d = erand48(xsubi);
28393     assert(d >= 0.647);
28394     assert(d <= 0.648);
28395     assert(xsubi[0] == 23659);
28396     assert(xsubi[1] == 29872);
28397     assert(xsubi[2] == 42445);
28398     d = erand48(xsubi);
28399     assert(d >= 0.500);
28400     assert(d <= 0.501);
28401     assert(xsubi[0] == 31642);
28402     assert(xsubi[1] == 7875);
28403     assert(xsubi[2] == 32802);
28404     d = erand48(xsubi);
28405     assert(d >= 0.506);
28406     assert(d <= 0.507);
28407     assert(xsubi[0] == 64669);
28408     assert(xsubi[1] == 14399);
28409     assert(xsubi[2] == 33170);
28410 }

28411 {
28412     unsigned short xsubi[3] = {25175, 11052, 45015};
28413     assert(jrand48(xsubi) == 1699503220);
28414     assert(xsubi[0] == 2326);
28415     assert(xsubi[1] == 23668);
28416     assert(xsubi[2] == 25932);
28417     assert(jrand48(xsubi) == -992276007);
28418     assert(xsubi[0] == 41577);
28419     assert(xsubi[1] == 4569);
28420     assert(xsubi[2] == 50395);
28421     assert(jrand48(xsubi) == -19535776);
28422     assert(xsubi[0] == 31936);
28423     assert(xsubi[1] == 59488);
28424     assert(xsubi[2] == 65237);
28425     assert(jrand48(xsubi) == 79438377);
28426     assert(xsubi[0] == 40395);
28427     assert(xsubi[1] == 8745);
28428     assert(xsubi[2] == 1212);
28429     assert(jrand48(xsubi) == -1258917728);
28430     assert(xsubi[0] == 37242);
28431     assert(xsubi[1] == 28832);
28432     assert(xsubi[2] == 46326);
28433 }

28434 {
28435     unsigned short xsubi[3] = {546, 33817, 23389};
28436     assert(nrand48(xsubi) == 914920692);
28437     assert(xsubi[0] == 29829);
```

```

28438         assert(xsubi[1] == 10728);
28439         assert(xsubi[2] == 27921);
28440         assert(nrand48(xsubi) == 754104482);
28441         assert(xsubi[0] == 6828);
28442         assert(xsubi[1] == 28997);
28443         assert(xsubi[2] == 23013);
28444         assert(nrand48(xsubi) == 609453945);
28445         assert(xsubi[0] == 58183);
28446         assert(xsubi[1] == 3826);
28447         assert(xsubi[2] == 18599);
28448         assert(nrand48(xsubi) == 1878644360);
28449         assert(xsubi[0] == 36678);
28450         assert(xsubi[1] == 44304);
28451         assert(xsubi[2] == 57331);
28452         assert(nrand48(xsubi) == 2114923686);
28453         assert(xsubi[0] == 58585);
28454         assert(xsubi[1] == 22861);
28455         assert(xsubi[2] == 64542);
28456     }
28457 }

```

#### 28458 APPLICATION USAGE

28459 These functions should be avoided whenever non-trivial requirements (including safety) have to  
 28460 be fulfilled, unless seeded using *getentropy()*.

#### 28461 RATIONALE

28462 None.

#### 28463 FUTURE DIRECTIONS

28464 None.

#### 28465 SEE ALSO

28466 *getentropy()*, *initstate()*, *rand()*

28467 XBD <stdlib.h>

#### 28468 CHANGE HISTORY

28469 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 28470 Issue 5

28471 A note indicating that the *drand48()*, *lrand48()*, and *mrnd48()* functions need not be reentrant is  
 28472 added to the DESCRIPTION.

#### 28473 Issue 6

28474 The normative text is updated to avoid use of the term “must” for application requirements.

#### 28475 Issue 7

28476 Austin Group Interpretation 1003.1-2001 #156 is applied.

28477 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0083 [743] is applied.

#### 28478 Issue 8

28479 Austin Group Defect 1107 is applied, clarifying how the return value is calculated from  $X_i$  for  
 28480 each function.

28481 Austin Group Defect 1134 is applied, adding *getentropy()*.



28482 **NAME**

28483 dup, dup2, dup3 — duplicate an open file descriptor

28484 **SYNOPSIS**

```
28485 #include <unistd.h>
28486 int dup(int filde);
28487 int dup2(int filde, int filde2);
28488 int dup3(int filde, int filde2, int flag);
```

28489 **DESCRIPTION**

28490 The *dup()* function provides an alternative interface to the service provided by *fcntl()* using the  
 28491 *F\_DUPFD* command. The call *dup(filde)* shall be equivalent to:

```
28492 fcntl(filde, F_DUPFD, 0);
```

28493 The *dup2()* function shall cause the file descriptor *filde2* to refer to the same open file  
 28494 description as the file descriptor *filde* and to share any locks, and shall return *filde2*. If *filde2* is  
 28495 already a valid open file descriptor, it shall be closed first, unless *filde* is equal to *filde2* in which  
 28496 case *dup2()* shall return *filde2* without closing it. If the close operation fails to close *filde2*,  
 28497 *dup2()* shall return  $-1$  without changing the open file description to which *filde2* refers. If *filde*  
 28498 is not a valid file descriptor, *dup2()* shall return  $-1$  and shall not close *filde2*. If *filde2* is less than  
 28499 0 or greater than or equal to  $\{\text{OPEN\_MAX}\}$ , *dup2()* shall return  $-1$  with *errno* set to  $[\text{EBADF}]$ .

28500 Upon successful completion, if *filde* is not equal to *filde2*, the *FD\_CLOEXEC* and  
 28501 *FD\_CLOFORK* flags associated with *filde2* shall be cleared. If *filde* is equal to *filde2*, the  
 28502 *FD\_CLOEXEC* and *FD\_CLOFORK* flags associated with *filde2* shall not be changed.

28503 The *dup3()* function shall be equivalent to the *dup2()* function, except that it shall be an error if  
 28504 *filde* is equal to *filde2*, and the state of *FD\_CLOEXEC* and *FD\_CLOFORK* on the *filde2* file  
 28505 descriptor shall be determined solely by the *flag* argument, which can be constructed from a  
 28506 bitwise-inclusive OR of flags from the following list:

28507 *O\_CLOEXEC*      Atomically set the *FD\_CLOEXEC* flag on *filde2*.

28508 *O\_CLOFORK*     Atomically set the *FD\_CLOFORK* flag on *filde2*.

28509 **TYP**      If *filde* refers to a typed memory object, the result of the *dup2()* or *dup3()* functions is  
 28510 unspecified.

28511 **RETURN VALUE**

28512 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned;  
 28513 otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

28514 **ERRORS**

28515 The *dup()* function shall fail if:

28516  $[\text{EBADF}]$       The *filde* argument is not a valid open file descriptor.

28517  $[\text{EMFILE}]$      All file descriptors available to the process are currently open.

28518 The *dup2()* and *dup3()* functions shall fail if:

28519  $[\text{EBADF}]$       The *filde* argument is not a valid open file descriptor or the argument *filde2* is  
 28520 negative or greater than or equal to  $\{\text{OPEN\_MAX}\}$ .

28521  $[\text{EINTR}]$       The function was interrupted by a signal.

28522 The *dup3()* function shall fail if:

- 28523 [EINVAL] The *fildest* and *fildest2* arguments are equal.
- 28524 The *dup2()* and *dup3()* functions may fail if:
- 28525 [EIO] An I/O error occurred while attempting to close *fildest2*.
- 28526 The *dup3()* function may fail if:
- 28527 [EINVAL] The value of the *flag* argument is invalid.

## 28528 EXAMPLES

### 28529 Redirecting Standard Output to a File

28530 The following example closes standard output for the current processes, re-assigns standard  
28531 output to go to the file referenced by *pfdest*, and closes the original file descriptor to clean up.

```
28532 #include <unistd.h>
28533 ...
28534 int pfdest;
28535 ...
28536 close(1);
28537 dup(pfdest);
28538 close(pfdest);
28539 ...
```

### 28540 Redirecting Error Messages

28541 The following example redirects messages from *stderr* to *stdout*.

```
28542 #include <unistd.h>
28543 ...
28544 dup2(1, 2);
28545 ...
```

## 28546 APPLICATION USAGE

28547 Implementations may use file descriptors that must be inherited into child processes for the  
28548 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,  
28549 an application that calls *dup2()* with an arbitrary integer for *fildest2* risks non-conforming  
28550 behavior, and *dup2()* can only portably be used to overwrite file descriptor values that the  
28551 application has obtained through explicit actions, or for the three file descriptors corresponding  
28552 to the standard file streams. In order to avoid a race condition of leaking an unintended file  
28553 descriptor into a child process or executed program, an application should consider opening all  
28554 file descriptors with the *FD\_CLOFORK* or *FD\_CLOEXEC* flag, or both flags, set unless the file  
28555 descriptor is intended to be inherited by child processes or executed programs, respectively.

## 28556 RATIONALE

28557 The *dup()* function is redundant. Its services are also provided by the *fcntl()* function. It has been  
28558 included in this volume of POSIX.1-2024 primarily for historical reasons, since many existing  
28559 applications use it. On the other hand, the *dup2()* function provides unique services, as no other  
28560 interface is able to atomically replace an existing file descriptor.

28561 The *dup2()* function is not marked obsolescent because it presents a type-safe version of  
28562 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

28563 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

28564 In the description of [EBADF], the case of *fildest* being out of range is covered by the given case of

28565 *fildev* not being valid. The descriptions for *fildev* and *fildev2* are different because the only kind of  
28566 invalidity that is relevant for *fildev2* is whether it is out of range; that is, it does not matter  
28567 whether *fildev2* refers to an open file when the *dup2()* call is made.

28568 The *dup3()* function with the `O_CLOEXEC` and `O_CLOFORK` flags is necessary to avoid a data  
28569 race in multi-threaded applications. Without `O_CLOFORK`, a file descriptor is leaked into a  
28570 child process created by one thread in the window between another thread creating a file  
28571 descriptor with *dup2()* and then using *fcntl()* to set the `FD_CLOFORK` flag. Without  
28572 `O_CLOEXEC`, a file descriptor intentionally inherited by child processes is similarly leaked into  
28573 an executed program if `FD_CLOEXEC` is not set atomically. The safe counterpart for avoiding  
28574 the same race with *dup()* is the use of the `F_DUPFD_CLOFORK` or `F_DUPFD_CLOEXEC` action  
28575 of the *fcntl()* function.

#### 28576 **FUTURE DIRECTIONS**

28577 None.

#### 28578 **SEE ALSO**

28579 *close()*, *fcntl()*, *open()*

28580 XBD <[unistd.h](#)>

#### 28581 **CHANGE HISTORY**

28582 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 28583 **Issue 7**

28584 SD5-XSH-ERN-187 is applied.

28585 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0075 [149,428] and  
28586 XSH/TC1-2008/0076 [149] are applied.

#### 28587 **Issue 8**

28588 Austin Group Defects 411, 1318, 1483, and 1577 are applied, adding *dup3()* and `FD_CLOFORK`.

28589 **NAME**

28590 duplocale — duplicate a locale object

28591 **SYNOPSIS**

```
28592 CX #include <locale.h>
28593 locale_t duplocale(locale_t locobj);
```

28594 **DESCRIPTION**

28595 The *duplocale()* function shall create a duplicate copy of the locale object referenced by the *locobj*  
 28596 argument.

28597 If the *locobj* argument is `LC_GLOBAL_LOCALE`, *duplocale()* shall create a new locale object  
 28598 containing a copy of the global locale determined by the *setlocale()* function.

28599 The behavior is undefined if the *locobj* argument is not a valid locale object handle.

28600 **RETURN VALUE**

28601 Upon successful completion, the *duplocale()* function shall return a handle for a new locale  
 28602 object. Otherwise, *duplocale()* shall return `(locale_t)0` and set *errno* to indicate the error.

28603 **ERRORS**

28604 The *duplocale()* function shall fail if:

28605 [ENOMEM] There is not enough memory available to create the locale object or load the  
 28606 locale data.

28607 **EXAMPLES**28608 **Constructing an Altered Version of an Existing Locale Object**

28609 The following example shows a code fragment to create a slightly altered version of an existing  
 28610 locale object. The function takes a locale object and a locale name and it replaces the `LC_TIME`  
 28611 category data in the locale object with that from the named locale.

```
28612 #include <locale.h>
28613 ...
28614 locale_t
28615 with_changed_lc_time (locale_t obj, const char *name)
28616 {
28617     locale_t retval = duplocale (obj);
28618     if (retval != (locale_t) 0)
28619     {
28620         locale_t changed = newlocale (LC_TIME_MASK, name, retval);
28621         if (changed == (locale_t) 0)
28622             /* An error occurred. Free all allocated resources. */
28623             freelocale (retval);
28624         retval = changed;
28625     }
28626     return retval;
28627 }
```

**28628 APPLICATION USAGE**

28629 The use of the *duplocale()* function is recommended for situations where a locale object is being  
28630 used in multiple places, and it is possible that the lifetime of the locale object might end before  
28631 all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is  
28632 needed. This can be achieved by a call to *newlocale()* following the *duplocale()* call.

28633 As with the *newlocale()* function, handles for locale objects created by the *duplocale()* function  
28634 should be released by a corresponding call to *freelocale()*.

28635 The *duplocale()* function can also be used in conjunction with *uselocale((locale\_t)0)*. This returns  
28636 the locale in effect for the calling thread, but can have the value LC\_GLOBAL\_LOCALE. Passing  
28637 LC\_GLOBAL\_LOCALE to functions such as *isalnum\_l()* results in undefined behavior, but  
28638 applications can convert it into a usable locale object by using *duplocale()*.

**28639 RATIONALE**

28640 None.

**28641 FUTURE DIRECTIONS**

28642 None.

**28643 SEE ALSO**

28644 *freelocale()*, *newlocale()*, *uselocale()*

28645 XBD <locale.h>

**28646 CHANGE HISTORY**

28647 First released in Issue 7.

28648 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0077 [283,301], XSH/TC1-2008/0078  
28649 [283], and XSH/TC1-2008/0079 [301] are applied.

28650 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0084 [753] is applied.

28651 **NAME**28652 encrypt — encoding function (**CRYPT**)28653 **SYNOPSIS**

```
28654 OB XSI #include <unistd.h>
28655 void encrypt(char block[64], int edflag);
```

28656 **DESCRIPTION**

28657 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm.  
 28658 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*.

28659 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes  
 28660 with values of 0 and 1. The array is modified in place to a similar array using the key set by  
 28661 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see  
 28662 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to  
 28663 [ENOSYS].

28664 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing  
 28665 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on  
 28666 return, an error has occurred.

28667 The *encrypt()* function need not be thread-safe.

28668 **RETURN VALUE**

28669 The *encrypt()* function shall not return a value.

28670 **ERRORS**

28671 The *encrypt()* function shall fail if:

28672 [ENOSYS] The functionality is not supported on this implementation.

28673 **EXAMPLES**

28674 None.

28675 **APPLICATION USAGE**

28676 The *encrypt()* function historically used the DES block cipher, which is no longer considered  
 28677 secure.

28678 In some environments, decoding might not be implemented. This is related to some Government  
 28679 restrictions on encryption and decryption routines. Historical practice has been to ship a  
 28680 different version of the encryption library without the decryption feature in the routines  
 28681 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

28682 **RATIONALE**

28683 None.

28684 **FUTURE DIRECTIONS**

28685 The *encrypt()* function may be removed in a future version.

28686 **SEE ALSO**

28687 [crypt\(\)](#), [setkey\(\)](#)

28688 XBD [<unistd.h>](#)

28689 **CHANGE HISTORY**

28690 First released in Issue 1. Derived from Issue 1 of the SVID.

- 28691 **Issue 5**  
28692 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 28693 **Issue 6**  
28694 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 28695 **Issue 7**  
28696 Austin Group Interpretation 1003.1-2001 #156 is applied.  
28697 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0085 [899] is applied.
- 28698 **Issue 8**  
28699 Austin Group Defect 1192 is applied, marking the *encrypt()* function as obsolescent.

28700 **NAME**

28701 endgrent, getgrent, setgrent — group database entry functions

28702 **SYNOPSIS**

```
28703 XSI #include <grp.h>
28704 void endgrent(void);
28705 struct group *getgrent(void);
28706 void setgrent(void);
```

28707 **DESCRIPTION**

28708 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an  
 28709 entry in the group database. If the group database is not already open, *getgrent()* shall open it  
 28710 and return a pointer to a **group** structure containing the first entry in the database. Thereafter, it  
 28711 shall return a pointer to a **group** structure containing the next **group** structure in the group  
 28712 database, so successive calls may be used to search the entire database.

28713 An implementation that provides extended security controls may impose further  
 28714 implementation-defined restrictions on accessing the group database. In particular, the system  
 28715 may deny the existence of some or all of the group database entries associated with groups other  
 28716 than those groups associated with the caller and may omit users other than the caller from the  
 28717 list of members of groups in database entries that are returned.

28718 The *setgrent()* function shall rewind the group database so that the next *getgrent()* call returns  
 28719 the first entry, allowing repeated searches.

28720 The *endgrent()* function shall close the group database.

28721 The *setgrent()* and *endgrent()* functions shall not change the setting of *errno* if successful.

28722 On error, the *setgrent()* and *endgrent()* functions shall set *errno* to indicate the error.

28723 Since no value is returned by the *setgrent()* and *endgrent()* functions, an application wishing to  
 28724 check for error situations should set *errno* to 0, then call the function, then check *errno*.

28725 These functions need not be thread-safe.

28726 **RETURN VALUE**

28727 On successful completion, *getgrent()* shall return a pointer to a **group** structure. On end-of-file,  
 28728 *getgrent()* shall return a null pointer and shall not change the setting of *errno*. On error,  
 28729 *getgrent()* shall return a null pointer and *errno* shall be set to indicate the error.

28730 The application shall not modify the structure to which the return value points, nor any storage  
 28731 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
 28732 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
 28733 subsequent call to *getgrgid()*, *getgrnam()*, or *getgrent()*. The returned pointer, and pointers  
 28734 within the structure, might also be invalidated if the calling thread is terminated.

28735 **ERRORS**

28736 These functions may fail if:

28737 [EINTR] A signal was caught during the operation.

28738 [EIO] An I/O error has occurred.

28739 In addition, the *getgrent()* and *setgrent()* functions may fail if:

28740 [EMFILE] All file descriptors available to the process are currently open.



- 28741 [ENFILE] The maximum allowable number of files is currently open in the system.
- 28742 **EXAMPLES**
- 28743 None.
- 28744 **APPLICATION USAGE**
- 28745 These functions are provided due to their historical usage. Applications should avoid
- 28746 dependencies on fields in the group database, whether the database is a single file, or where in
- 28747 the file system name space the database resides. Applications should use *getgrnam()* and
- 28748 *getgrgid()* whenever possible because it avoids these dependencies.
- 28749 **RATIONALE**
- 28750 None.
- 28751 **FUTURE DIRECTIONS**
- 28752 None.
- 28753 **SEE ALSO**
- 28754 *endpwent()*, *getgrgid()*, *getgrnam()*, *getlogin()*
- 28755 XBD <grp.h>
- 28756 **CHANGE HISTORY**
- 28757 First released in Issue 4, Version 2.
- 28758 **Issue 5**
- 28759 Moved from X/OPEN UNIX extension to BASE.
- 28760 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
- 28761 VALUE section.
- 28762 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.
- 28763 **Issue 6**
- 28764 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 28765 **Issue 7**
- 28766 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 28767 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 28768 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0080 [75] is applied.
- 28769 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0086 [493], XSH/TC2-2008/0087 [656],
- 28770 and XSH/TC2-2008/0088 [493] are applied.

28771 **NAME**

28772 endhostent, gethostent, sethostent — network host database functions

28773 **SYNOPSIS**

```
28774 #include <netdb.h>
28775 void endhostent(void);
28776 struct hostent *gethostent(void);
28777 void sethostent(int stayopen);
```

28778 **DESCRIPTION**

28779 These functions shall retrieve information about hosts. This information is considered to be  
28780 stored in a database that can be accessed sequentially or randomly. The implementation of this  
28781 database is unspecified.

28782 **Note:** In many cases this database is implemented by the Domain Name System, as documented in  
28783 RFC 1034, RFC 1035, and RFC 3596.

28784 The *sethostent()* function shall open a connection to the database and set the next entry for  
28785 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection  
28786 shall not be closed by a call to *gethostent()*, and the implementation may maintain an open file  
28787 descriptor. If a file descriptor is opened, the FD\_CLOEXEC flag shall be set; see <fcntl.h>.

28788 The *gethostent()* function shall read the next entry in the database, opening and closing a  
28789 connection to the database as necessary.

28790 Entries shall be returned in **hostent** structures.

28791 The *endhostent()* function shall close the connection to the database, releasing any open file  
28792 descriptor.

28793 These functions need not be thread-safe.

28794 **RETURN VALUE**

28795 Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent**  
28796 structure if the requested entry was found, and a null pointer if the end of the database was  
28797 reached or the requested entry was not found.

28798 The application shall not modify the structure to which the return value points, nor any storage  
28799 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
28800 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
28801 subsequent call to *gethostent()*. The returned pointer, and pointers within the structure, might  
28802 also be invalidated if the calling thread is terminated.

28803 **ERRORS**

28804 The *gethostent()* and *sethostent()* functions may fail if:

28805 [EMFILE] All file descriptors available to the process are currently open.

28806 [ENFILE] The maximum allowable number of files is currently open in the system.

28807 **EXAMPLES**

28808 None.

28809 **APPLICATION USAGE**

28810 None.

28811 **RATIONALE**

28812 None.

28813 **FUTURE DIRECTIONS**

28814 None.

28815 **SEE ALSO**28816 *endservent()*28817 XBD [<fcntl.h>](#), [<netdb.h>](#)28818 **CHANGE HISTORY**

28819 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

28820 **Issue 7**

28821 Austin Group Interpretation 1003.1-2001 #156 is applied.

28822 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0081 [75,428] and  
28823 XSH/TC1-2008/0082 [75] are applied.

28824 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0089 [656] is applied.

28825 **Issue 8**28826 Austin Group Defect 368 is applied, adding a requirement for FD\_CLOEXEC to be set if a file  
28827 descriptor is opened, and adding the [EMFILE] and [ENFILE] errors.

28828 Austin Group Defect 1685 is applied, updating RFC references.

28829 **NAME**

28830 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

28831 **SYNOPSIS**

```
28832 #include <netdb.h>
28833 void endnetent(void);
28834 struct netent *getnetbyaddr(uint32_t net, int type);
28835 struct netent *getnetbyname(const char *name);
28836 struct netent *getnetent(void);
28837 void setnetent(int stayopen);
```

28838 **DESCRIPTION**

28839 These functions shall retrieve information about networks. This information is considered to be  
 28840 stored in a database that can be accessed sequentially or randomly. The implementation of this  
 28841 database is unspecified.

28842 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-  
 28843 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either  
 28844 directly, or indirectly through one of the other *getnet\**(*)* functions), and the implementation may  
 28845 maintain an open file descriptor to the database. If a file descriptor is used, the *FD\_CLOEXEC*  
 28846 flag shall be set; see <*fcntl.h*>.

28847 The *getnetent()* function shall read the next entry of the database, opening and closing a  
 28848 connection to the database as necessary.

28849 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry  
 28850 for which the address family specified by *type* matches the *n\_addrtype* member and the network  
 28851 number *net* matches the *n\_net* member, opening and closing a connection to the database as  
 28852 necessary. The *net* argument shall be the network number in host byte order.

28853 The *getnetbyname()* function shall search the database from the beginning and find the first entry  
 28854 for which the network name specified by *name* matches the *n\_name* member, opening and  
 28855 closing a connection to the database as necessary.

28856 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions shall each return a pointer to a  
 28857 **netent** structure, the members of which shall contain the fields of an entry in the network  
 28858 database.

28859 The *endnetent()* function shall close the database, releasing any open file descriptor.

28860 These functions need not be thread-safe.

28861 **RETURN VALUE**

28862 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* shall return a  
 28863 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the  
 28864 database was reached or the requested entry was not found. Otherwise, a null pointer shall be  
 28865 returned.

28866 The application shall not modify the structure to which the return value points, nor any storage  
 28867 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
 28868 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
 28869 subsequent call to *getnetbyaddr()*, *getnetbyname()*, or *getnetent()*. The returned pointer, and  
 28870 pointers within the structure, might also be invalidated if the calling thread is terminated.

**28871 ERRORS**

28872 The `getnetbyaddr()`, `getnetbyname()`, `getnetent()`, and `setnetent()` functions may fail if:

28873 [EMFILE] All file descriptors available to the process are currently open.

28874 [ENFILE] The maximum allowable number of files is currently open in the system.

**28875 EXAMPLES**

28876 None.

**28877 APPLICATION USAGE**

28878 None.

**28879 RATIONALE**

28880 None.

**28881 FUTURE DIRECTIONS**

28882 None.

**28883 SEE ALSO**

28884 XBD [<fcntl.h>](#), [<netdb.h>](#)

**28885 CHANGE HISTORY**

28886 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

**28887 Issue 7**

28888 Austin Group Interpretation 1003.1-2001 #156 is applied.

28889 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0083 [75] and XSH/TC1-2008/0084 [75] are applied.

28891 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0090 [656] is applied.

**28892 Issue 8**

28893 Austin Group Defect 368 is applied, adding a requirement for `FD_CLOEXEC` to be set if a file descriptor is used, and adding the [EMFILE] and [ENFILE] errors.

28894

28895 **NAME**

28896 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol  
28897 database functions

28898 **SYNOPSIS**

```
28899 #include <netdb.h>
28900 void endprotoent(void);
28901 struct protoent *getprotobyname(const char *name);
28902 struct protoent *getprotobynumber(int proto);
28903 struct protoent *getprotoent(void);
28904 void setprotoent(int stayopen);
```

28905 **DESCRIPTION**

28906 These functions shall retrieve information about protocols. This information is considered to be  
28907 stored in a database that can be accessed sequentially or randomly. The implementation of this  
28908 database is unspecified.

28909 The *setprotoent()* function shall open a connection to the database, and set the next entry to the  
28910 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database  
28911 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the  
28912 other *getproto\**(*)* functions), and the implementation may maintain an open file descriptor for  
28913 the database. If a file descriptor is used, the FD\_CLOEXEC flag shall be set; see <fcntl.h>.

28914 The *getprotobyname()* function shall search the database from the beginning and find the first  
28915 entry for which the protocol name specified by *name* matches the *p\_name* member, opening and  
28916 closing a connection to the database as necessary.

28917 The *getprotobynumber()* function shall search the database from the beginning and find the first  
28918 entry for which the protocol number specified by *proto* matches the *p\_proto* member, opening  
28919 and closing a connection to the database as necessary.

28920 The *getprotoent()* function shall read the next entry of the database, opening and closing a  
28921 connection to the database as necessary.

28922 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions shall each return a pointer  
28923 to a **protoent** structure, the members of which shall contain the fields of an entry in the network  
28924 protocol database.

28925 The *endprotoent()* function shall close the connection to the database, releasing any open file  
28926 descriptor.

28927 These functions need not be thread-safe.

28928 **RETURN VALUE**

28929 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a  
28930 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of  
28931 the database was reached or the requested entry was not found. Otherwise, a null pointer is  
28932 returned.

28933 The application shall not modify the structure to which the return value points, nor any storage  
28934 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
28935 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
28936 subsequent call to *getprotobyname()*, *getprotobynumber()*, or *getprotoent()*. The returned pointer,  
28937 and pointers within the structure, might also be invalidated if the calling thread is terminated.

**28938 ERRORS**

28939 The *getprotobyname()*, *getprotobynumber()*, *getprotoent()*, and *setprotoent()* functions may fail if:

28940 [EMFILE] All file descriptors available to the process are currently open.

28941 [ENFILE] The maximum allowable number of files is currently open in the system.

**28942 EXAMPLES**

28943 None.

**28944 APPLICATION USAGE**

28945 None.

**28946 RATIONALE**

28947 None.

**28948 FUTURE DIRECTIONS**

28949 None.

**28950 SEE ALSO**

28951 XBD [<fcntl.h>](#), [<netdb.h>](#)

**28952 CHANGE HISTORY**

28953 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

**28954 Issue 7**

28955 Austin Group Interpretation 1003.1-2001 #156 is applied.

28956 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0085 [75] and XSH/TC1-2008/0086 [75] are applied.

28958 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0091 [656] is applied.

**28959 Issue 8**

28960 Austin Group Defect 368 is applied, adding a requirement for FD\_CLOEXEC to be set if a file descriptor is used, and adding the [EMFILE] and [ENFILE] errors.

28961

28962 **NAME**

28963 endpwent, getpwent, setpwent — user database functions

28964 **SYNOPSIS**

```
28965 XSI #include <pwd.h>
28966 void endpwent(void);
28967 struct passwd *getpwent(void);
28968 void setpwent(void);
```

28969 **DESCRIPTION**

28970 These functions shall retrieve information about users.

28971 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of  
 28972 an entry in the user database. Each entry in the user database contains a **passwd** structure. If the  
 28973 user database is not already open, *getpwent()* shall open it and return a pointer to a **passwd**  
 28974 structure containing the first entry in the database. Thereafter, it shall return a pointer to a  
 28975 **passwd** structure containing the next entry in the user database. Successive calls can be used to  
 28976 search the entire user database.

28977 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

28978 An implementation that provides extended security controls may impose further  
 28979 implementation-defined restrictions on accessing the user database. In particular, the system  
 28980 may deny the existence of some or all of the user database entries associated with users other  
 28981 than the caller.

28982 The *setpwent()* function shall rewind the user database so that the next *getpwent()* call returns  
 28983 the first entry, allowing repeated searches.

28984 The *endpwent()* function shall close the user database.28985 The *setpwent()* and *endpwent()* functions shall not change the setting of *errno* if successful.28986 On error, the *setpwent()* and *endpwent()* functions shall set *errno* to indicate the error.

28987 Since no value is returned by the *setpwent()* and *endpwent()* functions, an application wishing to  
 28988 check for error situations should set *errno* to 0, then call the function, then check *errno*.

28989 These functions need not be thread-safe.

28990 **RETURN VALUE**

28991 On successful completion, *getpwent()* shall return a pointer to a **passwd** structure. On end-of-  
 28992 file, *getpwent()* shall return a null pointer and shall not change the setting of *errno*. On error,  
 28993 *getpwent()* shall return a null pointer and *errno* shall be set to indicate the error.

28994 The application shall not modify the structure to which the return value points, nor any storage  
 28995 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
 28996 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
 28997 subsequent call to *getpwuid()*, *getpwnam()*, or *getpwent()*. The returned pointer, and pointers  
 28998 within the structure, might also be invalidated if the calling thread is terminated.

28999 **ERRORS**

29000 These functions may fail if:

29001 [EINTR] A signal was caught during the operation.

29002 [EIO] An I/O error has occurred.

29003 In addition, *getpwent()* and *setpwent()* may fail if:



- 29004 [EMFILE] All file descriptors available to the process are currently open.
- 29005 [ENFILE] The maximum allowable number of files is currently open in the system.

## 29006 EXAMPLES

### 29007 Searching the User Database

29008 The following example uses the *getpwent()* function to get successive entries in the user  
 29009 database, returning a pointer to a **passwd** structure that contains information about each user.  
 29010 The call to *endpwent()* closes the user database and cleans up.

```

29011 #include <pwd.h>
29012 #include <stdio.h>

29013 void printname(uid_t uid)
29014 {
29015     struct passwd *pwd;

29016     setpwent();
29017     while((pwd = getpwent()) != NULL) {
29018         if (pwd->pw_uid == uid) {
29019             printf("name=%s\n", pwd->pw_name);
29020             break;
29021         }
29022     }
29023     endpwent();
29024 }
```

## 29025 APPLICATION USAGE

29026 These functions are provided due to their historical usage. Applications should avoid  
 29027 dependencies on fields in the password database, whether the database is a single file, or where  
 29028 in the file system name space the database resides. Applications should use *getpwuid()*  
 29029 whenever possible because it avoids these dependencies.

## 29030 RATIONALE

29031 None.

## 29032 FUTURE DIRECTIONS

29033 None.

## 29034 SEE ALSO

29035 [endgrent\(\)](#), [getlogin\(\)](#), [getpwnam\(\)](#), [getpwuid\(\)](#)

29036 XBD [<pwd.h>](#)

## 29037 CHANGE HISTORY

29038 First released in Issue 4, Version 2.

### 29039 Issue 5

29040 Moved from X/OPEN UNIX extension to BASE.

29041 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 29042 VALUE section.

29043 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

29044 **Issue 6**

29045 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

29046 **Issue 7**

29047 Austin Group Interpretation 1003.1-2001 #156 is applied.

29048 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

29049 The EXAMPLES section is revised.

29050 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0087 [75] is applied.

29051 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0092 [493], XSH/TC2-2008/0093 [656],  
29052 and XSH/TC2-2008/0094 [493] are applied.

29053 **NAME**

29054 endservent, getservbyname, getservbyport, getservent, setservent — network services database  
29055 functions

29056 **SYNOPSIS**

```
29057 #include <netdb.h>
29058 void endservent(void);
29059 struct servent *getservbyname(const char *name, const char *proto);
29060 struct servent *getservbyport(int port, const char *proto);
29061 struct servent *getservent(void);
29062 void setservent(int stayopen);
```

29063 **DESCRIPTION**

29064 These functions shall retrieve information about network services. This information is  
29065 considered to be stored in a database that can be accessed sequentially or randomly. The  
29066 implementation of this database is unspecified.

29067 The *setservent()* function shall open a connection to the database, and set the next entry to the  
29068 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each  
29069 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv\**(  
29070 functions), and the implementation may maintain an open file descriptor for the database. If a  
29071 file descriptor is used, the FD\_CLOEXEC flag shall be set; see <fcntl.h>.

29072 The *getservent()* function shall read the next entry of the database, opening and closing a  
29073 connection to the database as necessary.

29074 The *getservbyname()* function shall search the database from the beginning and find the first  
29075 entry for which the service name specified by *name* matches the *s\_name* member and the protocol  
29076 name specified by *proto* matches the *s\_proto* member, opening and closing a connection to the  
29077 database as necessary. If *proto* is a null pointer, any value of the *s\_proto* member shall be  
29078 matched.

29079 The *getservbyport()* function shall search the database from the beginning and find the first entry  
29080 for which the port specified by *port* matches the *s\_port* member and the protocol name specified  
29081 by *proto* matches the *s\_proto* member, opening and closing a connection to the database as  
29082 necessary. If *proto* is a null pointer, any value of the *s\_proto* member shall be matched. The *port*  
29083 argument shall be a value obtained by converting a **uint16\_t** in network byte order to **int**.

29084 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a  
29085 **servent** structure, the members of which shall contain the fields of an entry in the network  
29086 services database.

29087 The *endservent()* function shall close the database, releasing any open file descriptor.

29088 These functions need not be thread-safe.

29089 **RETURN VALUE**

29090 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to  
29091 a **servent** structure if the requested entry was found, and a null pointer if the end of the database  
29092 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

29093 The application shall not modify the structure to which the return value points, nor any storage  
29094 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
29095 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
29096 subsequent call to *getservbyname()*, *getservbyport()*, or *getservent()*. The returned pointer, and  
29097 pointers within the structure, might also be invalidated if the calling thread is terminated.

29098 **ERRORS**29099 The *getserbyname()*, *getserbyport()*, *getservent()*, and *setservent()* functions may fail if:

29100 [EMFILE] All file descriptors available to the process are currently open.

29101 [ENFILE] The maximum allowable number of files is currently open in the system.

29102 **EXAMPLES**

29103 None.

29104 **APPLICATION USAGE**29105 The *port* argument of *getserbyport()* need not be compatible with the port values of all address  
29106 families.29107 **RATIONALE**

29108 None.

29109 **FUTURE DIRECTIONS**

29110 None.

29111 **SEE ALSO**29112 *endhostent()*, *endprotoent()*, *htonl()*, *inet\_addr()*29113 XBD <[fcntl.h](#)>, <[netdb.h](#)>29114 **CHANGE HISTORY**

29115 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

29116 **Issue 7**

29117 Austin Group Interpretation 1003.1-2001 #156 is applied.

29118 SD5-XBD-ERN-14 is applied.

29119 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0088 [75] and XSH/TC1-2008/0089  
29120 [75] are applied.

29121 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0095 [656] is applied.

29122 **Issue 8**29123 Austin Group Defect 368 is applied, adding a requirement for `FD_CLOEXEC` to be set if a file  
29124 descriptor is used, and adding the [EMFILE] and [ENFILE] errors.

29125 **NAME**

29126 endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database  
 29127 functions

29128 **SYNOPSIS**

```
29129 XSI #include <utmpx.h>
29130 void endutxent(void);
29131 struct utmpx *getutxent(void);
29132 struct utmpx *getutxid(const struct utmpx *id);
29133 struct utmpx *getutxline(const struct utmpx *line);
29134 struct utmpx *pututxline(const struct utmpx *utmpx);
29135 void setutxent(void);
```

29136 **DESCRIPTION**

29137 These functions shall provide access to the user accounting database.

29138 The *getutxent()* function shall read the next entry from the user accounting database. If the  
 29139 database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

29140 The *getutxid()* function shall search forward from the current point in the database. If the  
 29141 *ut\_type* value of the **utmpx** structure pointed to by *id* is *BOOT\_TIME*, *OLD\_TIME*, or  
 29142 *NEW\_TIME*, then it shall stop when it finds an entry with a matching *ut\_type* value. If the  
 29143 *ut\_type* value is *INIT\_PROCESS*, *LOGIN\_PROCESS*, *USER\_PROCESS*, or *DEAD\_PROCESS*,  
 29144 then it shall stop when it finds an entry whose type is one of these four and whose *ut\_id* member  
 29145 matches the *ut\_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is  
 29146 reached without a match, *getutxid()* shall fail.

29147 The *getutxline()* function shall search forward from the current point in the database until it  
 29148 finds an entry of the type *LOGIN\_PROCESS* or *USER\_PROCESS* which also has a *ut\_line* value  
 29149 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached  
 29150 without a match, *getutxline()* shall fail.

29151 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to  
 29152 search for multiple occurrences, the application shall zero out the static data after each success,  
 29153 or *getutxline()* may return a pointer to the same **utmpx** structure.

29154 There is one exception to the rule about clearing the structure before further reads are done. The  
 29155 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user  
 29156 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or  
 29157 *getutxline()*, if the application has modified this structure and passed the pointer back to  
 29158 *pututxline()*.

29159 For all entries that match a request, the *ut\_type* member indicates the type of the entry. Other  
 29160 members of the entry shall contain meaningful data based on the value of the *ut\_type* member as  
 29161 follows:

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	<i>ut_tv</i>
OLD_TIME	<i>ut_tv</i>
NEW_TIME	<i>ut_tv</i>
USER_PROCESS	<i>ut_id</i> , <i>ut_user</i> (login name of the user), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
INIT_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>
LOGIN_PROCESS	<i>ut_id</i> , <i>ut_user</i> (implementation-defined name of the login process), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
DEAD_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

If the process has appropriate privileges, the *pututxline()* function shall write out the structure into the user accounting database. It shall search for a record as if by *getutxid()* that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

The *endutxent()* function shall close the user accounting database.

The *setutxent()* function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

These functions need not be thread-safe.

#### RETURN VALUE

Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

The return value may point to a static area which is overwritten by a subsequent call to *getutxid()* or *getutxline()*.

Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

The *endutxent()* and *setutxent()* functions shall not return a value.

#### ERRORS

No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()* functions.

The *pututxline()* function may fail if:

[EPERM]           The process does not have appropriate privileges.

29199 **EXAMPLES**

29200 None.

29201 **APPLICATION USAGE**29202 The sizes of the arrays in the structure can be found using the *sizeof* operator.29203 **RATIONALE**

29204 None.

29205 **FUTURE DIRECTIONS**

29206 None.

29207 **SEE ALSO**29208 XBD [<utmpx.h>](#)29209 **CHANGE HISTORY**

29210 First released in Issue 4, Version 2.

29211 **Issue 5**

29212 Moved from X/OPEN UNIX extension to BASE.

29213 Normative text previously in the APPLICATION USAGE section is moved to the  
29214 DESCRIPTION.

29215 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

29216 **Issue 6**

29217 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

29218 **Issue 7**

29219 Austin Group Interpretation 1003.1-2001 #156 is applied.

29220 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0090 [213,428] and  
29221 XSH/TC1-2008/0091 [213] are applied.

29222 **NAME**29223            **environ** — array of character pointers to the environment strings29224 **SYNOPSIS**29225            extern char \*\***environ**;29226 **DESCRIPTION**29227            Refer to *exec* and XBD [Chapter 8](#) (on page 167).



29228 **NAME**

29229 erand48 — generate uniformly distributed pseudo-random numbers

29230 **SYNOPSIS**29231 XSI `#include <stdlib.h>`29232 `double erand48(unsigned short xsubi[3]);`29233 **DESCRIPTION**29234 Refer to *drand48()*.

29235 **NAME**

29236 erf, erff, erfl — error functions

29237 **SYNOPSIS**

```
29238 #include <math.h>
29239 double erf(double x);
29240 float erff(float x);
29241 long double erfl(long double x);
```

29242 **DESCRIPTION**

29243 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 29244 conflict between the requirements described here and the ISO C standard is unintentional. This  
 29245 volume of POSIX.1-2024 defers to the ISO C standard.

29246 These functions shall compute the error function of their argument  $x$ , defined as:

$$29247 \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

29248 An application wishing to check for error situations should set *errno* to zero and call  
 29249 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 29250 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 29251 zero, an error has occurred.

29252 **RETURN VALUE**

29253 Upon successful completion, these functions shall return the value of the error function.

29254 MX If  $x$  is NaN, a NaN shall be returned.

29255 If  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.

29256 If  $x$  is  $\pm \text{Inf}$ ,  $\pm 1$  shall be returned.

29257 If the correct value would cause underflow, a range error may occur, and *erf()*, *erff()*, and *erfl()*  
 29258 shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
 29259 MXX FLT\_MIN, and LDBL\_MIN, respectively. If the IEC 60559 Floating-Point option is supported,  
 29260  $2 * x / \text{sqrt}(\pi)$  should be returned.

29261 **ERRORS**

29262 These functions may fail if:

29263 MX **Range Error** The result underflows.

29264 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 29265 then *errno* shall be set to [ERANGE]. If the integer expression  
 29266 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 29267 floating-point exception shall be raised.

29268 **EXAMPLES**29269 **Computing the Probability for a Normal Variate**

29270 This example shows how to use *erf()* to compute the probability that a normal variate assumes a  
29271 value in the range  $[x_1, x_2]$  with  $x_1 \leq x_2$ .

29272 This example uses the constant `M_SQRT1_2` which is part of the XSI option.

```
29273 #include <math.h>
29274 double
29275 Phi(const double x1, const double x2)
29276 {
29277     return ( erf(x2*M_SQRT1_2) - erf(x1*M_SQRT1_2) ) / 2;
29278 }
```

29279 **APPLICATION USAGE**

29280 Underflow occurs when  $|x| < \text{DBL\_MIN} * (\text{sqrt}(\pi)/2)$ .

29281 On error, the expressions (*math\_errhandling* & `MATH_ERRNO`) and (*math\_errhandling* &  
29282 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

29283 **RATIONALE**

29284 None.

29285 **FUTURE DIRECTIONS**

29286 None.

29287 **SEE ALSO**

29288 [\*erfc\(\)\*](#), [\*feclearexcept\(\)\*](#), [\*fetestexcept\(\)\*](#), [\*isnan\(\)\*](#)

29289 XBD Section 4.23 (on page 109), [`<math.h>`](#)

29290 **CHANGE HISTORY**

29291 First released in Issue 1. Derived from Issue 1 of the SVID.

29292 **Issue 5**

29293 The DESCRIPTION is updated to indicate how an application should check for an error. This  
29294 text was previously published in the APPLICATION USAGE section.

29295 **Issue 6**

29296 The *erf()* function is no longer marked as an extension.

29297 The *erfc()* function is split out onto its own reference page.

29298 The *erfff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

29299 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
29300 revised to align with the ISO/IEC 9899:1999 standard.

29301 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
29302 marked.

29303 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/22 is applied, adding the example to the  
29304 EXAMPLES section.

29305 **Issue 7**

29306 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0092 [68] is applied.

29307 **Issue 8**  
29308

Austin Group Defect 1178 is applied, joining two paragraphs in the RETURN VALUE section.

29309 **NAME**

29310 erfc, erfcf, erfcl — complementary error functions

29311 **SYNOPSIS**

```
29312 #include <math.h>
29313 double erfc(double x);
29314 float erfcf(float x);
29315 long double erfcl(long double x);
```

29316 **DESCRIPTION**

29317 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 29318 conflict between the requirements described here and the ISO C standard is unintentional. This  
 29319 volume of POSIX.1-2024 defers to the ISO C standard.

29320 These functions shall compute the complementary error function  $1.0 - \text{erf}(x)$ .

29321 An application wishing to check for error situations should set *errno* to zero and call  
 29322 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 29323 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 29324 zero, an error has occurred.

29325 **RETURN VALUE**

29326 Upon successful completion, these functions shall return the value of the complementary error  
 29327 function.

29328 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
 29329 MXX and *erfc()*, *erfcf()*, and *erfcl()* shall return 0.0, or (if the IEC 60559 Floating-Point option is not  
 29330 supported) an implementation-defined value no greater in magnitude than DBL\_MIN,  
 29331 FLT\_MIN, and LDBL\_MIN, respectively.

29332 MX If *x* is NaN, a NaN shall be returned.

29333 If *x* is  $\pm 0$ , +1 shall be returned.

29334 If *x* is  $-\text{Inf}$ , +2 shall be returned.

29335 If *x* is  $+\text{Inf}$ , +0 shall be returned.

29336 MXX If the correct value would cause underflow and is representable, a range error may occur and  
 29337 the correct value shall be returned.

29338 **ERRORS**

29339 These functions may fail if:

29340 Range Error The result underflows.

29341 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 29342 then *errno* shall be set to [ERANGE]. If the integer expression  
 29343 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 29344 floating-point exception shall be raised.

29345 **EXAMPLES**

29346 None.

29347 **APPLICATION USAGE**

29348 The *erfc()* function is provided because of the extreme loss of relative accuracy if *erf(x)* is called  
29349 for large *x* and the result subtracted from 1.0.

29350 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
29351 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

29352 **RATIONALE**

29353 None.

29354 **FUTURE DIRECTIONS**

29355 None.

29356 **SEE ALSO**29357 *erf()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

29358 XBD Section 4.23 (on page 109), &lt;math.h&gt;

29359 **CHANGE HISTORY**

29360 First released in Issue 1. Derived from Issue 1 of the SVID.

29361 **Issue 5**

29362 The DESCRIPTION is updated to indicate how an application should check for an error. This  
29363 text was previously published in the APPLICATION USAGE section.

29364 **Issue 6**29365 The *erfc()* function is no longer marked as an extension.29366 These functions are split out from the *erf()* reference page.

29367 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
29368 revised to align with the ISO/IEC 9899:1999 standard.

29369 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
29370 marked.

29371 **Issue 7**

29372 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0093 [68] and XSH/TC1-2008/0094  
29373 [68] are applied.

29374 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0096 [630] is applied.

29375 **NAME**29376 `erff, erfl` — error functions29377 **SYNOPSIS**29378 `#include <math.h>`29379 `float erff(float x);`29380 `long double erfl(long double x);`29381 **DESCRIPTION**29382 Refer to *erf()*.

29383 **NAME**

29384           errno — error return value

29385 **SYNOPSIS**

29386           #include &lt;errno.h&gt;

29387 **DESCRIPTION**29388           The lvalue to which the macro *errno* expands is used by many functions to return error values.

29389           Many functions provide an error number in *errno*, which has type **int** and is defined in  
29390           <**errno.h**>. The value of *errno* in the initial thread shall be zero at program startup (the initial  
29391           value of *errno* in other threads is an indeterminate value) and shall otherwise be defined only  
29392           after a call to a function for which it is explicitly stated to be set and until it is changed by the  
29393           next function call or if the application assigns it a value. The value of *errno* should only be  
29394           examined when it is indicated to be valid by a function's return value. Applications shall obtain  
29395           the definition of *errno* by the inclusion of <**errno.h**>. No function in this volume of  
29396           POSIX.1-2024 shall set *errno* to 0. The setting of *errno* after a successful call to a function is  
29397           unspecified unless the description of that function specifies that *errno* shall not be modified.

29398           If the macro definition is suppressed in order to access an actual object, or a program defines an  
29399           identifier with the name *errno*, the behavior is undefined.

29400           The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant  
29401           pages.

29402 **RETURN VALUE**

29403           None.

29404 **ERRORS**

29405           None.

29406 **EXAMPLES**

29407           None.

29408 **APPLICATION USAGE**

29409           Previously both POSIX and X/Open documents were more restrictive than the ISO C standard  
29410           in that they required *errno* to be defined as an external variable, whereas the ISO C standard  
29411           required only that *errno* be defined as a modifiable lvalue with type **int**.

29412           An application that needs to examine the value of *errno* to determine the error should set it to 0  
29413           before a function call, then inspect it before a subsequent function call.

29414 **RATIONALE**

29415           None.

29416 **FUTURE DIRECTIONS**

29417           None.

29418 **SEE ALSO**29419           [Section 2.3](#)29420           XBD <**errno.h**>29421 **CHANGE HISTORY**

29422           First released in Issue 1. Derived from Issue 1 of the SVID.

29423 **Issue 5**

29424           The following sentence is deleted from the DESCRIPTION: “The value of *errno* is 0 at program  
29425           start-up, but is never set to 0 by any XSI function”.



29426 The DESCRIPTION also no longer states that conforming implementations may support the  
29427 declaration:

```
29428 extern int errno;
```

29429 **Issue 6**

29430 Obsolescent text regarding defining *errno* as:

```
29431 extern int errno
```

29432 is removed.

29433 Text regarding no function setting *errno* to zero to indicate an error is changed to no function  
29434 shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

29435 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/23 is applied, adding text to the  
29436 DESCRIPTION stating that the setting of *errno* after a successful call to a function is unspecified  
29437 unless the description of the function requires that it will not be modified.

29438 **Issue 8**

29439 Austin Group Defect 1302 is applied, aligning this macro with the ISO/IEC 9899:2018 standard.

29440 **NAME**

29441 environ, execl, execl, execlp, execv, execve, execvp, fexecve — execute a file

29442 **SYNOPSIS**

```

29443 #include <unistd.h>
29444 extern char **environ;
29445 int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
29446 int execl(const char *path, const char *arg0, ... /*,
29447         (char *)0, char *const envp[]*/);
29448 int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
29449 int execv(const char *path, char *const argv[]);
29450 int execve(const char *path, char *const argv[], char *const envp[]);
29451 int execvp(const char *file, char *const argv[]);
29452 int fexecve(int fd, char *const argv[], char *const envp[]);

```

29453 **DESCRIPTION**

29454 The *exec* family of functions shall replace the current process image with a new process image.  
 29455 The new image shall be constructed from a regular, executable file called the *new process image*  
 29456 *file*. There shall be no return from a successful *exec*, because the calling process image is overlaid  
 29457 by the new process image.

29458 The *fexecve()* function shall be equivalent to the *execve()* function except that the file to be  
 29459 executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is  
 29460 ignored.

29461 When a C-language program is executed as a result of a call to one of the *exec* family of  
 29462 functions, it shall be entered as a C-language function call as follows:

```
29463 int main (int argc, char *argv[]);
```

29464 where *argc* is the argument count and *argv* is an array of character pointers to the arguments  
 29465 themselves. In addition, the following variable, which needs to be declared by the user if it is to  
 29466 be used directly:

```
29467 extern char **environ;
```

29468 is initialized as a pointer to an array of character pointers to the environment strings. The *argv*  
 29469 and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv*  
 29470 array is not counted in *argc*.

29471 Applications can change the entire environment in a single operation by assigning the *environ*  
 29472 variable to point to an array of character pointers to the new environment strings. After  
 29473 assigning a new value to *environ*, applications should not rely on the new environment strings  
 29474 XSI remaining part of the environment, as a call to *getenv()*, *secure\_getenv()*, *putenv()*, *setenv()*,  
 29475 *unsetenv()*, or any function that is dependent on an environment variable may, on noticing that  
 29476 *environ* has changed, copy the environment strings to a new array and assign *environ* to point to  
 29477 it.

29478 Any application that directly modifies the pointers to which the *environ* variable points has  
 29479 undefined behavior.

29480 Conforming multi-threaded applications shall not use the *environ* variable to access or modify  
 29481 any environment variable while any other thread is concurrently modifying any environment  
 29482 variable. A call to any function dependent on any environment variable shall be considered a  
 29483 use of the *environ* variable to access that environment variable.

29484 The arguments specified by a program with one of the *exec* functions shall be passed on to the  
 29485 new process image in the corresponding *main()* arguments.

29486 For the *execl()*, *execle()*, *execv()*, and *execve()* functions, the argument *path* points to a pathname  
29487 that identifies the new process image file.

29488 For the *execlp()* and *execvp()* functions, the argument *file* is used to construct a pathname that  
29489 identifies the new process image file. If the *file* argument contains a <slash> character, the *file*  
29490 argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is  
29491 obtained by a search of the directories passed as the environment variable *PATH* (see XBD  
29492 Chapter 8, on page 167). If this environment variable is not present, the results of the search are  
29493 implementation-defined.

29494 There are two distinct ways in which the contents of the process image file may cause the  
29495 execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the  
29496 ERRORS section). In the cases where the other members of the *exec* family of functions would  
29497 fail and set *errno* to [ENOEXEC], the *execlp()* and *execvp()* functions shall execute a command  
29498 interpreter and the environment of the executed command shall be as if the process invoked the  
29499 *sh* utility using *execl()* as follows:

```
29500 execl(<shell path>, <name>, file, <args>, (char *)0);
```

29501 where <shell path> is an unspecified pathname for the *sh* utility, <name> is an unspecified string,  
29502 *file* is the process image file, and where <args> is zero or more parameters corresponding to any  
29503 initial non-null arguments passed after *argv0* for *execlp()* or to any initial non-null members of  
29504 *argv* starting at *argv[1]* for *execvp()*.

29505 The arguments represented by *argv0,...* are pointers to null-terminated character strings. These  
29506 strings shall constitute the argument list available to the new process image. The list is  
29507 terminated by a null pointer. The argument *argv0* should point to a filename string that is  
29508 associated with the process being started by one of the *exec* functions.

29509 The argument *argv* is an array of character pointers to null-terminated strings. The application  
29510 shall ensure that the last member of this array is a null pointer. These strings shall constitute the  
29511 argument list available to the new process image. The value in *argv[0]* should point to a filename  
29512 string that is associated with the process being started by one of the *exec* functions.

29513 The argument *envp* is an array of character pointers to null-terminated strings. These strings  
29514 shall constitute the environment for the new process image. The *envp* array is terminated by a  
29515 null pointer.

29516 For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*), the  
29517 environment for the new process image shall be taken from the external variable *environ* in the  
29518 calling process.

29519 The number of bytes available for the new process' combined argument and environment lists is  
29520 {ARG\_MAX}. It is implementation-defined whether null terminators, pointers, and/or any  
29521 alignment bytes are included in this total.

29522 File descriptors open in the calling process image shall remain open in the new process image,  
29523 except for those whose close-on-exec flag FD\_CLOEXEC is set. For those file descriptors that  
29524 remain open, all attributes of the open file description shall remain unchanged and the  
29525 FD\_CLOFORK file descriptor flag, if set, shall remain set. For any file descriptor that is closed  
29526 for this reason, process-owned file locks that the calling process owns on the file associated with  
29527 the file descriptor shall be unlocked as a result of the close, as described in *close()*. File locks that  
29528 are not unlocked by closing of file descriptors remain unchanged.

29529 If file descriptor 0, 1, or 2 would otherwise be closed after a successful call to one of the *exec*  
29530 family of functions, implementations may open an unspecified file for the file descriptor in the  
29531 new process image. If a standard utility or a conforming application is executed with file

- 29532 descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the  
29533 environment in which the utility or application is executed shall be deemed non-conforming,  
29534 and consequently the utility or application might not behave as described in this standard.
- 29535 Directory streams open in the calling process image shall be closed in the new process image.
- 29536 The state of the floating-point environment in the initial thread of the new process image shall  
29537 be set to the default.
- 29538 The state of conversion descriptors and message catalog descriptors in the new process image is  
29539 undefined.
- 29540 For the new process image, the equivalent of:
- 29541 `setlocale(LC_ALL, "C")`
- 29542 shall be executed at start-up.
- 29543 Signals set to the default action (SIG\_DFL) in the calling process image shall be set to the default  
29544 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG\_IGN) by  
29545 the calling process image shall be set to be ignored by the new process image. Signals set to be  
29546 caught by the calling process image shall be set to the default action in the new process image  
29547 (see <signal.h>).
- 29548 If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether  
29549 the SIGCHLD signal is set to be ignored or to the default action in the new process image.
- 29550 XSI After a successful call to any of the *exec* functions, alternate signal stacks are not preserved and  
29551 the SA\_ONSTACK flag shall be cleared for all signals.
- 29552 After a successful call to any of the *exec* functions, any functions previously registered by the  
29553 *atexit()*, *at\_quick\_exit()*, or *pthread\_atfork()* functions are no longer registered.
- 29554 XSI If the ST\_NOSUID bit is set for the file system containing the new process image file, then the  
29555 effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged  
29556 in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is  
29557 set, the effective user ID of the new process image shall be set to the user ID of the new process  
29558 image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the  
29559 effective group ID of the new process image shall be set to the group ID of the new process  
29560 image file. The real user ID, real group ID, and supplementary group IDs of the new process  
29561 image shall remain the same as those of the calling process image. The effective user ID and  
29562 effective group ID of the new process image shall be saved (as the saved set-user-ID and the  
29563 saved set-group-ID) for use by *setuid()*.
- 29564 XSI Any shared memory segments attached to the calling process image shall not be attached to the  
29565 new process image.
- 29566 Any named semaphores open in the calling process shall be closed as if by appropriate calls to  
29567 *sem\_close()*. Any unnamed semaphores open in the calling process shall be destroyed as if by  
29568 calls to *sem\_destroy()*.
- 29569 TYM Any blocks of typed memory that were mapped in the calling process are unmapped, as if  
29570 *munmap()* was implicitly called to unmap them.
- 29571 ML Memory locks established by the calling process via calls to *mlockall()* or *mlock()* shall be  
29572 removed. If locked pages in the address space of the calling process are also mapped into the  
29573 address spaces of other processes and are locked by those processes, the locks established by the  
29574 other processes shall be unaffected by the call by this process to the *exec* function. If the *exec*  
29575 function fails, the effect on memory locks is unspecified.

29576		Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image.
29577		
29578	SS	When the calling process image does not use the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC scheduling policies, the scheduling policy and parameters of the new process image and the initial thread in that new process image are implementation-defined.
29579		
29580		
29581	PS	When the calling process image uses the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC scheduling policies, the process policy and scheduling parameter settings shall not be changed by a call to an <i>exec</i> function. The initial thread in the new process image shall inherit the process scheduling policy and parameters. It shall have the default system contention scope, but shall inherit its allocation domain from the calling process image.
29582		
29583	TPS	
29584		
29585		
29586		Per-process timers created by the calling process shall be deleted before replacing the current process image with the new process image.
29587		
29588	MSG	All open message queue descriptors in the calling process shall be closed, as described in <i>mq_close()</i> .
29589		
29590		Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the <i>exec</i> function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the <i>exec</i> function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the <i>exec</i> function be affected by the presence of outstanding asynchronous I/O operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O may be canceled upon <i>exec</i> , is implementation-defined.
29591		
29592		
29593		
29594		
29595		
29596		
29597	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This inheritance means that the process CPU-time clock of the process being <i>exec</i> -ed shall not be reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the process executing the <i>exec</i> function itself.
29598		
29599		
29600		
29601	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set to zero.
29602		
29603		The thread ID of the initial thread in the new process image is unspecified.
29604		The size and location of the stack on which the initial thread in the new process image runs is unspecified.
29605		
29606		The initial thread in the new process image shall have its cancellation type set to PTHREAD_CANCEL_DEFERRED and its cancellation state set to PTHREAD_CANCEL_ENABLED.
29607		
29608		
29609		The initial thread in the new process image shall have all thread-specific data values set to NULL and all thread-specific data keys shall be removed by the call to <i>exec</i> without running destructors.
29610		
29611		
29612		The initial thread in the new process image shall be joinable, as if created with the <i>detachstate</i> attribute set to PTHREAD_CREATE_JOINABLE.
29613		
29614		The new process shall inherit at least the following attributes from the calling process image:
29615	XSI	• Nice value (see <i>nice()</i> )
29616	XSI	• <i>semadj</i> values (see <i>semop()</i> )
29617		• Process ID

- 29618 • Parent process ID
- 29619 • Process group ID
- 29620 • Session membership
- 29621 • Real user ID
- 29622 • Real group ID
- 29623 • Supplementary group IDs
- 29624 • Time left until an alarm clock signal (see *alarm()*)
- 29625 • Current working directory
- 29626 • Root directory
- 29627 • File mode creation mask (see *umask()*)
- 29628 • File size limit (see *getrlimit()* and *setrlimit()*)
- 29629 • Process signal mask (see *pthread\_sigmask()*)
- 29630 • Pending signal (see *sigpending()*)
- 29631 • *tms\_utime*, *tms\_stime*, *tms\_cutime*, and *tms\_cstime* (see *times()*)
- 29632 • Resource limits
- 29633 • Controlling terminal

29634 The initial thread of the new process shall inherit at least the following attributes from the  
 29635 calling thread:

- 29636 • Signal mask (see *sigprocmask()* and *pthread\_sigmask()*)
- 29637 • Pending signals (see *sigpending()*)

29638 All other process attributes defined in this volume of POSIX.1-2024 shall be inherited in the new  
 29639 process image from the old process image. All other thread attributes defined in this volume of  
 29640 POSIX.1-2024 shall be inherited in the initial thread in the new process image from the calling  
 29641 thread in the old process image. The inheritance of process or thread attributes not defined by  
 29642 this volume of POSIX.1-2024 is implementation-defined.

29643 A call to any *exec* function from a process with more than one thread shall result in all threads  
 29644 being terminated and the new executable image being loaded and executed. No destructor  
 29645 functions or cleanup handlers shall be called.

29646 Upon successful completion, the *exec* functions shall mark for update the last data access  
 29647 timestamp of the file. If an *exec* function failed but was able to locate the process image file,  
 29648 whether the last data access timestamp is marked for update is unspecified. Should the *exec*  
 29649 function succeed, the process image file shall be considered to have been opened with *open()*.  
 29650 The corresponding *close()* shall be considered to occur at a time after this open, but before  
 29651 process termination or successful completion of a subsequent call to one of the *exec* functions,  
 29652 *posix\_spawn()*, or *posix\_spawnnp()*. The *argv[]* and *envp[]* arrays of pointers and the strings to  
 29653 which those arrays point shall not be modified by a call to one of the *exec* functions, except as a  
 29654 consequence of replacing the process image.

29655 The saved resource limits in the new process image are set to be a copy of the process'  
 29656 corresponding hard and soft limits.

29657 **RETURN VALUE**

29658 If one of the *exec* functions returns to the calling process image, an error has occurred; the return  
 29659 value shall be  $-1$ , and *errno* shall be set to indicate the error.

29660 **ERRORS**

29661 The *exec* functions shall fail if:

29662 [E2BIG] The number of bytes used by the new process image's argument list and  
 29663 environment list is greater than the system-imposed limit of {ARG\_MAX}  
 29664 bytes.

29665 [EACCES] The new process image file is not a regular file and the implementation does  
 29666 not support execution of files of its type.

29667 [EINVAL] The new process image file has appropriate privileges and has a recognized  
 29668 executable binary format, but the system does not support execution of a file  
 29669 with this format.

29670 The *exec* functions, except for *fexecve()*, shall fail if:

29671 [EACCES] Search permission is denied for a directory listed in the new process image  
 29672 file's path prefix, or the new process image file denies execution permission.

29673 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file*  
 29674 argument.

29675 [ENAMETOOLONG]  
 29676 The length of a component of a pathname is longer than {NAME\_MAX}.

29677 [ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an  
 29678 empty string.

29679 [ENOTDIR] A component of the new process image file's path prefix names an existing file  
 29680 that is neither a directory nor a symbolic link to a directory, or the new process  
 29681 image file's pathname contains at least one non- $\langle$ slash $\rangle$  character and ends  
 29682 with one or more trailing  $\langle$ slash $\rangle$  characters and the last pathname  
 29683 component names an existing file that is neither a directory nor a symbolic  
 29684 link to a directory.

29685 The *exec* functions, except for *execlp()* and *execvp()*, shall fail if:

29686 [ENOEXEC] The new process image file has the appropriate access permission but has an  
 29687 unrecognized format.

29688 The *fexecve()* function shall fail if:

29689 [EBADF] The *fd* argument is not a valid file descriptor open for executing.

29690 The *exec* functions may fail if:

29691 [ENOMEM] The new process image requires more memory than is allowed by the  
 29692 hardware or system-imposed memory management constraints.

29693 The *exec* functions, except for *fexecve()*, may fail if:

29694 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 29695 resolution of the *path* or *file* argument.

29696 [ENAMETOOLONG]  
 29697 The length of the *path* argument or the length of the pathname constructed  
 29698 from the *file* argument exceeds {PATH\_MAX}, or pathname resolution of a

29699 symbolic link produced an intermediate result with a length that exceeds  
29700 {PATH\_MAX}.

29701 [ETXTBSY] The new process image file is a pure procedure (shared text) file that is  
29702 currently open for writing by some process.

## 29703 EXAMPLES

### 29704 Using `execl()`

29705 The following example executes the `ls` command, specifying the pathname of the executable  
29706 (`/bin/ls`) and using arguments supplied directly to the command to produce single-column  
29707 output.

```
29708 #include <unistd.h>
29709 int ret;
29710 ...
29711 ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

### 29712 Using `execle()`

29713 The following example is similar to [Using `execl\(\)`](#). In addition, it specifies the environment for  
29714 the new process image using the `env` argument.

```
29715 #include <unistd.h>
29716 int ret;
29717 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
29718 ...
29719 ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

### 29720 Using `execlp()`

29721 The following example searches for the location of the `ls` command among the directories  
29722 specified by the `PATH` environment variable.

```
29723 #include <unistd.h>
29724 int ret;
29725 ...
29726 ret = execlp ("ls", "ls", "-l", (char *)0);
```

### 29727 Using `execv()`

29728 The following example passes arguments to the `ls` command in the `cmd` array.

```
29729 #include <unistd.h>
29730 int ret;
29731 char *cmd[] = { "ls", "-l", (char *)0 };
29732 ...
29733 ret = execv ("/bin/ls", cmd);
```



29734 **Using `execve()`**

29735 The following example passes arguments to the `ls` command in the `cmd` array, and specifies the  
29736 environment for the new process image using the `env` argument.

```
29737 #include <unistd.h>
29738
29738 int ret;
29739 char *cmd[] = { "ls", "-l", (char *)0 };
29740 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
29741 ...
29742 ret = execve ("/bin/ls", cmd, env);
```

29743 **Using `execvp()`**

29744 The following example searches for the location of the `ls` command among the directories  
29745 specified by the `PATH` environment variable, and passes arguments to the `ls` command in the  
29746 `cmd` array.

```
29747 #include <unistd.h>
29748
29748 int ret;
29749 char *cmd[] = { "ls", "-l", (char *)0 };
29750 ...
29751 ret = execvp ("ls", cmd);
```

29752 **APPLICATION USAGE**

29753 As the state of conversion descriptors and message catalog descriptors in the new process image  
29754 is undefined, conforming applications should not rely on their use and should close them prior  
29755 to calling one of the `exec` functions.

29756 Applications that require other than the default POSIX locale as the global locale in the new  
29757 process image should call `setlocale()` with the appropriate parameters.

29758 When assigning a new value to the `environ` variable, applications should ensure that the  
29759 environment to which it will point contains at least the following:

- 29760 1. Any implementation-defined variables required by the implementation to provide a  
29761 conforming environment. See the `_CS_V8_ENV` entry in `<unistd.h>` and `confstr()` for  
29762 details.
- 29763 2. A value for `PATH` which finds conforming versions of all standard utilities before any  
29764 other versions.

29765 The same constraint applies to the `envp` array passed to `execle()` or `execve()`, in order to ensure  
29766 that the new process image is invoked in a conforming environment.

29767 Applications should not execute programs with file descriptor 0 not open for reading or with file  
29768 descriptor 1 or 2 not open for writing, as this might cause the executed program to misbehave.  
29769 In order not to pass on these file descriptors to an executed program, applications should not  
29770 just close them but should reopen them on, for example, `/dev/null`. Some implementations may  
29771 reopen them automatically, but applications should not rely on this being done.

29772 If an application wants to perform an integrity test of the file being executed before executing it,  
29773 the file will need to be opened with read permission to perform the integrity test.

29774 Since execute permission is checked by `fexecve()`, the file description `fd` need not have been  
29775 opened with the `O_EXEC` flag. However, if the file to be executed denies read and write  
29776 permission for the process preparing to do the `exec`, the only way to provide the `fd` to `fexecve()`

29777 will be to use the O\_EXEC flag when opening *fd*. In this case, the application will not be able to  
 29778 perform an integrity test since it will not be able to read the contents of the file.

29779 Note that when a file descriptor is opened with O\_RDONLY, O\_RDWR, or O\_WRONLY mode,  
 29780 the file descriptor can be used to read, read and write, or write the file, respectively, even if the  
 29781 mode of the file changes after the file was opened. Using the O\_EXEC open mode is different;  
 29782 *fexecve()* will ignore the mode that was used when the file descriptor was opened and the *exec*  
 29783 will fail if the mode of the file associated with *fd* does not grant execute permission to the calling  
 29784 process at the time *fexecve()* is called.

#### 29785 RATIONALE

29786 Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was  
 29787 driven by the same requirement in drafts of the ISO C standard. In fact, historical  
 29788 implementations have passed a value of zero when no arguments are supplied to the caller of  
 29789 the *exec* functions. This requirement was removed from the ISO C standard and subsequently  
 29790 removed from this volume of POSIX.1-2024 as well. The wording, in particular the use of the  
 29791 word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to  
 29792 the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an  
 29793 application. In fact, this is good practice, since many existing applications reference *argv[0]*  
 29794 without first checking the value of *argc*.

29795 The requirement on a Strictly Conforming POSIX Application also states that the value passed as  
 29796 the first argument be a filename string associated with the process being started. Although some  
 29797 existing applications pass a pathname rather than a filename string in some circumstances, a  
 29798 filename string is more generally useful, since the common usage of *argv[0]* is in printing  
 29799 diagnostics. In some cases the filename passed is not the actual filename of the file; for example,  
 29800 many implementations of the *login* utility use a convention of prefixing a <hyphen-minus>  
 29801 ('-') to the actual filename, which indicates to the command interpreter being invoked that it is  
 29802 a “login shell”.

29803 Also, note that the *test* and *[* utilities require specific strings for the *argv[0]* argument to have  
 29804 deterministic behavior across all implementations.

29805 Historically, there have been two ways that implementations can *exec* shell scripts.

29806 One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()*  
 29807 functions return an [ENOEXEC] error for any file not recognizable as executable, including a  
 29808 shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file  
 29809 to be a shell script and invoke a known command interpreter to interpret such files. This is now  
 29810 required by POSIX.1-2024. These implementations of *execvp()* and *execlp()* only give the  
 29811 [ENOEXEC] error in the rare case of a problem with the command interpreter’s executable file.  
 29812 Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp()* or  
 29813 *execvp()*, although implementations can still give it.

29814 Another way that some historical implementations handle shell scripts is by recognizing the first  
 29815 two bytes of the file as the character string “#!” and using the remainder of the first line of the  
 29816 file as the name of the command interpreter to execute.

29817 One potential source of confusion noted by the standard developers is over how the contents of  
 29818 a process image file affect the behavior of the *exec* family of functions. The following is a  
 29819 description of the actions taken:

- 29820 1. If the process image file is a valid executable (in a format that is executable and valid and  
 29821 having appropriate privileges) for this system, then the system executes the file.

- 29822 2. If the process image file has appropriate privileges and is in a format that is executable  
 29823 but not valid for this system (such as a recognized binary for another architecture), then  
 29824 this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
- 29825 3. If the process image file has appropriate privileges but is not otherwise recognized:
- 29826 a. If this is a call to *execlp()* or *execvp()*, then they invoke a command interpreter  
 29827 assuming that the process image file is a shell script.
- 29828 b. If this is not a call to *execlp()* or *execvp()*, then an error occurs and *errno* is set to  
 29829 [ENOEXEC].

29830 Applications that do not require to access their arguments may use the form:

29831 `main(void)`

29832 as specified in the ISO C standard. However, the implementation will always provide the two  
 29833 arguments *argc* and *argv*, even if they are not used.

29834 Some implementations provide a third argument to *main()* called *envp*. This is defined as a  
 29835 pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments,  
 29836 so implementations are required to support applications written this way. Since this volume of  
 29837 POSIX.1-2024 defines the global variable *environ*, which is also provided by historical  
 29838 implementations and can be used anywhere that *envp* could be used, there is no functional need  
 29839 for the *envp* argument. Applications should use the *getenv()* function rather than accessing the  
 29840 environment directly via either *envp* or *environ*. Implementations are required to support the  
 29841 two-argument calling sequence, but this does not prohibit an implementation from supporting  
 29842 *envp* as an optional third argument.

29843 This volume of POSIX.1-2024 specifies that signals set to SIG\_IGN remain set to SIG\_IGN, and  
 29844 that the new process image inherits the signal mask of the thread that called *exec* in the old  
 29845 process image. This is consistent with historical implementations, and it permits some useful  
 29846 functionality, such as the *nohup* command. However, it should be noted that many existing  
 29847 applications wrongly assume that they start with certain signals set to the default action and/or  
 29848 unblocked. In particular, applications written with a simpler signal model that does not include  
 29849 blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked  
 29850 with some signals blocked. Therefore, it is best not to block or ignore signals across *execs* without  
 29851 explicit reason to do so, and especially not to block signals across *execs* of arbitrary (not closely  
 29852 cooperating) programs.

29853 The *exec* functions always save the value of the effective user ID and effective group ID of the  
 29854 process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of  
 29855 the process image file is set.

29856 The statement about *argv[]* and *envp[]* being constants is included to make explicit to future  
 29857 writers of language bindings that these objects are completely constant. Due to a limitation of  
 29858 the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of  
 29859 *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the  
 29860 natural choice, given that these functions do not modify either the array of pointers or the  
 29861 characters to which the function points, but this would disallow existing correct code. Instead,  
 29862 only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src*  
 29863 derived from the ISO C standard summarizes the compatibility:

	<i>dst:</i>	<b>char *[]</b>	<b>const char *[]</b>	<b>char *const[]</b>	<b>const char *const[]</b>
29864	<i>src:</i>				
29865	<b>char *[]</b>	VALID	—	VALID	—
29866	<b>const char *[]</b>	—	VALID	—	VALID
29867	<b>char * const []</b>	—	—	VALID	—
29868	<b>const char *const[]</b>	—	—	—	VALID

29870 Since all existing code has a source type matching the first row, the column that gives the most  
 29871 valid combinations is the third column. The only other possibility is the fourth column, but  
 29872 using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth  
 29873 column cannot be used, because the declaration a non-expert would naturally use would be that  
 29874 in the second row.

29875 The ISO C standard and this volume of POSIX.1-2024 do not conflict on the use of *environ*, but  
 29876 some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in  
 29877 the same way as an entry point (for example, *fork()*), it conforms to both standards. A library can  
 29878 contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence and no  
 29879 problem ensues. The situation is similar for *environ*: the definition in this volume of  
 29880 POSIX.1-2024 is to be used if there is no user-provided *environ* to take precedence. At least three  
 29881 implementations are known to exist that solve this problem.

29882 [E2BIG] The limit {ARG\_MAX} applies not just to the size of the argument list, but to  
 29883 the sum of that and the size of the environment list.

29884 [EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the  
 29885 new process image file is corrupted. They are non-conforming.

29886 [EINVAL] This error condition was added to POSIX.1-2024 to allow an implementation  
 29887 to detect executable files generated for different architectures, and indicate this  
 29888 situation to the application. Historical implementations of shells, *execvp()*, and  
 29889 *execlp()* that encounter an [ENOEXEC] error will execute a shell on the  
 29890 assumption that the file is a shell script. This will not produce the desired  
 29891 effect when the file is a valid executable for a different architecture. An  
 29892 implementation may now choose to avoid this problem by returning  
 29893 [EINVAL] when a valid executable for a different architecture is encountered.  
 29894 Some historical implementations return [EINVAL] to indicate that the *path*  
 29895 argument contains a character with the high order bit set. The standard  
 29896 developers chose to deviate from historical practice for the following reasons:

- 29897 1. The new utilization of [EINVAL] will provide some measure of utility  
 29898 to the user community.
- 29899 2. Historical use of [EINVAL] is not acceptable in an internationalized  
 29900 operating environment.

29901 [ENAMETOOLONG] Since the file pathname may be constructed by taking elements in the *PATH*  
 29902 variable and putting them together with the filename, the  
 29903 [ENAMETOOLONG] error condition could also be reached this way.  
 29904

29905 [ETXTBSY] System V returns this error when the executable file is currently open for  
 29906 writing by some process. This volume of POSIX.1-2024 neither requires nor  
 29907 prohibits this behavior.

29908 Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this  
 29909 volume of POSIX.1-2024, but implementations may have a window between the call to *exec* and

29910 the time that a signal could cause one of the *exec* calls to return with [EINTR].

29911 An explicit statement regarding the floating-point environment (as defined in the `<fenv.h>`  
29912 header) was added to make it clear that the floating-point environment is set to its default when  
29913 a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the  
29914 default for other process and thread start-up functions is covered by more generic statements in  
29915 their descriptions and can be summarized as follows:

29916 *posix\_spawn()* Set to default.

29917 *fork()* Inherit.

29918 *pthread\_create()* Inherit.

29919 The purpose of the *fexecve()* function is to enable executing a file which has been verified to be  
29920 the intended file. It is possible to actively check the file by reading from the file descriptor and be  
29921 sure that the file is not exchanged for another between the reading and the execution.  
29922 Alternatively, a function like *openat()* can be used to open a file which has been found by  
29923 reading the content of a directory using *readdir()*.

29924 When *execlp()* or *execvp()* fall back to invoking *sh* because of an [ENOEXEC] condition, the  
29925 standard leaves the process name (what becomes *argv[0]* in the resulting *sh* process) unspecified.  
29926 Existing implementations vary on whether they pass a variation of "sh", or preserve the  
29927 original *arg0*. There are existing implementations of *sh* that behave differently depending on the  
29928 contents of *argv[0]*, such that blindly passing the original *arg0* on to the fallback execution can  
29929 fail to invoke a compliant shell environment. Because of the requirements on how *sh* handles its  
29930 command line arguments, the shell script will see \$0 containing the pathname of the script  
29931 being executed, regardless of the value of *argv[0]*.

#### 29932 FUTURE DIRECTIONS

29933 None.

#### 29934 SEE ALSO

29935 *alarm()*, *atexit()*, *chmod()*, *close()*, *confstr()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getrlimit()*,  
29936 *mknod()*, *mmap()*, *nice()*, *open()*, *posix\_spawn()*, *pthread\_atfork()*, *pthread\_sigmask()*, *putenv()*,  
29937 *readdir()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*, *sigpending()*, *system()*, *times()*,  
29938 *umask()*

29939 XBD Chapter 8 (on page 167), `<unistd.h>`

29940 XCU *test*

#### 29941 CHANGE HISTORY

29942 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 29943 Issue 5

29944 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
29945 Threads Extension.

29946 Large File Summit extensions are added.

29947 **Issue 6**

29948 The following new requirements on POSIX implementations derive from alignment with the  
29949 Single UNIX Specification:

- 29950 • In the DESCRIPTION, behavior is defined for when the process image file is not a valid  
29951 executable.
- 29952 • In this version, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective  
29953 group ID of the new process image shall be saved (as the saved set-user-ID and the saved  
29954 set-group-ID) for use by the `setuid()` function.
- 29955 • The [ELOOP] mandatory error condition is added.
- 29956 • A second [ENAMETOOLONG] is added as an optional error condition.
- 29957 • The [ETXTBSY] optional error condition is added.

29958 The following changes were made to align with the IEEE P1003.1a draft standard:

- 29959 • The [EINVAL] mandatory error condition is added.
- 29960 • The [ELOOP] optional error condition is added.

29961 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

29962 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics  
29963 for typed memory.

29964 The normative text is updated to avoid use of the term “must” for application requirements.

29965 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

29966 IEEE PASC Interpretation 1003.1 #132 is applied.

29967 The DESCRIPTION is updated to make it explicit that the floating-point environment in the new  
29968 process image is set to the default.

29969 The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents  
29970 of a process image file affect the behavior of the *exec* functions.

29971 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to  
29972 the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change  
29973 addresses a security concern, where implementations may want to reopen file descriptors 0, 1,  
29974 and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of  
29975 functions.

29976 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/24 is applied, applying changes to the  
29977 DESCRIPTION, addressing which attributes are inherited by threads, and behavioral  
29978 requirements for threads attributes.

29979 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/25 is applied, updating text in the  
29980 RATIONALE from “the process signal mask be unchanged across an *exec*” to “the new process  
29981 image inherits the signal mask of the thread that called *exec* in the old process image”.

29982 **Issue 7**

29983 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the description of `_CS_V7_ENV`  
29984 to the APPLICATION USAGE.

29985 Austin Group Interpretation 1003.1-2001 #143 is applied.

29986 The *fexecve()* function is added from The Open Group Technical Standard, 2006, Extended API  
29987 Set Part 2.

- 29988            Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads,  
29989            and Timers options is moved to the Base.
- 29990            Changes are made related to support for finegrained timestamps.
- 29991            POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0095 [386], XSH/TC1-2008/0096 [167],  
29992            XSH/TC1-2008/0097 [291], XSH/TC1-2008/0098 [173], XSH/TC1-2008/0099 [296],  
29993            XSH/TC1-2008/00100 [324], XSH/TC1-2008/00101 [296], XSH/TC1-2008/00102 [302],  
29994            XSH/TC1-2008/00103 [167], XSH/TC1-2008/00104 [173], and XSH/TC1-2008/00105 [291,429]  
29995            are applied.
- 29996            POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0097 [584], XSH/TC2-2008/0098 [898],  
29997            and XSH/TC2-2008/0099 [734] are applied.
- 29998    **Issue 8**
- 29999            Austin Group Defects 51 and 1669 are applied, moving the *getrlimit()* and *setrlimit()* functions  
30000            from the XSI option to the Base.
- 30001            Austin Group Defect 368 is applied, adding a requirement for unnamed semaphores to be  
30002            destroyed.
- 30003            Austin Group Defect 768 is applied, adding OFD-owned file locks.
- 30004            Austin Group Defect 922 is applied, adding the *secure\_getenv()* function.
- 30005            Austin Group Defect 1284 is applied, changing “checksum test” to “integrity test” in the  
30006            APPLICATION USAGE section.
- 30007            Austin Group Defect 1318 is applied, adding FD\_CLOFORK.
- 30008            Austin Group Defect 1330 is applied, removing obsolescent interfaces and changing “\_V7\_” to  
30009            “\_V8\_”.
- 30010            Austin Group Defect 1435 is applied, adding function lists to the descriptions of the *path* and *file*  
30011            arguments.
- 30012            Austin Group Defect 1645 is applied, making it unspecified what string is passed to the shell in  
30013            *argv[0]* when executed by *execlp()* or *execvp()*.
- 30014            Austin Group Defect 1646 is applied, adding *at\_quick\_exit()* to the list of registration functions  
30015            whose registrations are not inherited across *exec*.

30016 **NAME**

30017 exit — terminate a process

30018 **SYNOPSIS**

30019 #include &lt;stdlib.h&gt;

30020 void exit(int status);

30021 **DESCRIPTION**

30022 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30023 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30024 volume of POSIX.1-2024 defers to the ISO C standard.

30025 The *exit()* function shall cause normal process termination to occur. No functions registered by  
 30026 the *at\_quick\_exit()* function shall be called. If a process calls the *exit()* function more than once,  
 30027 or calls the *quick\_exit()* function in addition to the *exit()* function, the behavior is undefined.

30028 CX The value of *status* can be 0, EXIT\_SUCCESS, EXIT\_FAILURE, or any other value, though only  
 30029 the least significant 8 bits (that is, *status* & 0377) shall be available from *wait()* and *waitpid()*; the  
 30030 full value shall be available from *waitid()* and in the **siginfo\_t** passed to a signal handler for  
 30031 SIGCHLD.

30032 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their  
 30033 registration, except that a function is called after any previously registered functions that had  
 30034 already been called at the time it was registered. Each function is called as many times as it was  
 30035 registered. If, during the call to any such function, a call to the *longjmp()* function is made that  
 30036 would terminate the call to the registered function, the behavior is undefined.

30037 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall  
 30038 not be called and the rest of the *exit()* processing shall not be completed.

30039 CX The *exit()* function shall then flush all open streams with unwritten buffered data. For each  
 30040 stream which is the active handle to its underlying file descriptor, and for which the file is not  
 30041 already at EOF and is capable of seeking, the file offset of the underlying open file description  
 30042 shall be set to the file position of the stream. For each open stream, the *exit()* function shall  
 30043 perform the equivalent of a *close()* on the file descriptor that is associated with the stream.

30044 CX Finally, the process shall be terminated with the same consequences as described in  
 30045 [Consequences of Process Termination](#) (on page 568).

30046 **RETURN VALUE**30047 The *exit()* function does not return.30048 **ERRORS**

30049 No errors are defined.

30050 **EXAMPLES**

30051 See APPLICATION USAGE.

30052 **APPLICATION USAGE**

30053 When a stream that has unwritten buffered data is flushed by *exit()* there is no way for the  
 30054 calling process to discover whether or not *exit()* successfully wrote the data to the underlying  
 30055 file descriptor. Therefore, it is strongly recommended that applications always ensure there is no  
 30056 unwritten buffered data in any stream when calling *exit()*, or returning from the initial call to  
 30057 *main()*, with a *status* value that indicates no errors occurred.

30058 For example, the following code demonstrates one way to ensure that *stdout* has already been  
 30059 successfully flushed before calling *exit()* with status 0. If the flush fails, the file descriptor  
 30060 underlying *stdout* is closed so that *exit()* will not try to repeat the failed write operation. If the  
 30061 flush succeeds, a final check with *ferror()* is performed to ensure that there were no write errors



```

30062         during earlier flush operations (that were not handled at the time).
30063         int status = 0;
30064         if (fflush(stdout) != 0) {
30065             perror("appname: standard output");
30066             close(fileno(stdout));
30067             status = 1;
30068         }
30069         else if (ferror(stdout)) {
30070             fputs("appname: write error on standard output\n", stderr);
30071             status = 1;
30072         }
30073         exit(status);
30074         See also _Exit().

```

**30075 RATIONALE**

30076 See *\_Exit()*.

**30077 FUTURE DIRECTIONS**

30078 None.

**30079 SEE ALSO**

30080 *\_Exit()*, *at\_quick\_exit()*, *atexit()*, *exec*, *fflush()*, *longjmp()*, *quick\_exit()*, *tmpfile()*, *wait()*, *waitid()*

30081 XBD <stdlib.h>

**30082 CHANGE HISTORY****30083 Issue 7**

30084 Austin Group Interpretation 1003.1-2001 #031 is applied, separating the *\_Exit()* and *\_exit()*  
30085 functions from the *exit()* function.

30086 Austin Group Interpretation 1003.1-2001 #085 is applied.

30087 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0100 [594] is applied.

**30088 Issue 8**

30089 Austin Group Defect 610 is applied, clarifying the effects of *exit()* on open streams.

30090 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
30091 standard.

30092 Austin Group Defect 1490 is applied, changing the EXAMPLES and APPLICATION USAGE  
30093 sections.

30094 Austin Group Defect 1629 is applied, changing the APPLICATION USAGE section.

30095 **NAME**30096 `exp`, `expf`, `expl` — exponential function30097 **SYNOPSIS**

```
30098 #include <math.h>
30099 double exp(double x);
30100 float expf(float x);
30101 long double expl(long double x);
```

30102 **DESCRIPTION**

30103 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30104 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30105 volume of POSIX.1-2024 defers to the ISO C standard.

30106 These functions shall compute the base-*e* exponential of *x*.

30107 An application wishing to check for error situations should set *errno* to zero and call  
 30108 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 30109 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 30110 zero, an error has occurred.

30111 **RETURN VALUE**

30112 Upon successful completion, these functions shall return the exponential value of *x*.

30113 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*  
 30114 shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.

30115 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
 30116 MXX and *exp()*, *expf()*, and *expl()* shall return 0.0, or (if the IEC 60559 Floating-Point option is not  
 30117 supported) an implementation-defined value no greater in magnitude than DBL\_MIN,  
 30118 FLT\_MIN, and LDBL\_MIN, respectively.

30119 MX If *x* is NaN, a NaN shall be returned.

30120 If *x* is  $\pm 0$ , 1 shall be returned.

30121 If *x* is  $-\text{Inf}$ , +0 shall be returned.

30122 If *x* is  $+\text{Inf}$ , *x* shall be returned.

30123 MXX If the correct value would cause underflow, and is representable, a range error may occur and  
 30124 the correct value shall be returned.

30125 **ERRORS**

30126 These functions shall fail if:

30127 Range Error The result overflows.

30128 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 30129 then *errno* shall be set to [ERANGE]. If the integer expression  
 30130 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 30131 floating-point exception shall be raised.

30132 These functions may fail if:

30133 Range Error The result underflows.

30134 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 30135 then *errno* shall be set to [ERANGE]. If the integer expression  
 30136 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 30137 floating-point exception shall be raised.

30138 **EXAMPLES**30139 **Computing the Density of the Standard Normal Distribution**

30140 This function shows an implementation for the density of the standard normal distribution  
 30141 using `exp()`. This example uses the constant `M_PI` which is part of the XSI option.

```
30142 #include <math.h>
30143
30144 double
30145 normal_density (double x)
30146 {
30147     return exp(-x*x/2) / sqrt (2*M_PI);
30148 }
```

30148 **APPLICATION USAGE**

30149 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`  
 30150 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

30151 **RATIONALE**

30152 None.

30153 **FUTURE DIRECTIONS**

30154 None.

30155 **SEE ALSO**

30156 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log\(\)](#)

30157 XBD [Section 4.23](#) (on page 109), [<math.h>](#)

30158 **CHANGE HISTORY**

30159 First released in Issue 1. Derived from Issue 1 of the SVID.

30160 **Issue 5**

30161 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 30162 text was previously published in the APPLICATION USAGE section.

30163 **Issue 6**

30164 The `expf()` and `expl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

30165 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 30166 revised to align with the ISO/IEC 9899:1999 standard.

30167 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 30168 marked.

30169 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/26 is applied, adding the example to the  
 30170 EXAMPLES section.

30171 **Issue 7**

30172 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0106 [68] and XSH/TC1-2008/0107  
 30173 [68] are applied.

30174 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0101 [630] is applied.

30175 **NAME**

30176 exp2, exp2f, exp2l — exponential base 2 functions

30177 **SYNOPSIS**

```
30178 #include <math.h>
30179 double exp2(double x);
30180 float exp2f(float x);
30181 long double exp2l(long double x);
```

30182 **DESCRIPTION**

30183 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30184 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30185 volume of POSIX.1-2024 defers to the ISO C standard.

30186 These functions shall compute the base-2 exponential of  $x$ .

30187 An application wishing to check for error situations should set *errno* to zero and call  
 30188 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 30189 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 30190 zero, an error has occurred.

30191 **RETURN VALUE**

30192 Upon successful completion, these functions shall return  $2^x$ .

30193 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and  
 30194 *exp2l()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 30195 respectively.

30196 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
 30197 MXX and *exp2()*, *exp2f()*, and *exp2l()* shall return 0.0, or (if the IEC 60559 Floating-Point option is not  
 30198 supported) an implementation-defined value no greater in magnitude than DBL\_MIN,  
 30199 FLT\_MIN, and LDBL\_MIN, respectively.

30200 MX If  $x$  is NaN, a NaN shall be returned.

30201 If  $x$  is  $\pm 0$ , 1 shall be returned.

30202 If  $x$  is  $-\text{Inf}$ , +0 shall be returned.

30203 If  $x$  is  $+\text{Inf}$ ,  $x$  shall be returned.

30204 MXX If the correct value would cause underflow, and is representable, a range error may occur and  
 30205 the correct value shall be returned.

30206 **ERRORS**

30207 These functions shall fail if:

30208 Range Error The result overflows.

30209 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 30210 then *errno* shall be set to [ERANGE]. If the integer expression  
 30211 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 30212 floating-point exception shall be raised.

30213 These functions may fail if:

30214 Range Error The result underflows.

30215 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 30216 then *errno* shall be set to [ERANGE]. If the integer expression  
 30217 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow

30218 floating-point exception shall be raised.

30219 **EXAMPLES**

30220 None.

30221 **APPLICATION USAGE**

30222 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
30223 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

30224 **RATIONALE**

30225 None.

30226 **FUTURE DIRECTIONS**

30227 None.

30228 **SEE ALSO**

30229 *exp()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log()*

30230 XBD Section 4.23 (on page 109), <math.h>

30231 **CHANGE HISTORY**

30232 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

30233 **Issue 7**

30234 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0108 [68] and XSH/TC1-2008/0109  
30235 [68] are applied.

30236 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0102 [630] is applied.

30237 **NAME**

30238 expm1, expm1f, expm1l — compute exponential functions

30239 **SYNOPSIS**

```
30240 #include <math.h>
30241 double expm1(double x);
30242 float expm1f(float x);
30243 long double expm1l(long double x);
```

30244 **DESCRIPTION**

30245 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30246 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30247 volume of POSIX.1-2024 defers to the ISO C standard.

30248 These functions shall compute  $e^x - 1.0$ .

30249 An application wishing to check for error situations should set *errno* to zero and call  
 30250 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 30251 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 30252 zero, an error has occurred.

30253 **RETURN VALUE**30254 Upon successful completion, these functions return  $e^x - 1.0$ .

30255 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and  
 30256 *expm1l()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 30257 respectively.

30258 MX If  $x$  is NaN, a NaN shall be returned.30259 If  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.30260 If  $x$  is  $-\text{Inf}$ ,  $-1$  shall be returned.30261 If  $x$  is  $+\text{Inf}$ ,  $x$  shall be returned.30262 MXX If  $x$  is subnormal,  $x$  should be returned.

30263 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *expm1()*, *expm1f()*, and  
 30264 *expm1l()* shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
 30265 FLT\_MIN, and LDBL\_MIN, respectively.

30266 **ERRORS**

30267 These functions shall fail if:

30268 Range Error The result overflows.

30269 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 30270 then *errno* shall be set to [ERANGE]. If the integer expression  
 30271 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 30272 floating-point exception shall be raised.

30273 These functions may fail if:

30274 MX Range Error The value of  $x$  is subnormal.

30275 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 30276 then *errno* shall be set to [ERANGE]. If the integer expression  
 30277 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 30278 floating-point exception shall be raised.

30279 **EXAMPLES**

30280 None.

30281 **APPLICATION USAGE**30282 The value of  $\text{expm1}(x)$  may be more accurate than  $\text{exp}(x)-1.0$  for small values of  $x$ .30283 The  $\text{expm1}()$  and  $\text{log1p}()$  functions are useful for financial calculations of  $((1+x)^n-1)/x$ , namely:30284  $\text{expm1}(n * \text{log1p}(x)) / x$ 30285 when  $x$  is very small (for example, when calculating small daily interest rates). These functions  
30286 also simplify writing accurate inverse hyperbolic functions.30287 On error, the expressions ( $\text{math_errhandling}$  & MATH\_ERRNO) and ( $\text{math_errhandling}$  &  
30288 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.30289 **RATIONALE**

30290 None.

30291 **FUTURE DIRECTIONS**

30292 None.

30293 **SEE ALSO**30294 [exp\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [ilogb\(\)](#), [log1p\(\)](#)30295 XBD Section 4.23 (on page 109), [<math.h>](#)30296 **CHANGE HISTORY**

30297 First released in Issue 4, Version 2.

30298 **Issue 5**

30299 Moved from X/OPEN UNIX extension to BASE.

30300 **Issue 6**30301 The  $\text{expm1f}()$  and  $\text{expm1l}()$  functions are added for alignment with the ISO/IEC 9899:1999  
30302 standard.30303 The  $\text{expm1}()$  function is no longer marked as an extension.30304 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
30305 revised to align with the ISO/IEC 9899:1999 standard.30306 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
30307 marked.30308 **Issue 7**

30309 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0110 [68] is applied.

30310 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0103 [630] is applied.

30311 **Issue 8**30312 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when  $x$  is  
30313 subnormal to avoid the need for two shading changes.

30314 **NAME**

30315           fabs, fabsf, fabsl — absolute value function

30316 **SYNOPSIS**

```
30317       #include <math.h>
30318       double fabs(double x);
30319       float fabsf(float x);
30320       long double fabsl(long double x);
```

30321 **DESCRIPTION**

30322 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 30323 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30324 volume of POSIX.1-2024 defers to the ISO C standard.

30325       These functions shall compute the absolute value of their argument  $x$ ,  $|x|$ .30326 **RETURN VALUE**30327       Upon successful completion, these functions shall return the absolute value of  $x$ .

30328 MX       The returned value shall be exact and shall be independent of the current rounding direction  
 30329 mode.

30330       If  $x$  is NaN, a NaN shall be returned.30331       If  $x$  is  $\pm 0$ ,  $+0$  shall be returned.30332       If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.30333 **ERRORS**

30334       No errors are defined.

30335 **EXAMPLES**30336       **Computing the 1-Norm of a Floating-Point Vector**30337       This example shows the use of *fabs()* to compute the 1-norm of a vector defined as follows:30338        $\text{norm1}(v) = |v[0]| + |v[1]| + \dots + |v[n-1]|$ 

30339       where  $|x|$  denotes the absolute value of  $x$ ,  $n$  denotes the vector's dimension  $v[i]$  denotes the  $i$ -th  
 30340 component of  $v$  ( $0 \leq i < n$ ).

```
30341       #include <math.h>
30342       double
30343       norm1(const double v[], const int n)
30344       {
30345           int        i;
30346           double    n1_v; /* 1-norm of v */
30347           n1_v = 0;
30348           for (i=0; i<n; i++) {
30349               n1_v += fabs (v[i]);
30350           }
30351           return n1_v;
30352       }
```



30353 **APPLICATION USAGE**

30354 None.

30355 **RATIONALE**

30356 None.

30357 **FUTURE DIRECTIONS**

30358 None.

30359 **SEE ALSO**30360 *isnan()*

30361 XBD &lt;math.h&gt;

30362 **CHANGE HISTORY**

30363 First released in Issue 1. Derived from Issue 1 of the SVID.

30364 **Issue 5**30365 The DESCRIPTION is updated to indicate how an application should check for an error. This  
30366 text was previously published in the APPLICATION USAGE section.30367 **Issue 6**30368 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.30369 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
30370 revised to align with the ISO/IEC 9899:1999 standard.30371 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
30372 marked.30373 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/27 is applied, adding the example to the  
30374 EXAMPLES section.30375 **Issue 8**30376 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
30377 standard.

30378 **NAME**30379        `faccessat` — determine accessibility of a file relative to directory file descriptor30380 **SYNOPSIS**30381        `#include <unistd.h>`30382        `int faccessat(int fd, const char *path, int amode, int flag);`30383 **DESCRIPTION**30384        Refer to [\*access\(\)\*](#).

30385 **NAME**

30386 fchdir — change working directory

30387 **SYNOPSIS**

30388 #include &lt;unistd.h&gt;

30389 int fchdir(int *fildev*);30390 **DESCRIPTION**30391 The *fchdir()* function shall be equivalent to *chdir()* except that the directory that is to be the new  
30392 current working directory is specified by the file descriptor *fildev*.30393 A conforming application can obtain a file descriptor for a file of type directory using *open()*,  
30394 provided that the file status flags and access modes do not contain *O\_WRONLY* or *O\_RDWR*.30395 **RETURN VALUE**30396 Upon successful completion, *fchdir()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
30397 indicate the error. On failure the current working directory shall remain unchanged.30398 **ERRORS**30399 The *fchdir()* function shall fail if:30400 [EACCES] Search permission is denied for the directory referenced by *fildev*.30401 [EBADF] The *fildev* argument is not an open file descriptor.30402 [ENOTDIR] The open file descriptor *fildev* does not refer to a directory.30403 The *fchdir()* may fail if:30404 [EINTR] A signal was caught during the execution of *fchdir()*.

30405 [EIO] An I/O error occurred while reading from or writing to the file system.

30406 **EXAMPLES**

30407 None.

30408 **APPLICATION USAGE**

30409 None.

30410 **RATIONALE**

30411 None.

30412 **FUTURE DIRECTIONS**

30413 None.

30414 **SEE ALSO**30415 [chdir\(\)](#), [dirfd\(\)](#)30416 XBD [<unistd.h>](#)30417 **CHANGE HISTORY**

30418 First released in Issue 4, Version 2.

30419 **Issue 5**

30420 Moved from X/OPEN UNIX extension to BASE.

30421 **Issue 7**30422 The *fchdir()* function is moved from the XSI option to the Base.

30423 **NAME**

30424 fchmod — change mode of a file

30425 **SYNOPSIS**

```
30426 #include <sys/stat.h>
30427 int fchmod(int fildev, mode_t mode);
```

30428 **DESCRIPTION**

30429 The *fchmod()* function shall be equivalent to *chmod()* except that the file whose permissions are  
 30430 changed is specified by the file descriptor *fildev*.

30431 SHM If *fildev* references a shared memory object, the *fchmod()* function need only affect the S\_IRUSR,  
 30432 S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits.

30433 TYM If *fildev* references a typed memory object, the behavior of *fchmod()* is unspecified.

30434 If *fildev* refers to a socket, the behavior of *fchmod()* is unspecified.

30435 **RETURN VALUE**

30436 Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return  $-1$  and set *errno* to  
 30437 indicate the error.

30438 **ERRORS**

30439 The *fchmod()* function shall fail if:

30440 [EBADF] The *fildev* argument is not an open file descriptor.

30441 [EPERM] The effective user ID does not match the owner of the file and the process does  
 30442 not have appropriate privileges.

30443 [EROFS] The file referred to by *fildev* resides on a read-only file system.

30444 The *fchmod()* function may fail if:

30445 XSI [EINTR] The *fchmod()* function was interrupted by a signal.

30446 XSI [EINVAL] The value of the *mode* argument is invalid.

30447 [EINVAL] The *fildev* argument refers to a pipe and the implementation disallows  
 30448 execution of *fchmod()* on a pipe.

30449 **EXAMPLES**30450 **Changing the Current Permissions for a File**

30451 The following example shows how to change the permissions for a file named `/home/cnd/mod1`  
 30452 so that the owner and group have read/write/execute permissions, but the world only has  
 30453 read/write permissions.

```
30454 #include <sys/stat.h>
30455 #include <fcntl.h>
30456 mode_t mode;
30457 int fildev;
30458 ...
30459 fildev = open("/home/cnd/mod1", O_RDWR);
30460 fchmod(fildev, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);
```

**30461 APPLICATION USAGE**

30462 None.

**30463 RATIONALE**

30464 None.

**30465 FUTURE DIRECTIONS**

30466 None.

**30467 SEE ALSO**

30468 *chmod()*, *chown()*, *creat()*, *fcntl()*, *fstatat()*, *fstatofs()*, *mknod()*, *open()*, *read()*, *write()*

30469 XBD <sys/stat.h>

**30470 CHANGE HISTORY**

30471 First released in Issue 4, Version 2.

**30472 Issue 5**

30473 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX  
30474 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a  
30475 second instance of [EINVAL] is defined in the list of optional errors.

**30476 Issue 6**

30477 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*  
30478 behavior is unspecified for typed memory objects.

**30479 Issue 8**

30480 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

30481 **NAME**

30482 fchmodat — change mode of a file relative to directory file descriptor

30483 **SYNOPSIS**

30484 #include <sys/stat.h>

30485 int fchmodat(int *fd*, const char \**path*, mode\_t *mode*, int *flag*);

30486 **DESCRIPTION**

30487 Refer to *chmod()*.

30488 **NAME**

30489 fchown — change owner and group of a file

30490 **SYNOPSIS**

30491 #include &lt;unistd.h&gt;

30492 int fchown(int *fildev*, uid\_t *owner*, gid\_t *group*);30493 **DESCRIPTION**30494 The *fchown()* function shall be equivalent to *chown()* except that the file whose owner and group  
30495 are changed is specified by the file descriptor *fildev*.30496 **RETURN VALUE**30497 Upon successful completion, *fchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
30498 indicate the error.30499 **ERRORS**30500 The *fchown()* function shall fail if:30501 [EBADF] The *fildev* argument is not an open file descriptor.30502 [EPERM] The effective user ID does not match the owner of the file or the process does  
30503 not have appropriate privileges and `_POSIX_CHOWN_RESTRICTED`  
30504 indicates that such privilege is required.30505 [EROFS] The file referred to by *fildev* resides on a read-only file system.30506 The *fchown()* function may fail if:30507 [EINVAL] The owner or group ID is not a value supported by the implementation. The  
30508 *fildev* argument refers to a pipe or socket and the implementation disallows  
30509 execution of *fchown()* on a pipe.

30510 [EIO] A physical I/O error has occurred.

30511 [EINTR] The *fchown()* function was interrupted by a signal which was caught.30512 **EXAMPLES**30513 **Changing the Current Owner of a File**30514 The following example shows how to change the owner of a file named `/home/cnd/mod1` to  
30515 `“jones”` and the group to `“cnd”`.30516 The numeric value for the user ID is obtained by extracting the user ID from the user database  
30517 entry associated with `“jones”`. Similarly, the numeric value for the group ID is obtained by  
30518 extracting the group ID from the group database entry associated with `“cnd”`. This example  
30519 assumes the calling program has appropriate privileges.

```

30520 #include <sys/types.h>
30521 #include <unistd.h>
30522 #include <fcntl.h>
30523 #include <pwd.h>
30524 #include <grp.h>
30525
30526 struct passwd *pwd;
30527 struct group *grp;
30528 int
30529     fildev;
30530
30531 ...
30532 fildev = open("/home/cnd/mod1", O_RDWR);
30533 pwd = getpwnam("jones");

```

```
30531     grp = getgrnam("cnd");
30532     fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

## 30533 APPLICATION USAGE

30534 None.

## 30535 RATIONALE

30536 None.

## 30537 FUTURE DIRECTIONS

30538 None.

## 30539 SEE ALSO

30540 [chown\(\)](#)

30541 XBD <[unistd.h](#)>

## 30542 CHANGE HISTORY

30543 First released in Issue 4, Version 2.

### 30544 Issue 5

30545 Moved from X/OPEN UNIX extension to BASE.

### 30546 Issue 6

30547 The following changes were made to align with the IEEE P1003.1a draft standard:

- 30548 • Clarification is added that a call to *fchown()* may not be allowed on a pipe.

30549 The *fchown()* function is defined as mandatory.

### 30550 Issue 7

30551 Functionality relating to XSI STREAMS is marked obsolescent.

### 30552 Issue 8

30553 Austin Group Defect 1330 is applied, removing obsolescent interfaces.



30554 **NAME**

30555 fchownat — change owner and group of a file relative to directory file descriptor

30556 **SYNOPSIS**

30557 #include &lt;unistd.h&gt;

30558 int fchownat(int *fd*, const char \**path*, uid\_t *owner*, gid\_t *group*,  
30559 int *flag*);30560 **DESCRIPTION**30561 Refer to *chown()*.

30562 **NAME**

30563           fclose — close a stream

30564 **SYNOPSIS**

30565           #include <stdio.h>

30566           int fclose(FILE \*stream);

30567 **DESCRIPTION**

30568 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
30569 conflict between the requirements described here and the ISO C standard is unintentional. This  
30570 volume of POSIX.1-2024 defers to the ISO C standard.

30571       The *fclose()* function shall cause the stream pointed to by *stream* to be flushed and the associated  
30572 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any  
30573 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be  
30574 disassociated from the file and any buffer set by the *setbuf()* or *setvbuf()* function shall be  
30575 disassociated from the stream. If the associated buffer was automatically allocated, it shall be  
30576 deallocated.

30577 CX       If the file is not already at EOF, and the file is one capable of seeking, the file offset of the  
30578 underlying open file description shall be set to the file position of the stream if the stream is the  
30579 active handle to the underlying file description.

30580       The *fclose()* function shall mark for update the last data modification and last file status change  
30581 timestamps of the underlying file, if the stream was writable, and if buffered data remains that  
30582 has not yet been written to the file. The *fclose()* function shall perform the equivalent of a *close()*  
30583 on the file descriptor that is associated with the stream pointed to by *stream*.

30584       After the call to *fclose()*, any use of *stream* results in undefined behavior.

30585 **RETURN VALUE**

30586 CX       Upon successful completion, *fclose()* shall return 0; otherwise, it shall return EOF and set *errno*  
30587 to indicate the error.

30588 **ERRORS**

30589       The *fclose()* function shall fail if:

30590 CX       [EAGAIN]       The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and  
30591                           the thread would be delayed in the write operation.

30592 CX       [EBADF]       The file descriptor underlying stream is not valid.

30593 CX       [EFBIG]       An attempt was made to write a file that exceeds the maximum file size.

30594 CX       [EFBIG]       An attempt was made to write a file that exceeds the file size limit of the  
30595 process.

30596 XSI       A SIGXFSZ signal shall also be generated for the thread.

30597 CX       [EFBIG]       The file is a regular file and an attempt was made to write at or beyond the  
30598 offset maximum associated with the corresponding stream.

30599 CX       [EINTR]       The *fclose()* function was interrupted by a signal.

30600 CX       [EIO]         The process is a member of a background process group attempting to write to  
30601 its controlling terminal, TOSTOP is set, the calling thread is not blocking  
30602 SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the  
30603 process is orphaned. This error may also be returned under implementation-  
30604 defined conditions.

30605 CX [ENOMEM] The underlying stream was created by *open\_memstream()* or  
 30606 *open\_wmemstream()* and insufficient memory is available.

30607 CX [ENOSPC] There was no free space remaining on the device containing the file or in the  
 30608 buffer used by the *fmemopen()* function.

30609 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 30610 any process. A SIGPIPE signal shall also be sent to the thread.

30611 The *fclose()* function may fail if:

30612 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
 30613 capabilities of the device.

#### 30614 EXAMPLES

30615 None.

#### 30616 APPLICATION USAGE

30617 Since after the call to *fclose()* any use of *stream* results in undefined behavior, *fclose()* should not  
 30618 be used on *stdin*, *stdout*, or *stderr* except immediately before process termination (see XBD  
 30619 Section 3.287, on page 73), so as to avoid triggering undefined behavior in other standard  
 30620 interfaces that rely on these streams. If there are any *atexit()* handlers registered by the  
 30621 application, such a call to *fclose()* should not occur until the last handler is finishing. Once  
 30622 *fclose()* has been used to close *stdin*, *stdout*, or *stderr*, there is no standard way to reopen any of  
 30623 these streams.

30624 Use of *freopen()* to change *stdin*, *stdout*, or *stderr* instead of closing them avoids the danger of a  
 30625 file unexpectedly being opened as one of the special file descriptors `STDIN_FILENO`,  
 30626 `STDOUT_FILENO`, or `STDERR_FILENO` at a later time in the application.

#### 30627 RATIONALE

30628 None.

#### 30629 FUTURE DIRECTIONS

30630 None.

#### 30631 SEE ALSO

30632 Section 2.5 (on page 521), *atexit()*, *close()*, *fmemopen()*, *fopen()*, *freopen()*, *getrlimit()*,  
 30633 *open\_memstream()*

30634 XBD <stdio.h>

#### 30635 CHANGE HISTORY

30636 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 30637 Issue 5

30638 Large File Summit extensions are added.

#### 30639 Issue 6

30640 Extensions beyond the ISO C standard are marked.

30641 The following new requirements on POSIX implementations derive from alignment with the  
 30642 Single UNIX Specification:

- 30643 • The [EFBIG] error is added as part of the large file support extensions.
- 30644 • The [ENXIO] optional error condition is added.

30645 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether  
 30646 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

30647 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/28 is applied, updating the [EAGAIN]

30648 error in the ERRORS section from “the process would be delayed” to “the thread would be  
30649 delayed”.

30650 **Issue 7**

30651 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file  
30652 descriptors and streams.

30653 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open  
30654 Group Technical Standard, 2006, Extended API Set Part 1.

30655 Changes are made related to support for finegrained timestamps.

30656 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0113 [87], XSH/TC1-2008/0114 [79],  
30657 and XSH/TC1-2008/0115 [14] are applied.

30658 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0104 [555] is applied.

30659 **Issue 8**

30660 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

30661 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
30662 relating to the file size limit for the process.

30663 **NAME**

30664           fcntl — file control

30665 **SYNOPSIS**

30666           #include &lt;fcntl.h&gt;

30667           int fcntl(int *fildev*, int *cmd*, ...);30668 **DESCRIPTION**30669           The *fcntl()* function shall perform the operations described below on open files. The *fildev*  
30670           argument is a file descriptor.30671           The available values for *cmd* are defined in <fcntl.h> and are as follows:

30672           **F\_DUPFD**           Return a new file descriptor which shall be allocated as described in  
30673                                Section 2.6 (on page 525), except that it shall be the lowest numbered  
30674                                available file descriptor greater than or equal to the third argument, *arg*,  
30675                                taken as an integer of type **int**. The new file descriptor shall refer to the  
30676                                same open file description as the original file descriptor, and shall share  
30677                                any locks. The FD\_CLOEXEC and FD\_CLOFORK flags associated with  
30678                                the new file descriptor shall be cleared.

30679           **F\_DUPFD\_CLOEXEC**30680                                Like F\_DUPFD, but the FD\_CLOEXEC flag associated with the new file  
30681                                descriptor shall be set.30682           **F\_DUPFD\_CLOFORK**30683                                Like F\_DUPFD, but the FD\_CLOFORK flag associated with the new file  
30684                                descriptor shall be set.30685           **F\_GETFD**30686                                Get the file descriptor flags defined in <fcntl.h> that are associated with  
30687                                the file descriptor *fildev*. File descriptor flags are associated with a single  
30688                                file descriptor and do not affect other file descriptors that refer to the  
30689                                same file.30689           **F\_SETFD**30690                                Set the file descriptor flags defined in <fcntl.h>, that are associated with  
30691                                *fildev*, to the third argument, *arg*, taken as type **int**. If the FD\_CLOEXEC  
30692                                flag in the third argument is set, the file descriptor shall be closed upon  
30693                                successful execution of an *exec* family function and in the new process  
30694                                image created by *posix\_spawn()* or *posix\_spawnnp()*; otherwise, the file  
30695                                descriptor shall remain open. If the FD\_CLOFORK flag in the third  
30696                                argument is set, the file descriptor shall not be inherited by any child  
30697                                process created from a process that has the file descriptor open;  
30698                                otherwise, the file descriptor shall be inherited.30698           **F\_GETFL**30699                                Get the file status flags and file access modes, defined in <fcntl.h>, for the  
30700                                file description associated with *fildev*. The file access modes can be  
30701                                extracted from the return value using the mask O\_ACCMODE, which is  
30702                                defined in <fcntl.h>. File status flags and file access modes are associated  
30703                                with the file description and do not affect other file descriptors that refer  
30704                                to the same file with different open file descriptions. The flags returned  
30705                                may include non-standard file status flags which the application did not  
30706                                set, provided that these additional flags do not alter the behavior of a  
30707                                conforming application.30707           **F\_SETFL**30708                                Set the file status flags, defined in <fcntl.h>, for the file description  
30709                                associated with *fildev* from the corresponding bits in the third argument,  
30709                                *arg*, taken as type **int**. Bits corresponding to the file access mode and the

30710		file creation flags, as defined in <code>&lt;fcntl.h&gt;</code> , that are set in <i>arg</i> shall be
30711		ignored. If any bits in <i>arg</i> other than those mentioned here are changed by
30712		the application, the result is unspecified. If <i>fildev</i> does not support non-
30713		blocking operations, it is unspecified whether the <code>O_NONBLOCK</code> flag
30714		will be ignored.
30715	F_GETOWN	If <i>fildev</i> refers to a socket, get the process ID or process group ID specified
30716		to receive SIGURG signals when out-of-band data is available. Positive
30717		values shall indicate a process ID; negative values, other than <code>-1</code> , shall
30718		indicate a process group ID; the value zero shall indicate that no SIGURG
30719		signals are to be sent. If <i>fildev</i> does not refer to a socket, the results are
30720		unspecified.
30721	F_SETOWN	If <i>fildev</i> refers to a socket, atomically set the process ID or process group
30722		ID specified to receive SIGURG signals when out-of-band data is
30723		available, using the value of the third argument, <i>arg</i> , taken as type <code>int</code> .
30724		Positive values shall indicate a process ID; negative values, other than <code>-1</code> ,
30725		shall indicate a process group ID; the value zero shall indicate that no
30726		SIGURG signals are to be sent. If <i>fildev</i> does not refer to a socket, the
30727		results are unspecified.
30728	F_GETOWN_EX	If <i>fildev</i> refers to a socket, get the process ID or process group ID specified
30729		to receive SIGURG signals when out-of-band data is available, by setting
30730		the <i>type</i> and <i>pid</i> members of the <code>f_owner_ex</code> structure pointed to by the
30731		third argument, <i>arg</i> . The value of <i>type</i> shall be <code>F_OWNER_PID</code> or
30732		<code>F_OWNER_PGRP</code> to indicate that <i>pid</i> contains a process ID or a process
30733		group ID, respectively. The value of <i>pid</i> shall be zero if no SIGURG signals
30734		are to be sent. If <i>fildev</i> does not refer to a socket, the results are
30735		unspecified.
30736	F_SETOWN_EX	If <i>fildev</i> refers to a socket, set the process ID or process group ID specified
30737		to receive SIGURG signals when out-of-band data is available, using the
30738		value of the third argument, <i>arg</i> , taken as type pointer to <code>struct</code>
30739		<code>f_owner_ex</code> . The <i>type</i> and <i>pid</i> members of this structure shall be used as
30740		follows:
30741		• A <i>pid</i> value of zero shall indicate that no SIGURG signals are to be
30742		sent.
30743		• A <i>type</i> value of <code>F_OWNER_PID</code> and a positive <i>pid</i> value shall
30744		indicate that SIGURG signals are to be sent to the process ID
30745		specified in <i>pid</i> .
30746		• A <i>type</i> value of <code>F_OWNER_PGRP</code> and a positive <i>pid</i> value shall
30747		indicate that SIGURG signals are to be sent to the process group ID
30748		specified in <i>pid</i> .
30749		If <i>fildev</i> does not refer to a socket, the results are unspecified.
30750		For <code>F_SETOWN</code> and <code>F_SETOWN_EX</code> , each time a SIGURG signal is sent to the specified process
30751		or process group, permission checks equivalent to those performed by <code>kill()</code> shall be performed,
30752		as if <code>kill()</code> were called by a process with the same real user ID, effective user ID, and privileges
30753		that the process calling <code>fcntl()</code> has at the time of the call; if the <code>kill()</code> call would fail, no signal
30754		shall be sent. These permission checks may also be performed by the <code>fcntl()</code> call. If the process
30755		specified by <i>arg</i> later terminates, or the process group specified by <i>arg</i> later becomes empty,
30756		while still being specified to receive SIGURG signals when out-of-band data is available from

30757		<i>files</i> , then no signals shall be sent to any subsequently created process that has the same process
30758		ID or process group ID, regardless of permission; it is unspecified whether this is achieved by
30759		the equivalent of a <i>fcntl(files, F_SETOWN, 0)</i> call at the time the process terminates or is waited
30760		for or the process group becomes empty, or by other means.
30761		The following values for <i>cmd</i> are available for advisory record locking. Record locking shall be
30762		supported for regular files, and may be supported for other files.
30763	F_GETLK	Get any lock which blocks the process-owned file lock description
30764		pointed to by the third argument, <i>arg</i> , taken as a pointer to type <b>struct</b>
30765		<b>lock</b> , defined in <b>&lt;fcntl.h&gt;</b> . The information retrieved shall overwrite the
30766		information passed to <i>fcntl()</i> in the structure <b>lock</b> . If no lock is found
30767		that would prevent this lock from being created, then the structure shall
30768		be left unchanged except for the lock type in <i>l_type</i> which shall be set to
30769		F_UNLCK.
30770	F_SETLK	Set or clear a process-owned file lock according to the lock description
30771		pointed to by the third argument, <i>arg</i> , taken as a pointer to type <b>struct</b>
30772		<b>lock</b> , defined in <b>&lt;fcntl.h&gt;</b> . F_SETLK can establish shared (or read) locks
30773		(F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as remove
30774		either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are
30775		defined in <b>&lt;fcntl.h&gt;</b> . If a shared or exclusive lock cannot be set, <i>fcntl()</i>
30776		shall return immediately with a return value of -1.
30777	F_SETLKW	This command shall be equivalent to F_SETLK except that if a shared or
30778		exclusive lock is blocked by other locks, the thread shall wait until the
30779		request can be satisfied. If a signal that is to be caught is received while
30780		<i>fcntl()</i> is waiting for a region, <i>fcntl()</i> shall be interrupted. Upon return
30781		from the signal handler, <i>fcntl()</i> shall return -1 with <i>errno</i> set to [EINTR],
30782		and the lock operation shall not be done.
30783	F_OFD_GETLK	Get any lock which blocks the OFD-owned file lock description pointed to
30784		by the third argument, <i>arg</i> , taken as a pointer to type <b>struct lock</b> , defined
30785		in <b>&lt;fcntl.h&gt;</b> ; the application shall ensure that the <i>l_pid</i> member of the
30786		structure pointed to by <i>arg</i> is set to 0 on input. The information retrieved
30787		shall overwrite the information passed to <i>fcntl()</i> in the structure <b>lock</b> . If
30788		no lock is found that would prevent this lock from being created, then the
30789		structure shall be left unchanged except for the lock type in <i>l_type</i> which
30790		shall be set to F_UNLCK.
30791	F_OFD_SETLK	Set or clear an OFD-owned file lock according to the lock description
30792		pointed to by the third argument, <i>arg</i> , taken as a pointer to type <b>struct</b>
30793		<b>lock</b> , defined in <b>&lt;fcntl.h&gt;</b> ; the application shall ensure that the <i>l_pid</i>
30794		member of the structure pointed to by <i>arg</i> is set to 0 on input.
30795		F_OFD_SETLK can establish shared (or read) locks (F_RDLCK) or
30796		exclusive (or write) locks (F_WRLCK), as well as remove either type of
30797		lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in
30798		<b>&lt;fcntl.h&gt;</b> . If a shared or exclusive lock cannot be set, <i>fcntl()</i> shall return
30799		immediately with a return value of -1.
30800	F_OFD_SETLKW	This command shall be equivalent to F_OFD_SETLK except that if a
30801		shared or exclusive lock is blocked by other locks, the thread shall wait
30802		until the request can be satisfied. If a signal that is to be caught is received
30803		while <i>fcntl()</i> is waiting for a region, <i>fcntl()</i> shall be interrupted. Upon
30804		return from the signal handler, <i>fcntl()</i> shall return -1 with <i>errno</i> set to

30805 [EINTR], and the lock operation shall not be done.

30806 Additional implementation-defined values for *cmd* may be defined in `<fcntl.h>`. Their names  
30807 shall start with `F_`.

30808 When a shared lock is set on a segment of a file, other processes can set shared process-owned  
30809 locks, and other open file descriptions can be used to set shared OFD-owned locks, on that  
30810 segment or a portion of it. A shared process-owned lock shall prevent any other process from  
30811 setting an exclusive process-owned lock, and shall prevent any exclusive OFD-owned lock from  
30812 being set, on any portion of the protected area. A shared OFD-owned lock shall prevent any  
30813 other open file description from being used to set an exclusive OFD-owned lock, and shall  
30814 prevent any exclusive process-owned lock from being set, on any portion of the protected area.  
30815 A request for a shared lock shall fail if the file descriptor is not open for reading.

30816 An exclusive process-owned lock shall prevent any other process from setting a shared or  
30817 exclusive process-owned lock, and shall prevent any shared or exclusive OFD-owned lock from  
30818 being set, on any portion of the protected area. An exclusive OFD-owned lock shall prevent any  
30819 other open file description from being used to set a shared or exclusive OFD-owned lock, and  
30820 shall prevent any shared or exclusive process-owned lock from being set, on any portion of the  
30821 protected area. A request for an exclusive lock shall fail if the file descriptor is not open for  
30822 writing.

30823 The structure **flock** describes the type (*l\_type*), starting offset (*l\_whence*), relative offset (*l\_start*),  
30824 size (*l\_len*), and process ID (*l\_pid*) of the segment of the file to be affected.

30825 The value of *l\_whence* is `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, to indicate that the relative  
30826 offset *l\_start* bytes shall be measured from the start of the file, current position, or end of the file,  
30827 respectively. The value of *l\_len* is the number of consecutive bytes to be locked. The value of *l\_len*  
30828 may be negative (where the definition of **off\_t** permits negative values of *l\_len*). On input, the  
30829 *l\_pid* field shall be ignored for `F_GETLK`, `F_SETLK` and `F_SETLKW`; the application shall ensure  
30830 that it is set to zero for `F_OFD_GETLK`, `F_OFD_SETLK` and `F_OFD_SETLKW`. It is set by  
30831 `F_GETLK` and `F_OFD_GETLK` when identifying a blocking lock. After a successful `F_GETLK` or  
30832 `F_OFD_GETLK` request, when a blocking lock is found, the values returned in the **flock**  
30833 structure shall be as follows:

30834	<i>l_type</i>	Type of blocking lock found.
30835	<i>l_whence</i>	<code>SEEK_SET</code> .
30836	<i>l_start</i>	Start of the blocking lock.
30837	<i>l_len</i>	Length of the blocking lock.
30838	<i>l_pid</i>	Process ID of the process that holds the blocking lock if the blocking lock is a 30839 process-owned file lock, or <code>(pid_t)-1</code> if the blocking lock is an OFD-owned file 30840 lock.

30841 If the command is `F_SETLKW` or `F_OFD_SETLKW` and the thread needs to wait for a blocking  
30842 lock to be released, then the range of bytes to be locked shall be determined before the `fcntl()`  
30843 function blocks. If the file size or file descriptor seek offset change while `fcntl()` is blocked, this  
30844 shall not affect the range of bytes locked.

30845 If *l\_len* is positive, the area affected shall start at *l\_start* and end at *l\_start+l\_len-1*. If *l\_len* is  
30846 negative, the area affected shall start at *l\_start+l\_len* and end at *l\_start-1*. Locks may start and  
30847 extend beyond the current end of a file, but shall not extend before the beginning of the file. A  
30848 lock shall be set to extend to the largest possible value of the file offset for that file by setting  
30849 *l\_len* to 0. If such a lock also has *l\_start* set to 0 and *l\_whence* is set to `SEEK_SET`, the whole file  
30850 shall be locked.



30851		Each byte in the file can be locked either with one or more shared locks (F_RDLCK) or with one
30852		exclusive lock (F_WRLCK).
30853		Before a successful return from an F_SETLK or an F_SETLKW request when the calling process
30854		has previously existing process-owned locks on bytes in the region specified by the request, the
30855		previous shared or exclusive lock for each byte in the specified region shall be replaced by the
30856		new shared or exclusive lock. An F_SETLK or an F_SETLKW request (respectively) shall fail or
30857		block when another process has existing process-owned locks, or any open file description
30858		(including the one associated with <i>fildes</i> ) has existing OFD-owned locks, on bytes in the specified
30859		region and any of those locks conflicts with the requested lock.
30860		Before a successful return from an F_OFD_SETLK or an F_OFD_SETLKW request when the
30861		open file description associated with <i>fildes</i> has previously existing OFD-owned locks on bytes in
30862		the region specified by the request, the previous shared or exclusive lock for each byte in the
30863		specified region shall be replaced by the new shared or exclusive lock. An F_OFD_SETLK or an
30864		F_OFD_SETLKW request (respectively) shall fail or block when another open file description
30865		has existing OFD-owned locks, or any process (including the calling process) has existing
30866		process-owned locks, on bytes in the specified region and any of those locks conflicts with the
30867		requested lock.
30868		All process-owned locks associated with a file for a given process shall be removed when any
30869		file descriptor for that file is closed by that process (even if via a different open file description)
30870		or the process holding that file descriptor terminates. Process-owned locks shall not be inherited
30871		by a child process.
30872		All OFD-owned locks associated with a given open file description shall be removed when all
30873		file descriptors associated with that open file description have been closed (either directly or as a
30874		side-effect of, for example, process termination or FD_CLOEXEC). OFD-owned locks shall be
30875		shared across all file descriptors that are associated with the owning open file description,
30876		regardless of which process holds the file descriptor.
30877		A potential for deadlock occurs if a process or thread controlling a locked region is put to sleep
30878		by attempting to lock a region that has an existing conflicting lock. If the system detects that
30879		sleeping until a locked region is unlocked would cause a deadlock, <i>fcntl()</i> shall fail with an
30880		[EDEADLK] error. Deadlock detection may differ between process-owned locks and OFD-
30881		owned locks.
30882	XSI	The interaction between <i>fcntl()</i> and <i>lockf()</i> locks is unspecified.
30883		An unlock (F_UNLCK) request in which <i>l_len</i> is non-zero and the offset of the last byte of the
30884		requested segment is the maximum value for an object of type <b>off_t</b> , when the process (for
30885		F_SETLK and F_SETLKW) or open file description (for F_OFD_SETLK and F_OFD_SETLKW)
30886		has an existing lock in which <i>l_len</i> is 0 and which includes the last byte of the requested
30887		segment, shall be treated as a request to unlock from the start of the requested segment with an
30888		<i>l_len</i> equal to 0. Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the
30889		requested segment.
30890	SHM	When the file descriptor <i>fildes</i> refers to a shared memory object, the behavior of <i>fcntl()</i> shall be
30891		the same as for a regular file except the effect of the following values for the argument <i>cmd</i> is
30892		unspecified: F_SETFL, F_GETLK, F_SETLK, F_SETLKW, F_OFD_GETLK, F_OFD_SETLK, and
30893		F_OFD_SETLKW.
30894	TYM	If <i>fildes</i> refers to a typed memory object, the result of the <i>fcntl()</i> function is unspecified.

30895 **RETURN VALUE**30896 Upon successful completion, the value returned shall depend on *cmd* as follows:

30897 F\_DUPFD A new file descriptor.

30898 F\_DUPFD\_CLOEXEC

30899 A new file descriptor.

30900 F\_DUPFD\_CLOFORK

30901 A new file descriptor.

30902 F\_GETFD Value of flags defined in **<fcntl.h>**. The return value shall not be negative.30903 F\_SETFD Value other than  $-1$ .30904 F\_GETFL Value of file status flags and access modes. The return value shall not be  
30905 negative.30906 F\_SETFL Value other than  $-1$ .30907 F\_GETLK Value other than  $-1$ .30908 F\_SETLK Value other than  $-1$ .30909 F\_SETLKW Value other than  $-1$ .30910 F\_OFD\_GETLK Value other than  $-1$ .30911 F\_OFD\_SETLK Value other than  $-1$ .

30912 F\_OFD\_SETLKW

30913 Value other than  $-1$ .30914 F\_GETOWN Value of the socket owner process or process group; this shall not be  $-1$ .30915 F\_SETOWN Value other than  $-1$ .30916 F\_GETOWN\_EX Value other than  $-1$ .30917 F\_SETOWN\_EX Value other than  $-1$ .30918 Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.30919 **ERRORS**30920 The *fcntl()* function shall fail if:

30921 [EACCES] or [EAGAIN]

30922 The *cmd* argument is F\_SETLK, the type of lock (*l\_type*) is a shared (F\_RDLCK)  
30923 or exclusive (F\_WRLCK) lock, and the requested lock cannot be set because it  
30924 is blocked by an existing lock on the file.30925 [EAGAIN] The *cmd* argument is F\_OFD\_SETLK, the type of lock (*l\_type*) is a shared  
30926 (F\_RDLCK) or exclusive (F\_WRLCK) lock, and the requested lock cannot be  
30927 set because it is blocked by an existing lock on the file.30928 [EBADF] The *fildev* argument is not a valid open file descriptor; or the argument *cmd* is  
30929 F\_SETLK, F\_SETLKW, F\_OFD\_SETLK, or F\_OFD\_SETLKW, the type of lock,  
30930 *l\_type*, is a shared lock (F\_RDLCK), and *fildev* is not a valid file descriptor open  
30931 for reading, or the type of lock, *l\_type*, is an exclusive lock (F\_WRLCK), and  
30932 *fildev* is not a valid file descriptor open for writing.

30933	[EINTR]	The <i>cmd</i> argument is F_SETLKW or F_OFD_SETLKW and the function was interrupted by a signal.
30934		
30935	[EINVAL]	The <i>cmd</i> argument is invalid; or the <i>cmd</i> argument is F_DUPFD, F_DUPFD_CLOEXEC, or F_DUPFD_CLOFORK and <i>arg</i> is negative or is greater than or equal to {OPEN_MAX}; or the <i>cmd</i> argument is F_SETOWN_EX and the <i>type</i> member of the <b>f_owner_ex</b> structure pointed to by <i>arg</i> is invalid, or the <i>pid</i> member is negative and the <i>type</i> member is F_OWNER_PID or F_OWNER_PGRP; or the <i>cmd</i> argument is F_GETLK, F_SETLK, F_SETLKW, F_OFD_GETLK, F_OFD_SETLK, or F_OFD_SETLKW and the data pointed to by <i>arg</i> is not valid, or <i>fildev</i> refers to a file that does not support locking.
30936		
30937		
30938		
30939		
30940		
30941		
30942		
30943		
30944	[EMFILE]	The argument <i>cmd</i> is F_DUPFD, F_DUPFD_CLOEXEC, or F_DUPFD_CLOFORK and all file descriptors available to the process are currently open, or no file descriptors greater than or equal to <i>arg</i> are available.
30945		
30946		
30947	[ENOLCK]	The argument <i>cmd</i> is F_SETLK, F_SETLKW, F_OFD_SETLK, or F_OFD_SETLKW and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.
30948		
30949		
30950	[EOVERFLOW]	One of the values to be returned cannot be represented correctly.
30951	[EOVERFLOW]	The <i>cmd</i> argument is F_GETLK, F_SETLK, F_SETLKW, F_OFD_GETLK, F_OFD_SETLK, or F_OFD_SETLKW and the smallest or, if <i>l_len</i> is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type <b>off_t</b> .
30952		
30953		
30954		
30955	[ESRCH]	The <i>cmd</i> argument is F_SETOWN or F_SETOWN_EX and no process or process group can be found corresponding to that specified by <i>arg</i> .
30956		
30957		The <i>fcntl()</i> function may fail if:
30958	[EDEADLK]	The <i>cmd</i> argument is F_SETLKW or F_OFD_SETLKW, the type of lock ( <i>l_type</i> ) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock, the requested lock is blocked by an existing lock on the file, and the system determines that waiting for that lock to be released would cause a deadlock.
30959		
30960		
30961		
30962	[EINVAL]	The <i>cmd</i> argument is F_SETOWN and the value of <i>arg</i> is positive and is not valid as a process ID or the value of <i>arg</i> is negative and its absolute value is not valid as a process group ID; or the <i>cmd</i> argument is F_SETOWN_EX, the value of the <i>type</i> member of the <b>f_owner_ex</b> structure pointed to by <i>arg</i> is F_OWNER_PID, and the value of the <i>pid</i> member is not valid as a process ID; or the <i>cmd</i> argument is F_SETOWN_EX, the value of the <i>type</i> member of the <b>f_owner_ex</b> structure pointed to by <i>arg</i> is F_OWNER_PGRP, and the value of the <i>pid</i> member is not valid as a process group ID.
30963		
30964		
30965		
30966		
30967		
30968		
30969		
30970	[EPERM]	The <i>cmd</i> argument is F_SETOWN or F_SETOWN_EX and the calling process does not have permission to send a SIGURG signal to any process specified by <i>arg</i> .
30971		
30972		

## 30973 EXAMPLES

## 30974 Locking and Unlocking a File

30975 The following example demonstrates how to place a lock on bytes 100 to 109 of a file and then  
 30976 later remove it. F\_SETLK is used to perform a non-blocking lock request so that the process does  
 30977 not have to wait if an incompatible lock is held by another process; instead the process can take  
 30978 some other action.

```

30979 #include <stdlib.h>
30980 #include <unistd.h>
30981 #include <fcntl.h>
30982 #include <errno.h>
30983 #include <stdio.h>

30984 int
30985 main(int argc, char *argv[])
30986 {
30987     int fd;
30988     struct flock fl;

30989     fd = open("testfile", O_RDWR);
30990     if (fd == -1)
30991         /* Handle error */;

30992     /* Make a non-blocking request to place a write lock
30993        on bytes 100-109 of testfile */

30994     fl.l_type = F_WRLCK;
30995     fl.l_whence = SEEK_SET;
30996     fl.l_start = 100;
30997     fl.l_len = 10;

30998     if (fcntl(fd, F_SETLK, &fl) == -1) {
30999         if (errno == EACCES || errno == EAGAIN) {
31000             printf("Already locked by another process\n");
31001             /* We cannot get the lock at the moment */
31002         } else {
31003             /* Handle unexpected error */;
31004         }
31005     } else { /* Lock was granted... */

31006         /* Perform I/O on bytes 100 to 109 of file */

31007         /* Unlock the locked bytes */

31008         fl.l_type = F_UNLCK;
31009         fl.l_whence = SEEK_SET;
31010         fl.l_start = 100;
31011         fl.l_len = 10;
31012         if (fcntl(fd, F_SETLK, &fl) == -1)
31013             /* Handle error */;
31014     }
31015     exit(EXIT_SUCCESS);
31016 } /* main */

```

31017 **Setting the Close-on-Exec Flag**31018 The following example demonstrates how to set the close-on-exec flag for the file descriptor *fd*.

```

31019 #include <unistd.h>
31020 #include <fcntl.h>
31021 ...
31022     int flags;
31023
31024     flags = fcntl(fd, F_GETFD);
31025     if (flags == -1)
31026         /* Handle error */;
31027     flags |= FD_CLOEXEC;
31028     if (fcntl(fd, F_SETFD, flags) == -1)
31029         /* Handle error */;

```

31029 **APPLICATION USAGE**

31030 The *arg* values to `F_GETFD`, `F_SETFD`, `F_GETFL`, and `F_SETFL` all represent flag values to allow  
 31031 for future growth. Applications using these functions should do a read-modify-write operation  
 31032 on them, rather than assuming that only the values defined by this volume of POSIX.1-2024 are  
 31033 valid. It is a common error to forget this, particularly in the case of `F_SETFD`. Some  
 31034 implementations set additional file status flags to advise the application of default behavior,  
 31035 even though the application did not request these flags.

31036 In order to set both `FD_CLOEXEC` and `FD_CLOFORK` when duplicating a file descriptor,  
 31037 applications should use `F_DUPFD_CLOFORK` to obtain the new file descriptor with  
 31038 `FD_CLOFORK` already set, and then use `F_SETFD` to set the `FD_CLOEXEC` flag on the new  
 31039 descriptor. (The alternative of first using `F_DUPFD_CLOEXEC` and then setting `FD_CLOFORK`  
 31040 with `F_SETFD` has a timing window where another thread could create a child process which  
 31041 inherits the new descriptor because `FD_CLOFORK` has not yet been set.)

31042 The `FD_CLOFORK` flag takes effect for all child processes, not just those created using `fork()` or  
 31043 `_Fork()`.

31044 On implementations where process IDs can be greater than `{INT_MAX}`, `F_SETOWN` cannot be  
 31045 used with process IDs greater than `{INT_MAX}` or process group IDs greater than `{INT_MAX}+1`  
 31046 because the value is passed to `fcntl()` in an argument of type `int`. In this situation,  
 31047 `F_SETOWN_EX` should be used instead.

31048 Similarly, if a process ID greater than `{INT_MAX}` or a process group ID greater than  
 31049 `{INT_MAX}+1` has been set to receive SIGURG signals (using `F_SETOWN_EX`), `F_GETOWN`  
 31050 cannot be used to obtain the value because `fcntl()` returns the value as type `int` and will thus  
 31051 give an `[EOVERFLOW]` error for such values. `F_GETOWN_EX` should be used instead.

31052 Note that the convention of negating a process group ID is only used with `F_SETOWN` and  
 31053 `F_GETOWN`; the `pid` member of the `f_owner_ex` structure used with `F_SETOWN_EX` and  
 31054 `F_GETOWN_EX` is not negated when it specifies a process group ID.

31055 On systems which do not perform permission checks at the time of an `fcntl()` call with  
 31056 `F_SETOWN` or `F_SETOWN_EX`, if the permission checks performed at the time the signal is sent  
 31057 disallow sending the signal to any process, the process that called `fcntl()` has no way of  
 31058 discovering that this has happened. A call to `kill()` with signal 0 can be used as a prior check of  
 31059 permissions, although this is no guarantee that permission will be granted at the time a signal is  
 31060 sent, since the target process(es) could change user IDs or privileges in the meantime.

31061 Record-locking should not be used in combination with buffered standard I/O streams (see  
 31062 [Section 2.5](#), on page 521). Instead, non-buffered I/O should be used. Unexpected results may

31063 occur in processes that do buffering in the user address space. The process may later read/write  
 31064 data which is/was locked. Functions that operate on standard I/O streams are the most  
 31065 common source of such buffering.

#### 31066 RATIONALE

31067 The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number  
 31068 of arguments. It is used because System V uses pointers for the implementation of file locking  
 31069 functions.

31070 This volume of POSIX.1-2024 permits concurrent read and write access to file data using the  
 31071 *fcntl()* function; this is a change from the 1984 /usr/group standard and early proposals.  
 31072 Without concurrency controls, this feature may not be fully utilized without occasional loss of  
 31073 data.

31074 Data losses occur in several ways. One case occurs when several processes try to update the  
 31075 same record, without sequencing controls; several updates may occur in parallel and the last  
 31076 writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing  
 31077 reorganization. Without exclusive use to the tree segment by the updating process, other reading  
 31078 processes chance getting lost in the database when the index blocks are split, condensed,  
 31079 inserted, or deleted. While *fcntl()* is useful for many applications, it is not intended to be overly  
 31080 general and does not handle the bit-tree example well.

31081 This facility is only required for regular files because it is not appropriate for many devices such  
 31082 as terminals and network connections.

31083 Since *fcntl()* works with “any file descriptor associated with that file, however it is obtained”,  
 31084 the file descriptor may have been inherited through a *fork()* or *exec* operation and thus may  
 31085 affect a file that another process also has open.

31086 The use of the open file description to identify what to lock requires extra calls and presents  
 31087 problems if several processes are sharing an open file description, but there are too many  
 31088 implementations of the existing mechanism for this volume of POSIX.1-2024 to use different  
 31089 specifications.

31090 Another consequence of this model is that closing any file descriptor for a given file (whether or  
 31091 not it is the same open file description that created the lock) causes the locks on that file to be  
 31092 relinquished for that process. Equivalently, any close for any file/process pair relinquishes the  
 31093 locks owned on that file for that process. But note that while an open file description may be  
 31094 shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through  
 31095 one of the *exec* functions.

31096 The identification of a machine in a network environment is outside the scope of this volume of  
 31097 POSIX.1-2024. Thus, an *l\_sysid* member, such as found in System V, is not included in the locking  
 31098 structure.

31099 Changing of lock types can result in a previously locked region being split into smaller regions.

31100 Mandatory locking was a major feature of the 1984 /usr/group standard.

31101 For advisory file record locking to be effective, all processes that have access to a file must  
 31102 cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode  
 31103 record locking is important when it cannot be assumed that all processes are cooperating. For  
 31104 example, if one user uses an editor to update a file at the same time that a second user executes  
 31105 another process that updates the same file and if only one of the two processes is using advisory  
 31106 locking, the processes are not cooperating. Enforcement-mode record locking would protect  
 31107 against accidental collisions.

31108 Secondly, advisory record locking requires a process using locking to bracket each I/O operation

31109 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a  
 31110 process can lock the file once and unlock when all I/O operations have been completed.  
 31111 Enforcement-mode record locking provides a base that can be enhanced; for example, with  
 31112 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so  
 31113 other processes could read it, but none of them could write it.

31114 Mandatory locks were omitted for several reasons:

- 31115 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most  
 31116 implementations; this was confusing, at best.
- 31117 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
- 31118 3. Any publicly readable file could be locked by anyone. Many historical implementations  
 31119 keep the password database in a publicly readable file. A malicious user could thus  
 31120 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
- 31121 4. Some demand-paged historical implementations offer memory mapped files, and  
 31122 enforcement cannot be done on that type of file.

31123 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a  
 31124 timeout facility in applications requiring it. This is useful in deadlock detection. Since  
 31125 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was  
 31126 made optional.

31127 The F\_SETOWN\_EX and F\_GETOWN\_EX values for *cmd* and the associated **f\_owner\_ex**  
 31128 structure were adopted from the GNU C library. In addition to the values F\_OWNER\_PID and  
 31129 F\_OWNER\_PGRP for the *type* member, this also has F\_OWNER\_TID to specify that the *pid*  
 31130 member contains a thread ID. However, this relies on thread IDs being representable in a **pid\_t**  
 31131 and so was not included in POSIX.1-2024. The aim of adding F\_SETOWN\_EX and  
 31132 F\_GETOWN\_EX was to address the inability of F\_SETOWN and F\_GETOWN to handle process  
 31133 IDs greater than {INT\_MAX} and process group IDs greater than {INT\_MAX}+1, and this need is  
 31134 satisfied without including F\_OWNER\_TID.

### 31135 FUTURE DIRECTIONS

31136 None.

### 31137 SEE ALSO

31138 *alarm()*, *close()*, *exec*, *kill()*, *open()*, *sigaction()*

31139 XBD <fcntl.h>, <signal.h>

### 31140 CHANGE HISTORY

31141 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 31142 Issue 5

31143 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 31144 Threads Extension.

31145 Large File Summit extensions are added.

#### 31146 Issue 6

31147 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

31148 The following new requirements on POSIX implementations derive from alignment with the  
 31149 Single UNIX Specification:

- 31150 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 31151 required for conforming implementations of previous POSIX specifications, it was not  
 31152 required for UNIX applications.

31153 • In the DESCRIPTION, sentences describing behavior when *l\_len* is negative are now  
 31154 mandated, and the description of unlock (F\_UNLOCK) when *l\_len* is non-negative is  
 31155 mandated.

31156 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd*  
 31157 is invalid, and two [EOVERFLOW] error conditions are added.

31158 The F\_GETOWN and F\_SETOWN values are added for sockets.

31159 The following changes were made to align with the IEEE P1003.1a draft standard:

31160 • Clarification is added that the extent of the bytes locked is determined prior to the  
 31161 blocking action.

31162 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 31163 *fcntl()* results are unspecified for typed memory objects.

31164 The normative text is updated to avoid use of the term “must” for application requirements.

31165 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/29 is applied, adding the example to the  
 31166 EXAMPLES section.

#### 31167 Issue 7

31168 Austin Group Interpretation 1003.1-2001 #150 is applied, clarifying the file status flags returned  
 31169 when *cmd* is F\_GETFL.

31170 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the  
 31171 FD\_CLOEXEC flag atomically at *open()*, and adding the F\_DUPFD\_CLOEXEC flag.

31172 The optional `<unistd.h>` header is removed from this function, since `<fcntl.h>` now defines  
 31173 SEEK\_SET, SEEK\_CUR, and SEEK\_END as part of the Base.

31174 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0116 [141] is applied.

31175 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0105 [835], XSH/TC2-2008/0106 [677],  
 31176 XSH/TC2-2008/0107 [484], XSH/TC2-2008/0108 [675], and XSH/TC2-2008/0109 [675,677] are  
 31177 applied.

#### 31178 Issue 8

31179 Austin Group Defect 695 is applied, adding an atomicity requirement to the F\_SETOWN  
 31180 operation.

31181 Austin Group Defects 768 and 1671 are applied, adding OFD-owned file locks.

31182 Austin Group Defect 1203 is applied, changing some wording in the RETURN VALUE section to  
 31183 use “shall”.

31184 Austin Group Defects 1274 and 1670 are applied, adding F\_GETOWN\_EX and F\_SETOWN\_EX.

31185 Austin Group Defect 1318 is applied, adding FD\_CLOFORK and F\_DUPFD\_CLOFORK.



31186 **NAME**31187 `fdatasync` — synchronize the data of a file (**REALTIME**)31188 **SYNOPSIS**

```
31189 SIO #include <unistd.h>
31190 int fdatasync(int fildes);
```

31191 **DESCRIPTION**

31192 The `fdatasync()` function shall force all currently queued I/O operations associated with the file  
 31193 indicated by file descriptor `fildes` to the synchronized I/O completion state.

31194 The functionality shall be equivalent to `fsync()` with the symbol `_POSIX_SYNCHRONIZED_IO`  
 31195 defined, with the exception that all I/O operations shall be completed as defined for  
 31196 synchronized I/O data integrity completion.

31197 **RETURN VALUE**

31198 If successful, the `fdatasync()` function shall return the value 0; otherwise, the function shall return  
 31199 the value `-1` and set `errno` to indicate the error. If the `fdatasync()` function fails, outstanding I/O  
 31200 operations are not guaranteed to have been completed.

31201 **ERRORS**

31202 The `fdatasync()` function shall fail if:

31203 [EBADF] The `fildes` argument is not a valid file descriptor.

31204 [EINVAL] This implementation does not support synchronized I/O for this file.

31205 In the event that any of the queued I/O operations fail, `fdatasync()` shall return the error  
 31206 conditions defined for `read()` and `write()`.

31207 **EXAMPLES**

31208 None.

31209 **APPLICATION USAGE**

31210 Note that even if the file descriptor is not open for writing, if there are any pending write  
 31211 requests on the underlying file, then that I/O will be completed prior to the return of `fdatasync()`.

31212 An application that modifies a directory, for example, by creating a file in the directory, can  
 31213 invoke `fdatasync()` on the directory to ensure that the directory's entries are synchronized,  
 31214 although for most applications this should not be necessary (see XBD Section 4.11, on page 98).

31215 **RATIONALE**

31216 None.

31217 **FUTURE DIRECTIONS**

31218 None.

31219 **SEE ALSO**

31220 `aio_fsync()`, `fcntl()`, `fsync()`, `open()`, `read()`, `write()`

31221 XBD `<unistd.h>`

31222 **CHANGE HISTORY**

31223 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

31224 **Issue 6**

31225 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 31226 implementation does not support the Synchronized Input and Output option.

31227 The `fdatasync()` function is marked as part of the Synchronized Input and Output option.

31228 **Issue 7**

31229 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0110 [501] is applied.

31230 **Issue 8**

31231 Austin Group Defect 672 is applied, changing the APPLICATION USAGE section.

31232 **NAME**

31233 fdim, fdimf, fdiml — compute positive difference between two floating-point numbers

31234 **SYNOPSIS**

```
31235 #include <math.h>
31236 double fdim(double x, double y);
31237 float fdimf(float x, float y);
31238 long double fdiml(long double x, long double y);
```

31239 **DESCRIPTION**

31240 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 31241 conflict between the requirements described here and the ISO C standard is unintentional. This  
 31242 volume of POSIX.1-2024 defers to the ISO C standard.

31243 These functions shall determine the positive difference between their arguments. If  $x$  is greater  
 31244 than  $y$ ,  $x-y$  is returned. If  $x$  is less than or equal to  $y$ ,  $+0$  is returned.

31245 An application wishing to check for error situations should set *errno* to zero and call  
 31246 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 31247 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 31248 zero, an error has occurred.

31249 **RETURN VALUE**

31250 Upon successful completion, these functions shall return the positive difference value.

31251 If  $x-y$  is positive and overflows, a range error shall occur and *fdim()*, *fdimf()*, and *fdiml()* shall  
 31252 return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.

31253 If the correct value would cause underflow, a range error may occur, and *fdim()*, *fdimf()*, and  
 31254 MXX *fdiml()* shall return the correct value, or (if the IEC 60559 Floating-Point option is not  
 31255 supported) an implementation-defined value no greater in magnitude than DBL\_MIN,  
 31256 FLT\_MIN, and LDBL\_MIN, respectively.

31257 MX If  $x$  or  $y$  is NaN, a NaN shall be returned.

31258 **ERRORS**31259 The *fdim()* function shall fail if:

31260 Range Error The result overflows.

31261 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 31262 then *errno* shall be set to [ERANGE]. If the integer expression  
 31263 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 31264 floating-point exception shall be raised.

31265 The *fdim()* function may fail if:

31266 Range Error The result underflows.

31267 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 31268 then *errno* shall be set to [ERANGE]. If the integer expression  
 31269 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 31270 floating-point exception shall be raised.

31271 **EXAMPLES**

31272 None.

31273 **APPLICATION USAGE**

31274 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
31275 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

31276 **RATIONALE**

31277 None.

31278 **FUTURE DIRECTIONS**

31279 None.

31280 **SEE ALSO**31281 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [fmax\(\)](#), [fmin\(\)](#)31282 [Section 4.23](#) (on page 109), [<math.h>](#)31283 **CHANGE HISTORY**

31284 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31285 **Issue 7**

31286 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0119 [68,428] and  
31287 XSH/TC1-2008/0120 [68,428] are applied.

31288 **NAME**

31289 fdopen — associate a stream with a file descriptor

31290 **SYNOPSIS**

```
31291 CX #include <stdio.h>
31292 FILE *fdopen(int fildev, const char *mode);
```

31293 **DESCRIPTION**31294 The *fdopen()* function shall associate a stream with a file descriptor.

31295 The *mode* argument points to a character string that is valid for *fopen()*. If the string begins with  
 31296 one of the following characters, then the stream shall be associated with *fildev* as specified.  
 31297 Otherwise, the behavior is undefined.

31298 'r' If *mode* includes '+', the associated stream shall be open for update (reading and  
 31299 writing); otherwise, the stream shall be open for reading only. If the open file description  
 31300 referenced by *fildev* has O\_APPEND set, it shall remain set.

31301 'w' If *mode* includes '+', the associated stream shall be open for update (reading and  
 31302 writing); otherwise, the stream shall be open for writing only. The file shall not be  
 31303 truncated by the *fdopen()* call. If the open file description referenced by *fildev* has  
 31304 O\_APPEND set, it shall remain set.

31305 'a' If *mode* includes '+', the associated stream shall be open for update (reading and  
 31306 writing); otherwise, the stream shall be open for writing only. If the open file description  
 31307 referenced by *fildev* has O\_APPEND clear, it is unspecified whether O\_APPEND is set by  
 31308 the *fdopen()* call or remains clear.

31309 The presence of 'x' in *mode* shall have no effect. The FD\_CLOEXEC flag of *fildev* shall be  
 31310 unchanged if 'e' is not present, and shall be set by the *fdopen()* call if 'e' is present.

31311 Additional values for the *mode* argument may be supported by an implementation.

31312 The application shall ensure that the mode of the stream as expressed by the *mode* argument is  
 31313 allowed by the file access mode of the open file description to which *fildev* refers. The file  
 31314 position indicator associated with the new stream is set to the position indicated by the file offset  
 31315 associated with the file descriptor.

31316 The error and end-of-file indicators for the stream shall be cleared. The *fdopen()* function may  
 31317 cause the last data access timestamp of the underlying file to be marked for update.

31318 SHM If *fildev* refers to a shared memory object, the result of the *fdopen()* function is unspecified.31319 TYM If *fildev* refers to a typed memory object, the result of the *fdopen()* function is unspecified.

31320 The *fdopen()* function shall preserve the offset maximum previously set for the open file  
 31321 description corresponding to *fildev*.

31322 **RETURN VALUE**

31323 Upon successful completion, *fdopen()* shall return a pointer to a stream; otherwise, a null pointer  
 31324 shall be returned and *errno* set to indicate the error.

31325 **ERRORS**31326 The *fdopen()* function shall fail if:

31327 [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

31328 The *fdopen()* function may fail if:

- 31329 [EBADF] The *fildev* argument is not a valid file descriptor.
- 31330 [EINVAL] The *mode* argument is not a valid mode.
- 31331 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.
- 31332 [ENOMEM] Insufficient space to allocate a buffer.

**31333 EXAMPLES**

31334 None.

**31335 APPLICATION USAGE**

31336 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but  
31337 do not return streams.

**31338 RATIONALE**

31339 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, *fcntl()*, or *socket()*;  
31340 inherited through *fork()*, *posix\_spawn()*, or *exec*; or perhaps obtained by other means.

31341 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, write ('w')  
31342 mode cannot create or truncate a file, and append ('a') mode cannot create a file. Inclusion of a  
31343 'b' in the *mode* argument is allowed for consistency with *fopen()*; the 'b' has no effect on the  
31344 resulting stream. Implementations differ as to whether specifying append ('a') mode causes  
31345 the O\_APPEND flag to be set if it was clear, but they are encouraged to do so. Since *fdopen()*  
31346 does not create a file, the 'x' mode modifier is silently ignored. The 'e' mode modifier is not  
31347 strictly necessary for *fdopen()*, since FD\_CLOEXEC must not be changed when it is absent;  
31348 however, it is standardized here since that modifier is necessary to avoid a data race in multi-  
31349 threaded applications using *freopen()*, and consistency dictates that all functions accepting *mode*  
31350 strings should allow the same set of strings.

**31351 FUTURE DIRECTIONS**

31352 None.

**31353 SEE ALSO**

31354 [Section 2.5.1](#) (on page 522), *fclose()*, *fmemopen()*, *fopen()*, *open()*, *open\_memstream()*,  
31355 *posix\_spawn()*, *socket()*

31356 XBD <stdio.h>

**31357 CHANGE HISTORY**

31358 First released in Issue 1. Derived from Issue 1 of the SVID.

**31359 Issue 5**

31360 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

31361 Large File Summit extensions are added.

**31362 Issue 6**

31363 The following new requirements on POSIX implementations derive from alignment with the  
31364 Single UNIX Specification:

- 31365 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include  
31366 binary streams.
- 31367 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset  
31368 maximum in the open file description.
- 31369 • All errors identified in the ERRORS section are added.

31370 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st\_atime* to be  
31371 updated.

31372 The following changes were made to align with the IEEE P1003.1a draft standard:

31373 • Clarification is added that it is the responsibility of the application to ensure that the mode  
31374 is compatible with the open file descriptor.

31375 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
31376 *fdopen()* results are unspecified for typed memory objects.

31377 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/30 is applied, making corrections to the  
31378 RATIONALE.

31379 **Issue 7**

31380 SD5-XSH-ERN-149 is applied, adding the {STREAM\_MAX} [EMFILE] error condition.

31381 Changes are made related to support for finegrained timestamps.

31382 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0121 [409] is applied.

31383 **Issue 8**

31384 Austin Group Defects 411 and 1526 are applied, changing the requirements for the *mode*  
31385 argument.

31386 **NAME**

31387 fdopendir, opendir — open directory associated with file descriptor

31388 **SYNOPSIS**

```
31389 #include <dirent.h>
31390 DIR *fdopendir(int fd);
31391 DIR *opendir(const char *dirname);
```

31392 **DESCRIPTION**

31393 The *fdopendir()* function shall be equivalent to the *opendir()* function except that the directory is  
 31394 specified by a file descriptor rather than by a name. The file offset associated with the file  
 31395 descriptor at the time of the call determines which entries are returned.

31396 Upon successful return from *fdopendir()*, the file descriptor is under the control of the system,  
 31397 and if any attempt is made to close the file descriptor, or to modify the state of the associated  
 31398 XSI description, other than by means of *closedir()*, *readdir()*, *readdir\_r()*, *rewinddir()*, or *seekdir()*, the  
 31399 behavior is undefined. Upon calling *closedir()* the file descriptor shall be closed.

31400 It is unspecified whether the FD\_CLOEXEC flag will be set on the file descriptor by a successful  
 31401 call to *fdopendir()* if it was not previously set. However, the flag shall not be cleared if it was  
 31402 previously set.

31403 The *opendir()* function shall open a directory stream corresponding to the directory named by  
 31404 the *dirname* argument. The directory stream shall be positioned at the first entry. If *opendir()*  
 31405 opens a file descriptor for *dirname* to associate with the returned stream:

- 31406 • The descriptor shall be allocated as if the O\_DIRECTORY and O\_CLOEXEC flags were  
 31407 passed to *open()*.
- 31408 • The descriptor shall be subject to the limit of {OPEN\_MAX} file descriptors available to the  
 31409 process.

31410 **RETURN VALUE**

31411 Upon successful completion, these functions shall return a pointer to an object of type **DIR**.  
 31412 Otherwise, these functions shall return a null pointer and set *errno* to indicate the error.

31413 **ERRORS**

31414 The *fdopendir()* function shall fail if:

- 31415 [EBADF] The *fd* argument is not a valid file descriptor open for reading.
- 31416 [ENOTDIR] The descriptor *fd* is not associated with a directory.

31417 The *opendir()* function shall fail if:

- 31418 [EACCES] Search permission is denied for the component of the path prefix of *dirname* or  
 31419 read permission is denied for *dirname*.
- 31420 [ELOOP] A loop exists in symbolic links encountered during resolution of the *dirname*  
 31421 argument.
- 31422 [ENAMETOOLONG]  
 31423 The length of a component of a pathname is longer than {NAME\_MAX}.
- 31424 [ENOENT] A component of *dirname* does not name an existing directory or *dirname* is an  
 31425 empty string.
- 31426 [ENOTDIR] A component of *dirname* names an existing file that is neither a directory nor a  
 31427 symbolic link to a directory.



- 31428 The *opendir()* function may fail if:
- 31429 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
31430 resolution of the *dirname* argument.
- 31431 [EMFILE] All file descriptors available to the process are currently open.
- 31432 [ENAMETOOLONG]  
31433 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
31434 symbolic link produced an intermediate result with a length that exceeds  
31435 {PATH\_MAX}.
- 31436 [ENFILE] Too many files are currently open in the system.

31437 **EXAMPLES**31438 **Open a Directory Stream**

31439 The following program fragment demonstrates how the *opendir()* function is used.

```
31440 #include <dirent.h>
31441 ...
31442     DIR *dir;
31443     struct dirent *dp;
31444 ...
31445     if ((dir = opendir(".")) == NULL) {
31446         perror("Cannot open .");
31447         exit(1);
31448     }
31449     while ((dp = readdir(dir)) != NULL) {
31450         ...
```

31451 **Find And Open a File**

31452 The following program searches through a given directory looking for files whose name does  
31453 not begin with a dot and whose size is larger than 1 MiB.

```
31454 #include <stdio.h>
31455 #include <dirent.h>
31456 #include <fcntl.h>
31457 #include <sys/stat.h>
31458 #include <stdint.h>
31459 #include <stdlib.h>
31460 #include <unistd.h>
31461 int
31462 main(int argc, char *argv[])
31463 {
31464     struct stat statbuf;
31465     DIR *d;
31466     struct dirent *dp;
31467     int dfd, ffd;
31468     if ((d = fdopendir((dfd = open("./tmp", O_RDONLY)))) == NULL) {
31469         fprintf(stderr, "Cannot open ./tmp directory\n");
31470         exit(1);
31471     }
31472     while ((dp = readdir(d)) != NULL) {
```

```

31473         if (dp->d_name[0] == '.')
31474             continue;
31475         /* there is a possible race condition here as the file
31476          * could be renamed between the readdir and the open */
31477         if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
31478             perror(dp->d_name);
31479             continue;
31480         }
31481         if (fstat(ffd, &statbuf) == 0 && statbuf.st_size > (1024*1024)) {
31482             /* found it ... */
31483             printf("%s: %jdK\n", dp->d_name,
31484                 (intmax_t)(statbuf.st_size / 1024));
31485         }
31486         close(ffd);
31487     }
31488     closedir(d); // note this implicitly closes dfd
31489     return 0;
31490 }

```

### APPLICATION USAGE

The `opendir()` function should be used in conjunction with `readdir()`, `closedir()`, and `rewinddir()` to examine the contents of the directory (see the EXAMPLES section in `readdir()`). This method is recommended for portability.

### RATIONALE

The purpose of the `fdopendir()` function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `opendir()`, resulting in unspecified behavior.

Based on historical implementations, the rules about file descriptors apply to directory streams as well. However, this volume of POSIX.1-2024 does not mandate that `opendir()` opens a file descriptor to associate with the stream; this may instead be done by the first call to `dirfd()`, thus avoiding the need to allocate a file descriptor if `dirfd()` is never called. Once a file descriptor has been associated with the stream, it is mandatory that `closedir()` deallocate the file descriptor. If `opendir()` opens a file descriptor to associate with the stream, it behaves as if the `O_CLOEXEC` flag for `open()` had been used, so that the `FD_CLOEXEC` flag is set for the file descriptor. If `fdopendir()` is used to create a directory stream, it is unspecified whether the `FD_CLOEXEC` flag on the file descriptor specified by the `fd` argument is set or left unchanged.

The directory entries for dot and dot-dot are optional. This volume of POSIX.1-2024 does not provide a way to test *a priori* for their existence because an application that is portable must be written to look for (and usually ignore) those entries. Writing code that presumes that they are the first two entries does not always work, as many implementations permit them to be other than the first two entries, with a “normal” entry preceding them. There is negligible value in providing a way to determine what the implementation does because the code to deal with dot and dot-dot must be written in any case and because such a flag would add to the list of those flags (which has proven in itself to be objectionable) and might be abused.

Since the structure and buffer allocation, if any, for directory operations are defined by the implementation, this volume of POSIX.1-2024 imposes no portability requirements for erroneous program constructs, erroneous data, or the use of unspecified values such as the use or referencing of a `dirp` value or a `dirent` structure value after a directory stream has been closed or after a `fork()` or one of the `exec` function calls.

31521 **FUTURE DIRECTIONS**

31522 None.

31523 **SEE ALSO**31524 *closedir()*, *dirfd()*, *fstatat()*, *open()*, *posix\_getdents()*, *readdir()*, *rewinddir()*, *symlink()*31525 XBD <*dirent.h*>, <*fcntl.h*>, <*sys/types.h*>31526 **CHANGE HISTORY**

31527 First released in Issue 2.

31528 **Issue 6**31529 In the SYNOPSIS, the optional include of the <*sys/types.h*> header is removed.31530 The following new requirements on POSIX implementations derive from alignment with the  
31531 Single UNIX Specification:31532 • The requirement to include <*sys/types.h*> has been removed. Although <*sys/types.h*> was  
31533 required for conforming implementations of previous POSIX specifications, it was not  
31534 required for UNIX applications.

31535 • The [ELOOP] mandatory error condition is added.

31536 • A second [ENAMETOOLONG] is added as an optional error condition.

31537 The following changes were made to align with the IEEE P1003.1a draft standard:

31538 • The [ELOOP] optional error condition is added.

31539 **Issue 7**

31540 Austin Group Interpretation 1003.1-2001 #143 is applied.

31541 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

31542 The *fdopendir()* function is added from The Open Group Technical Standard, 2006, Extended API  
31543 Set Part 2.

31544 An additional example is added.

31545 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0122 [422] and XSH/TC1-2008/0123  
31546 [324] are applied.31547 **Issue 8**31548 Austin Group Defect 368 is applied, adding a requirement for FD\_CLOEXEC to be set by  
31549 *opendir()* if it associates a file descriptor with the returned stream.31550 Austin Group Defect 411 is applied, clarifying that FD\_CLOEXEC is not cleared by *fdopendir()* if  
31551 it was previously set.31552 Austin Group Defect 697 is applied, adding *posix\_getdents()* to the SEE ALSO section.31553 Austin Group Defect 1360 is applied, clarifying that type **DIR** always has the ability to store a  
31554 file descriptor; what is optional is whether one is opened by *opendir()*.

31555 **NAME**

31556           feclearexcept — clear floating-point exception

31557 **SYNOPSIS**

31558           #include &lt;fenv.h&gt;

31559           int feclearexcept(int *excepts*);31560 **DESCRIPTION**

31561 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
31562       conflict between the requirements described here and the ISO C standard is unintentional. This  
31563       volume of POSIX.1-2024 defers to the ISO C standard.

31564       The *feclearexcept()* function shall attempt to clear the supported floating-point exceptions  
31565       represented by *excepts*.

31566 **RETURN VALUE**

31567       If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept()*  
31568       shall return zero. Otherwise, it shall return a non-zero value.

31569 **ERRORS**

31570       No errors are defined.

31571 **EXAMPLES**

31572       None.

31573 **APPLICATION USAGE**

31574       None.

31575 **RATIONALE**

31576       None.

31577 **FUTURE DIRECTIONS**

31578       None.

31579 **SEE ALSO**31580       *fegetexceptflag()*, *feraiseexcept()*, *fetestexcept()*

31581       XBD &lt;fenv.h&gt;

31582 **CHANGE HISTORY**

31583       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31584       ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

31585 **NAME**

31586 fegetenv, fesetenv — get and set current floating-point environment

31587 **SYNOPSIS**

```
31588 #include <fenv.h>
31589 int fegetenv(fenv_t *envp);
31590 int fesetenv(const fenv_t *envp);
```

31591 **DESCRIPTION**

31592 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 31593 conflict between the requirements described here and the ISO C standard is unintentional. This  
 31594 volume of POSIX.1-2024 defers to the ISO C standard.

31595 The *fegetenv()* function shall attempt to store the current floating-point environment in the object  
 31596 pointed to by *envp*.

31597 The *fesetenv()* function shall attempt to establish the floating-point environment represented by  
 31598 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to  
 31599 *fegetenv()* or *fehldexcept()*, or equal a floating-point environment macro. The *fesetenv()* function  
 31600 does not raise floating-point exceptions, but only installs the state of the floating-point status  
 31601 flags represented through its argument.

31602 **RETURN VALUE**

31603 If the representation was successfully stored, *fegetenv()* shall return zero. Otherwise, it shall  
 31604 return a non-zero value. If the environment was successfully established, *fesetenv()* shall return  
 31605 zero. Otherwise, it shall return a non-zero value.

31606 **ERRORS**

31607 No errors are defined.

31608 **EXAMPLES**

31609 None.

31610 **APPLICATION USAGE**

31611 None.

31612 **RATIONALE**

31613 None.

31614 **FUTURE DIRECTIONS**

31615 None.

31616 **SEE ALSO**31617 *fehldexcept()*, *feupdateenv()*31618 XBD **<fenv.h>**31619 **CHANGE HISTORY**

31620 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31621 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

31622 **NAME**

31623 fegetexceptflag, fesetexceptflag — get and set floating-point status flags

31624 **SYNOPSIS**

31625 #include &lt;fenv.h&gt;

31626 int fegetexceptflag(fexcept\_t \*flagp, int excepts);

31627 int fesetexceptflag(const fexcept\_t \*flagp, int excepts);

31628 **DESCRIPTION**31629 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31630 conflict between the requirements described here and the ISO C standard is unintentional. This  
31631 volume of POSIX.1-2024 defers to the ISO C standard.31632 The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of  
31633 the states of the floating-point status flags indicated by the argument *excepts* in the object  
31634 pointed to by the argument *flagp*.31635 The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the  
31636 argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by  
31637 *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument  
31638 represented at least those floating-point exceptions represented by the argument *excepts*. This  
31639 function does not raise floating-point exceptions, but only sets the state of the flags.31640 **RETURN VALUE**31641 If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it  
31642 shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions  
31643 were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero  
31644 value.31645 **ERRORS**

31646 No errors are defined.

31647 **EXAMPLES**

31648 None.

31649 **APPLICATION USAGE**

31650 None.

31651 **RATIONALE**

31652 None.

31653 **FUTURE DIRECTIONS**

31654 None.

31655 **SEE ALSO**31656 [feclearexcept\(\)](#), [feraiseexcept\(\)](#), [fetestexcept\(\)](#)31657 XBD [<fenv.h>](#)31658 **CHANGE HISTORY**

31659 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31660 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

31661 **NAME**

31662 fegetround, fesetround — get and set current rounding direction

31663 **SYNOPSIS**

```
31664 #include <fenv.h>
31665 int fegetround(void);
31666 int fesetround(int round);
```

31667 **DESCRIPTION**

31668 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 31669 conflict between the requirements described here and the ISO C standard is unintentional. This  
 31670 volume of POSIX.1-2024 defers to the ISO C standard.

31671 The *fegetround()* function shall get the current rounding direction.

31672 The *fesetround()* function shall establish the rounding direction represented by its argument  
 31673 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding  
 31674 direction is not changed.

31675 **RETURN VALUE**

31676 The *fegetround()* function shall return the value of the rounding direction macro representing the  
 31677 current rounding direction or a negative value if there is no such rounding direction macro or  
 31678 the current rounding direction is not determinable.

31679 The *fesetround()* function shall return a zero value if and only if the requested rounding direction  
 31680 was established.

31681 **ERRORS**

31682 No errors are defined.

31683 **EXAMPLES**

31684 The following example saves, sets, and restores the rounding direction, reporting an error and  
 31685 aborting if setting the rounding direction fails:

```
31686 #include <fenv.h>
31687 #include <assert.h>
31688 void f(int round_dir)
31689 {
31690     #pragma STDC FENV_ACCESS ON
31691     int save_round;
31692     int setround_ok;
31693     save_round = fegetround();
31694     setround_ok = fesetround(round_dir);
31695     assert(setround_ok == 0);
31696     /* ... */
31697     fesetround(save_round);
31698     /* ... */
31699 }
```

31700 **APPLICATION USAGE**

31701 None.

31702 **RATIONALE**

31703 None.

31704 **FUTURE DIRECTIONS**

31705 None.

31706 **SEE ALSO**

31707 XBD [<fenv.h>](#)

31708 **CHANGE HISTORY**

31709 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31710 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.



31711 **NAME**

31712 feholdexcept — save current floating-point environment

31713 **SYNOPSIS**

31714 #include &lt;fenv.h&gt;

31715 int feholdexcept(fenv\_t \*envp);

31716 **DESCRIPTION**

31717 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31718 conflict between the requirements described here and the ISO C standard is unintentional. This  
31719 volume of POSIX.1-2024 defers to the ISO C standard.

31720 The *feholdexcept()* function shall save the current floating-point environment in the object  
31721 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on  
31722 floating-point exceptions) mode, if available, for all floating-point exceptions.

31723 **RETURN VALUE**

31724 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception  
31725 handling was successfully installed.

31726 **ERRORS**

31727 No errors are defined.

31728 **EXAMPLES**

31729 None.

31730 **APPLICATION USAGE**

31731 None.

31732 **RATIONALE**

31733 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard  
31734 implementations which have the default non-stop mode and at least one other mode for trap  
31735 handling or aborting. If the implementation provides only the non-stop mode, then installing the  
31736 non-stop mode is trivial.

31737 **FUTURE DIRECTIONS**

31738 None.

31739 **SEE ALSO**31740 *fegetenv()*, *feupdateenv()*

31741 XBD &lt;fenv.h&gt;

31742 **CHANGE HISTORY**

31743 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31744 **NAME**

31745 feof — test end-of-file indicator on a stream

31746 **SYNOPSIS**

31747 #include &lt;stdio.h&gt;

31748 int feof(FILE \*stream);

31749 **DESCRIPTION**

31750 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31751 conflict between the requirements described here and the ISO C standard is unintentional. This  
31752 volume of POSIX.1-2024 defers to the ISO C standard.

31753 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.31754 CX The *feof()* function shall not change the setting of *errno* if *stream* is valid.31755 **RETURN VALUE**31756 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.31757 **ERRORS**

31758 No errors are defined.

31759 **EXAMPLES**

31760 None.

31761 **APPLICATION USAGE**

31762 None.

31763 **RATIONALE**

31764 None.

31765 **FUTURE DIRECTIONS**

31766 None.

31767 **SEE ALSO**31768 [\*clearerr\(\)\*](#), [\*ferror\(\)\*](#), [\*fopen\(\)\*](#)31769 XBD [\*\*<stdio.h>\*\*](#)31770 **CHANGE HISTORY**

31771 First released in Issue 1. Derived from Issue 1 of the SVID.

31772 **Issue 7**

31773 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0124 [401] is applied.

31774 **NAME**

31775           feraiseexcept — raise floating-point exception

31776 **SYNOPSIS**

31777           #include &lt;fenv.h&gt;

31778           int feraiseexcept(int *excepts*);31779 **DESCRIPTION**

31780 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
31781       conflict between the requirements described here and the ISO C standard is unintentional. This  
31782       volume of POSIX.1-2024 defers to the ISO C standard.

31783       The *feraiseexcept()* function shall attempt to raise the supported floating-point exceptions  
31784       represented by the *excepts* argument. The order in which these floating-point exceptions are  
31785 MX       raised is unspecified, except that if the *excepts* argument represents IEC 60559 valid coincident  
31786       floating-point exceptions for atomic operations (namely overflow and inexact, or underflow and  
31787       inexact), then overflow or underflow shall be raised before inexact. Whether the *feraiseexcept()*  
31788       function additionally raises the inexact floating-point exception whenever it raises the overflow  
31789       or underflow floating-point exception is implementation-defined.

31790 **RETURN VALUE**

31791       If the argument is zero or if all the specified exceptions were successfully raised, *feraiseexcept()*  
31792       shall return zero. Otherwise, it shall return a non-zero value.

31793 **ERRORS**

31794       No errors are defined.

31795 **EXAMPLES**

31796       None.

31797 **APPLICATION USAGE**

31798       The effect is intended to be similar to that of floating-point exceptions raised by arithmetic  
31799       operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

31800 **RATIONALE**

31801       Raising overflow or underflow is allowed to also raise inexact because on some architectures the  
31802       only practical way to raise an exception is to execute an instruction that has the exception as a  
31803       side-effect. The function is not restricted to accept only valid coincident expressions for atomic  
31804       operations, so the function can be used to raise exceptions accrued over several operations.

31805 **FUTURE DIRECTIONS**

31806       None.

31807 **SEE ALSO**31808       [\*feclearexcept\(\)\*](#), [\*feketexceptflag\(\)\*](#), [\*fetestexcept\(\)\*](#)31809       XBD <[\*\*fenv.h\*\*](#)>31810 **CHANGE HISTORY**

31811       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31812       ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

31813 **Issue 7**

31814       POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0111 [543] is applied.

31815 **NAME**

31816 error — test error indicator on a stream

31817 **SYNOPSIS**

31818 #include &lt;stdio.h&gt;

31819 int ferror(FILE \*stream);

31820 **DESCRIPTION**31821 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31822 conflict between the requirements described here and the ISO C standard is unintentional. This  
31823 volume of POSIX.1-2024 defers to the ISO C standard.31824 The *ferror()* function shall test the error indicator for the stream pointed to by *stream*.31825 CX The *ferror()* function shall not change the setting of *errno* if *stream* is valid.31826 **RETURN VALUE**31827 The *ferror()* function shall return non-zero if and only if the error indicator is set for *stream*.31828 **ERRORS**

31829 No errors are defined.

31830 **EXAMPLES**

31831 None.

31832 **APPLICATION USAGE**

31833 None.

31834 **RATIONALE**

31835 None.

31836 **FUTURE DIRECTIONS**

31837 None.

31838 **SEE ALSO**31839 *clearerr()*, *feof()*, *fopen()*

31840 XBD &lt;stdio.h&gt;

31841 **CHANGE HISTORY**

31842 First released in Issue 1. Derived from Issue 1 of the SVID.

31843 **Issue 7**

31844 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0125 [401] is applied.

31845 **NAME**

31846       fesetenv — set current floating-point environment

31847 **SYNOPSIS**

31848       #include &lt;fenv.h&gt;

31849       int fesetenv(const fenv\_t \*envp);

31850 **DESCRIPTION**31851       Refer to *fegetenv()*.

31852 **NAME**

31853       fesetexceptflag — set floating-point status flags

31854 **SYNOPSIS**

31855       #include <fenv.h>

31856       int fesetexceptflag(const fexcept\_t \*flagp, int excepts);

31857 **DESCRIPTION**

31858       Refer to *fegetexceptflag()*.

31859 **NAME**

31860           fesetround — set current rounding direction

31861 **SYNOPSIS**

31862           #include <fenv.h>

31863           int fesetround(int *round*);

31864 **DESCRIPTION**

31865           Refer to *fegetround()*.

31866 **NAME**

31867 fetestexcept — test floating-point exception flags

31868 **SYNOPSIS**

```
31869 #include <fenv.h>
31870 int fetestexcept(int excepts);
```

31871 **DESCRIPTION**

31872 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 31873 conflict between the requirements described here and the ISO C standard is unintentional. This  
 31874 volume of POSIX.1-2024 defers to the ISO C standard.

31875 The *fetestexcept()* function shall determine which of a specified subset of the floating-point  
 31876 exception flags are currently set. The *excepts* argument specifies the floating-point status flags to  
 31877 be queried.

31878 **RETURN VALUE**

31879 The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point  
 31880 exception macros corresponding to the currently set floating-point exceptions included in  
 31881 *excepts*.

31882 **ERRORS**

31883 No errors are defined.

31884 **EXAMPLES**

31885 The following example calls function *f()* if an invalid exception is set, and then function *g()* if an  
 31886 overflow exception is set:

```
31887 #include <fenv.h>
31888 /* ... */
31889 {
31890     #pragma STDC FENV_ACCESS ON
31891     int set_excepts;
31892     feclearexcept (FE_INVALID | FE_OVERFLOW);
31893     // maybe raise exceptions
31894     set_excepts = fetestexcept (FE_INVALID | FE_OVERFLOW);
31895     if (set_excepts & FE_INVALID) f();
31896     if (set_excepts & FE_OVERFLOW) g();
31897     /* ... */
31898 }
```

31899 **APPLICATION USAGE**

31900 None.

31901 **RATIONALE**

31902 None.

31903 **FUTURE DIRECTIONS**

31904 None.

31905 **SEE ALSO**31906 [feclearexcept\(\)](#), [fegetexceptflag\(\)](#), [feraiseexcept\(\)](#)31907 XBD [<fenv.h>](#)



31908 **CHANGE HISTORY**

31909 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

31910 **NAME**

31911 feupdateenv — update floating-point environment

31912 **SYNOPSIS**

31913 #include &lt;fenv.h&gt;

31914 int feupdateenv(const fenv\_t \*envp);

31915 **DESCRIPTION**

31916 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31917 conflict between the requirements described here and the ISO C standard is unintentional. This  
31918 volume of POSIX.1-2024 defers to the ISO C standard.

31919 The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in  
31920 its automatic storage, attempt to install the floating-point environment represented by the object  
31921 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument  
31922 *envp* shall point to an object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point  
31923 environment macro.

31924 **RETURN VALUE**

31925 The *feupdateenv()* function shall return a zero value if and only if all the required actions were  
31926 successfully carried out.

31927 **ERRORS**

31928 No errors are defined.

31929 **EXAMPLES**

31930 The following example shows sample code to hide spurious underflow floating-point  
31931 exceptions:

```
31932 #include <fenv.h>
31933 double f(double x)
31934 {
31935     #pragma STDC FENV_ACCESS ON
31936     double result;
31937     fenv_t save_env;
31938     feholdexcept(&save_env);
31939     // compute result
31940     if (/* test spurious underflow */)
31941         feclearexcept(FE_UNDERFLOW);
31942     feupdateenv(&save_env);
31943     return result;
31944 }
```

31945 **APPLICATION USAGE**

31946 None.

31947 **RATIONALE**

31948 None.

31949 **FUTURE DIRECTIONS**

31950 None.

31951 **SEE ALSO**31952 *fegetenv()*, *feholdexcept()*

31953 XBD &lt;fenv.h&gt;

31954 **CHANGE HISTORY**

- 31955 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.
- 31956 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

31957 **NAME**

31958           fexecve — execute a file

31959 **SYNOPSIS**

31960           #include &lt;unistd.h&gt;

31961           int fexecve(int *fd*, char \*const *argv*[], char \*const *envp*[]);31962 **DESCRIPTION**31963           Refer to *exec*.

31964 **NAME**

31965 fflush — flush a stream

31966 **SYNOPSIS**

31967 #include &lt;stdio.h&gt;

31968 int fflush(FILE \*stream);

31969 **DESCRIPTION**

31970 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 31971 conflict between the requirements described here and the ISO C standard is unintentional. This  
 31972 volume of POSIX.1-2024 defers to the ISO C standard.

31973 If *stream* points to an output stream or an update stream in which the most recent operation was  
 31974 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the  
 31975 last data modification and last file status change timestamps of the underlying file shall be  
 31976 marked for update.

31977 For a stream open for reading with an underlying file description, if the file is not already at  
 31978 EOF, and the file is one capable of seeking, the file offset of the underlying open file description  
 31979 shall be set to the file position of the stream, and any characters pushed back onto the stream by  
 31980 *ungetc()* or *ungetwc()* that have not subsequently been read from the stream shall be discarded  
 31981 (without further changing the file offset).

31982 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the  
 31983 behavior is defined above.

31984 **RETURN VALUE**

31985 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for  
 31986 CX the stream, return EOF, and set *errno* to indicate the error.

31987 **ERRORS**31988 The *fflush()* function shall fail if:

31989 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and  
 31990 the thread would be delayed in the write operation.

31991 CX [EBADF] The file descriptor underlying *stream* is not valid.

31992 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

31993 CX [EFBIG] An attempt was made to write a file that exceeds the file size limit of the  
 31994 process.

31995 XSI A SIGXFSZ signal shall also be generated for the thread.

31996 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 31997 offset maximum associated with the corresponding stream.

31998 CX [EINTR] The *fflush()* function was interrupted by a signal.

31999 CX [EIO] The process is a member of a background process group attempting to write to  
 32000 its controlling terminal, TOSTOP is set, the calling thread is not blocking  
 32001 SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the  
 32002 process is orphaned. This error may also be returned under implementation-  
 32003 defined conditions.

32004 CX [ENOMEM] The underlying stream was created by *open\_memstream()* or  
 32005 *open\_wmemstream()* and insufficient memory is available.

32006 CX [ENOSPC] There was no free space remaining on the device containing the file or in the  
32007 buffer used by the *fmemopen()* function.

32008 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
32009 any process. A SIGPIPE signal shall also be sent to the thread.

32010 The *fflush()* function may fail if:

32011 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
32012 capabilities of the device.

## 32013 EXAMPLES

### 32014 Sending Prompts to Standard Output

32015 The following example uses *printf()* calls to print a series of prompts for information the user  
32016 must enter from standard input. The *fflush()* calls force the output to standard output. The  
32017 *fflush()* function is used because standard output is usually buffered and the prompt may not  
32018 immediately be printed on the output or terminal. The *getline()* function calls read strings from  
32019 standard input and place the results in variables, for use later in the program.

```
32020 char *user;
32021 char *oldpasswd;
32022 char *newpasswd;
32023 ssize_t llen;
32024 size_t blen;
32025 struct termios term;
32026 tcflag_t saveflag;

32027 printf("User name: ");
32028 fflush(stdout);
32029 blen = 0;
32030 llen = getline(&user, &blen, stdin);
32031 user[llen-1] = 0;
32032 tcgetattr(fileno(stdin), &term);
32033 saveflag = term.c_lflag;
32034 term.c_lflag &= ~ECHO;
32035 tcsetattr(fileno(stdin), TCSANOW, &term);
32036 printf("Old password: ");
32037 fflush(stdout);
32038 blen = 0;
32039 llen = getline(&oldpasswd, &blen, stdin);
32040 oldpasswd[llen-1] = 0;

32041 printf("\nNew password: ");
32042 fflush(stdout);
32043 blen = 0;
32044 llen = getline(&newpasswd, &blen, stdin);
32045 newpasswd[llen-1] = 0;
32046 term.c_lflag = saveflag;
32047 tcsetattr(fileno(stdin), TCSANOW, &term);
32048 free(user);
32049 free(oldpasswd);
32050 free(newpasswd);
```

32051 **APPLICATION USAGE**

32052 None.

32053 **RATIONALE**

32054 Data buffered by the system may make determining the validity of the position of the current  
 32055 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*  
 32056 on streams open for *read()* is not mandated by POSIX.1-2024.

32057 **FUTURE DIRECTIONS**

32058 None.

32059 **SEE ALSO**32060 [Section 2.5](#) (on page 521), *fmemopen()*, *getrlimit()*, *open\_memstream()*32061 XBD [<stdio.h>](#)32062 **CHANGE HISTORY**

32063 First released in Issue 1. Derived from Issue 1 of the SVID.

32064 **Issue 5**

32065 Large File Summit extensions are added.

32066 **Issue 6**

32067 Extensions beyond the ISO C standard are marked.

32068 The following new requirements on POSIX implementations derive from alignment with the  
 32069 Single UNIX Specification:

- 32070 • The [EFBIG] error is added as part of the large file support extensions.
- 32071 • The [ENXIO] optional error condition is added.

32072 The RETURN VALUE section is updated to note that the error indicator shall be set for the  
 32073 stream. This is for alignment with the ISO/IEC 9899:1999 standard.

32074 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/31 is applied, updating the [EAGAIN]  
 32075 error in the ERRORS section from “the process would be delayed” to “the thread would be  
 32076 delayed”.

32077 **Issue 7**

32078 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file  
 32079 descriptors and streams.

32080 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open  
 32081 Group Technical Standard, 2006, Extended API Set Part 1.

32082 The EXAMPLES section is revised.

32083 Changes are made related to support for finegrained timestamps.

32084 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0126 [87], XSH/TC1-2008/0127 [79],  
 32085 and XSH/TC1-2008/0128 [14] are applied.

32086 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0112 [816] and XSH/TC2-2008/0113  
 32087 [626] are applied.

32088 **Issue 8**

32089 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

32090  
32091

Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error relating to the file size limit for the process.



32092 **NAME**

32093           ffs, ffs1, ffs11 — find first set bit

32094 **SYNOPSIS**

```
32095 XSI       #include <strings.h>
32096           int ffs(int i);
32097           int ffs1(long i);
32098           int ffs11(long long i);
```

32099 **DESCRIPTION**

32100       The *ffs()*, *ffs1()*, and *ffs11()* functions shall find the first bit set (beginning with the least significant bit) in *i*, and return the index of that bit. Bits are numbered starting at one (the least significant bit).

32103 **RETURN VALUE**

32104       The *ffs()*, *ffs1()*, and *ffs11()* functions shall return the index of the first bit set. If *i* is 0, then these functions shall return 0.

32106 **ERRORS**

32107       No errors are defined.

32108 **EXAMPLES**

32109       None.

32110 **APPLICATION USAGE**

32111       None.

32112 **RATIONALE**

32113       None.

32114 **FUTURE DIRECTIONS**

32115       None.

32116 **SEE ALSO**32117       XBD <[strings.h](#)>32118 **CHANGE HISTORY**

32119       First released in Issue 4, Version 2.

32120 **Issue 5**

32121       Moved from X/OPEN UNIX extension to BASE.

32122 **Issue 8**32123       Austin Group Defect 617 is applied, adding *ffs1()* and *ffs11()*.

32124 **NAME**

32125 fgetc — get a byte from a stream

32126 **SYNOPSIS**

32127 #include &lt;stdio.h&gt;

32128 int fgetc(FILE \*stream);

32129 **DESCRIPTION**

32130 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32131 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32132 volume of POSIX.1-2024 defers to the ISO C standard.

32133 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is  
 32134 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,  
 32135 from the input stream pointed to by *stream*, and advance the associated file position indicator for  
 32136 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple  
 32137 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.

32138 CX The *fgetc()* function may mark the last data access timestamp of the file associated with *stream*  
 32139 for update. The last data access timestamp shall be marked for update by the first successful  
 32140 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, or *scanf()* using  
 32141 *stream* that returns data not supplied by a prior call to *ungetc()*.

32142 **RETURN VALUE**

32143 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to  
 32144 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the  
 32145 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If an error occurs, the  
 32146 CX error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to indicate  
 32147 the error.

32148 **ERRORS**32149 The *fgetc()* function shall fail if data needs to be read and:

32150 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and  
 32151 the thread would be delayed in the *fgetc()* operation.

32152 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for  
 32153 reading.

32154 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
 32155 was transferred.

32156 CX [EIO] A physical I/O error has occurred, or the process is in a background process  
 32157 group attempting to read from its controlling terminal, and either the calling  
 32158 thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process  
 32159 group of the process is orphaned. This error may also be generated for  
 32160 implementation-defined reasons.

32161 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the  
 32162 offset maximum associated with the corresponding stream.

32163 The *fgetc()* function may fail if:

32164 CX [ENOMEM] Insufficient storage space is available.

32165 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
 32166 capabilities of the device.

32167 **EXAMPLES**

32168 None.

32169 **APPLICATION USAGE**

32170 If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared  
 32171 against the integer constant EOF, the comparison may never succeed, because sign-extension of  
 32172 a variable of type **char** on widening to integer is implementation-defined.

32173 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
 32174 end-of-file condition.

32175 **RATIONALE**

32176 None.

32177 **FUTURE DIRECTIONS**

32178 None.

32179 **SEE ALSO**32180 [Section 2.5](#) (on page 521), *feof()*, *ferror()*, *fgets()*, *fread()*, *fscanf()*, *getchar()*, *getc()*, *ungetc()*32181 XBD [<stdio.h>](#)32182 **CHANGE HISTORY**

32183 First released in Issue 1. Derived from Issue 1 of the SVID.

32184 **Issue 5**

32185 Large File Summit extensions are added.

32186 **Issue 6**

32187 Extensions beyond the ISO C standard are marked.

32188 The following new requirements on POSIX implementations derive from alignment with the  
 32189 Single UNIX Specification:

- 32190 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 32191 • The [ENOMEM] and [ENXIO] optional error conditions are added.

32192 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 32193 • The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the  
 32194 input stream is not set.
- 32195 • The RETURN VALUE section is updated to note that the error indicator shall be set for the  
 32196 stream.

32197 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/32 is applied, updating the [EAGAIN]  
 32198 error in the ERRORS section from “the process would be delayed” to “the thread would be  
 32199 delayed”.

32200 **Issue 7**

32201 Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark  
 32202 the last data access timestamp for update.

32203 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0129 [79] and XSH/TC1-2008/0130  
 32204 [14] are applied.

32205 **Issue 8**

32206 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

32207 Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

32208 **NAME**

32209 fgetpos — get current file position information

32210 **SYNOPSIS**

32211 #include &lt;stdio.h&gt;

32212 int fgetpos(FILE \*restrict stream, fpos\_t \*restrict pos);

32213 **DESCRIPTION**

32214 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32215 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32216 volume of POSIX.1-2024 defers to the ISO C standard.

32217 The *fgetpos()* function shall store the current values of the parse state (if any) and file position  
 32218 indicator for the stream pointed to by *stream* in the object pointed to by *pos*. The value stored  
 32219 contains unspecified information usable by *fsetpos()* for repositioning the stream to its position  
 32220 at the time of the call to *fgetpos()*.

32221 The *fgetpos()* function shall not change the setting of *errno* if successful.

32222 **RETURN VALUE**

32223 Upon successful completion, *fgetpos()* shall return 0; otherwise, it shall return a non-zero value  
 32224 and set *errno* to indicate the error.

32225 **ERRORS**

32226 The *fgetpos()* function shall fail if:

32227 CX [EBADF] The file descriptor underlying *stream* is not valid.

32228 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an  
 32229 object of type **fpos\_t**.

32230 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

32231 **EXAMPLES**

32232 None.

32233 **APPLICATION USAGE**

32234 None.

32235 **RATIONALE**

32236 None.

32237 **FUTURE DIRECTIONS**

32238 None.

32239 **SEE ALSO**

32240 [Section 2.5](#) (on page 521), *fopen()*, *ftell()*, *rewind()*, *ungetc()*

32241 XBD [<stdio.h>](#)

32242 **CHANGE HISTORY**

32243 First released in Issue 4. Derived from the ISO C standard.

32244 **Issue 5**

32245 Large File Summit extensions are added.

32246 **Issue 6**

32247 Extensions beyond the ISO C standard are marked.

32248 The following new requirements on POSIX implementations derive from alignment with the  
32249 Single UNIX Specification:

32250 • The [EBADF] and [ESPIPE] optional error conditions are added.

32251 An additional [ESPIPE] error condition is added for sockets.

32252 The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

32253 **Issue 7**

32254 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0131 [105], XSH/TC1-2008/0132 [122],  
32255 and XSH/TC1-2008/0133 [14] are applied.

32256 **NAME**

32257 fgets — get a string from a stream

32258 **SYNOPSIS**

32259 #include &lt;stdio.h&gt;

32260 char \*fgets(char \*restrict s, int n, FILE \*restrict stream);

32261 **DESCRIPTION**

32262 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32263 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32264 volume of POSIX.1-2024 defers to the ISO C standard.

32265 The `fgets()` function shall read bytes from *stream* into the array pointed to by *s* until *n*−1 bytes are  
 32266 read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered. A  
 32267 null byte shall be written immediately after the last byte read into the array. If the end-of-file  
 32268 condition is encountered before any bytes are read, the contents of the array pointed to by *s* shall  
 32269 not be changed.

32270 CX The `fgets()` function may mark the last data access timestamp of the file associated with *stream*  
 32271 for update. The last data access timestamp shall be marked for update by the first successful  
 32272 execution of `fgetc()`, `fgets()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `getdelim()`, `getline()`, or `scanf()` using  
 32273 *stream* that returns data not supplied by a prior call to `ungetc()`.

32274 **RETURN VALUE**

32275 Upon successful completion, `fgets()` shall return *s*. If the stream is at end-of-file, the end-of-file  
 32276 indicator for the stream shall be set and `fgets()` shall return a null pointer. If an error occurs, the  
 32277 CX error indicator for the stream shall be set, `fgets()` shall return a null pointer, and shall set `errno` to  
 32278 indicate the error.

32279 **ERRORS**32280 Refer to `fgetc()`.32281 **EXAMPLES**32282 **Reading Input**

32283 The following example uses `fgets()` to read lines of input. It assumes that the file it is reading is a  
 32284 text file and that lines in this text file are no longer than 16384 (or `{LINE_MAX}` if it is less than  
 32285 16384 on the implementation where it is running) bytes long. (Note that the standard utilities  
 32286 have no line length limit if `sysconf(_SC_LINE_MAX)` returns −1 without setting `errno`. This  
 32287 example assumes that `sysconf(_SC_LINE_MAX)` will not fail.)

```

32288 #include <limits.h>
32289 #include <stdio.h>
32290 #include <unistd.h>
32291 #define MYLIMIT 16384

32292 char *line;
32293 int line_max;
32294 if (LINE_MAX >= MYLIMIT) {
32295     // Use maximum line size of MYLIMIT. If LINE_MAX is
32296     // bigger than our limit, sysconf() cannot report a
32297     // smaller limit.
32298     line_max = MYLIMIT;
32299 } else {
32300     long limit = sysconf(_SC_LINE_MAX);
32301     line_max = (limit < 0 || limit > MYLIMIT) ? MYLIMIT : (int)limit;

```

```

32302     }
32303     // line_max + 1 leaves room for the null byte added by fgets().
32304     line = malloc(line_max + 1);
32305     if (line == NULL) {
32306         // out of space
32307         ...
32308         return error;
32309     }
32310     while (fgets(line, line_max + 1, fp) != NULL) {
32311         // Verify that a full line has been read ...
32312         // If not, report an error or prepare to treat the
32313         // next time through the loop as a read of a
32314         // continuation of the current line.
32315         ...
32316         // Process line ...
32317         ...
32318     }
32319     free(line);
32320     ...

```

32321 **APPLICATION USAGE**  
32322 None.

32323 **RATIONALE**  
32324 None.

32325 **FUTURE DIRECTIONS**  
32326 None.

32327 **SEE ALSO**  
32328 [Section 2.5](#) (on page 521), [fgets\(\)](#), [fopen\(\)](#), [fread\(\)](#), [fscanf\(\)](#), [getc\(\)](#), [getchar\(\)](#), [getdelim\(\)](#), [ungetc\(\)](#)  
32329 XBD [<stdio.h>](#)

32330 **CHANGE HISTORY**  
32331 First released in Issue 1. Derived from Issue 1 of the SVID.

32332 **Issue 6**  
32333 Extensions beyond the ISO C standard are marked.  
32334 The prototype for `fgets()` is changed for alignment with the ISO/IEC 9899:1999 standard.

32335 **Issue 7**  
32336 Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark  
32337 the last data access timestamp for update.  
32338 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0134 [182] and XSH/TC1-2008/0135  
32339 [14] are applied.  
32340 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0114 [468] is applied.

32341 **Issue 8**  
32342 Austin Group Defect 1330 is applied, removing obsolescent interfaces.  
32343 Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

32344 **NAME**

32345 fgetwc — get a wide-character code from a stream

32346 **SYNOPSIS**

32347 #include &lt;stdio.h&gt;

32348 #include &lt;wchar.h&gt;

32349 wint\_t fgetwc(FILE \*stream);

32350 **DESCRIPTION**

32351 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32352 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32353 volume of POSIX.1-2024 defers to the ISO C standard.

32354 The *fgetwc()* function shall obtain the next character (if present) from the input stream pointed to  
 32355 by *stream*, convert that to the corresponding wide-character code, and advance the associated file  
 32356 position indicator for the stream (if defined).

32357 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

32358 CX The *fgetwc()* function may mark the last data access timestamp of the file associated with *stream*  
 32359 for update. The last data access timestamp shall be marked for update by the first successful  
 32360 execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *vfwscanf()*, *vwscanf()*, or *wscanf()*  
 32361 using *stream* that returns data not supplied by a prior call to *ungetwc()*.

32362 The *fgetwc()* function shall not change the setting of *errno* if successful.

32363 **RETURN VALUE**

32364 Upon successful completion, the *fgetwc()* function shall return the wide-character code of the  
 32365 character read from the input stream pointed to by *stream* converted to a type **wint\_t**. If the end-  
 32366 of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for  
 32367 the stream shall be set and *fgetwc()* shall return WEOF. If an error other than an encoding error  
 32368 CX occurs, the error indicator for the stream shall be set, and *fgetwc()* shall return WEOF and shall  
 32369 CX set *errno* to indicate the error. If an encoding error occurs, the error indicator for the stream shall  
 32370 be set, and *fgetwc()* shall return WEOF and shall set *errno* to indicate the error.

32371 **ERRORS**

32372 The *fgetwc()* function shall fail if data needs to be read and:

32373 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and  
 32374 the thread would be delayed in the *fgetwc()* operation.

32375 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for  
 32376 reading.

32377 [EILSEQ] The data obtained from the input stream does not form a valid character.

32378 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data  
 32379 was transferred.

32380 CX [EIO] A physical I/O error has occurred, or the process is in a background process  
 32381 group attempting to read from its controlling terminal, and either the calling  
 32382 thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process  
 32383 group of the process is orphaned. This error may also be generated for  
 32384 implementation-defined reasons.

32385 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the  
 32386 offset maximum associated with the corresponding stream.



32387 The *fgetwc()* function may fail if:

32388 CX [ENOMEM] Insufficient storage space is available.

32389 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
32390 capabilities of the device.

### 32391 EXAMPLES

32392 None.

### 32393 APPLICATION USAGE

32394 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
32395 end-of-file condition.

### 32396 RATIONALE

32397 The requirement to set the error indicator when an encoding error occurs is shaded CX because  
32398 this is not required by the ISO C standard. However, the next revision of the ISO C standard is  
32399 expected to add this requirement.

### 32400 FUTURE DIRECTIONS

32401 None.

### 32402 SEE ALSO

32403 [Section 2.5](#) (on page 521), *feof()*, *ferror()*, *fopen()*

32404 XBD [<stdio.h>](#), [<wchar.h>](#)

### 32405 CHANGE HISTORY

32406 First released in Issue 4. Derived from the MSE working draft.

#### 32407 Issue 5

32408 The Optional Header (OH) marking is removed from [<stdio.h>](#).

32409 Large File Summit extensions are added.

#### 32410 Issue 6

32411 Extensions beyond the ISO C standard are marked.

32412 The following new requirements on POSIX implementations derive from alignment with the  
32413 Single UNIX Specification:

- 32414 • The [EIO] and [Eoverflow] mandatory error conditions are added.
- 32415 • The [ENOMEM] and [ENXIO] optional error conditions are added.

32416 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/33 is applied, updating the [EAGAIN]  
32417 error in the ERRORS section from “the process would be delayed” to “the thread would be  
32418 delayed”.

#### 32419 Issue 7

32420 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

32421 Changes are made related to support for finegrained timestamps.

32422 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0136 [105], XSH/TC1-2008/0137 [79],  
32423 and XSH/TC1-2008/0138 [14] are applied.

#### 32424 Issue 8

32425 Austin Group Defect 1624 is applied, changing the RETURN VALUE and RATIONALE sections.

32426 **NAME**

32427 fgetws — get a wide-character string from a stream

32428 **SYNOPSIS**

32429 #include &lt;stdio.h&gt;

32430 #include &lt;wchar.h&gt;

32431 wchar\_t \*fgetws(wchar\_t \*restrict ws, int n,

32432 FILE \*restrict stream);

32433 **DESCRIPTION**

32434 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32435 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32436 volume of POSIX.1-2024 defers to the ISO C standard.

32437 The *fgetws()* function shall read characters from the *stream*, convert these to the corresponding  
 32438 wide-character codes, place them in the **wchar\_t** array pointed to by *ws*, until *n*−1 characters are  
 32439 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is  
 32440 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character  
 32441 code.

32442 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

32443 CX The *fgetws()* function may mark the last data access timestamp of the file associated with *stream*  
 32444 for update. The last data access timestamp shall be marked for update by the first successful  
 32445 execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *vfwscanf()*, *vwsscanf()*, or *wscanf()*  
 32446 using *stream* that returns data not supplied by a prior call to *ungetwc()*.

32447 **RETURN VALUE**

32448 Upon successful completion, *fgetws()* shall return *ws*. If the end-of-file indicator for the stream is  
 32449 set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and  
 32450 *fgetws()* shall return a null pointer. If an error occurs, the error indicator for the stream shall be  
 32451 CX set, and *fgetws()* shall return a null pointer and shall set *errno* to indicate the error.

32452 **ERRORS**32453 Refer to *fgetwc()*.32454 **EXAMPLES**

32455 None.

32456 **APPLICATION USAGE**

32457 None.

32458 **RATIONALE**

32459 None.

32460 **FUTURE DIRECTIONS**

32461 None.

32462 **SEE ALSO**32463 [Section 2.5](#) (on page 521), *fopen()*, *fread()*32464 XBD [<stdio.h>](#), [<wchar.h>](#)32465 **CHANGE HISTORY**

32466 First released in Issue 4. Derived from the MSE working draft.

- 32467 **Issue 5**  
32468 The Optional Header (OH) marking is removed from `<stdio.h>`.
- 32469 **Issue 6**  
32470 Extensions beyond the ISO C standard are marked.  
32471 The prototype for `fgetws()` is changed for alignment with the ISO/IEC 9899:1999 standard.
- 32472 **Issue 7**  
32473 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.  
32474 Changes are made related to support for finegrained timestamps.  
32475 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0139 [14] is applied.
- 32476 **Issue 8**  
32477 Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

32478 **NAME**

32479 fileno — map a stream pointer to a file descriptor

32480 **SYNOPSIS**

```
32481 CX #include <stdio.h>
32482 int fileno(FILE *stream);
```

32483 **DESCRIPTION**

32484 The *fileno()* function shall return the integer file descriptor associated with the stream pointed to  
 32485 by *stream*.

32486 **RETURN VALUE**

32487 Upon successful completion, *fileno()* shall return the integer value of the file descriptor  
 32488 associated with *stream*. Otherwise, the value  $-1$  shall be returned and *errno* set to indicate the  
 32489 error.

32490 **ERRORS**

32491 The *fileno()* function shall fail if:

32492 [EBADF] The stream is not associated with a file.

32493 The *fileno()* function may fail if:

32494 [EBADF] The file descriptor underlying *stream* is not a valid file descriptor.

32495 **EXAMPLES**

32496 None.

32497 **APPLICATION USAGE**

32498 None.

32499 **RATIONALE**

32500 Without some specification of which file descriptors are associated with these streams, it is  
 32501 impossible for an application to set up the streams for another application it starts with *fork()*  
 32502 and *exec*. In particular, it would not be possible to write a portable version of the *sh* command  
 32503 interpreter (although there may be other constraints that would prevent that portability).

32504 **FUTURE DIRECTIONS**

32505 None.

32506 **SEE ALSO**

32507 [Section 2.5.1](#) (on page 522), *dirfd()*, *fdopen()*, *fopen()*, *stdin*

32508 XBD [<stdio.h>](#)

32509 **CHANGE HISTORY**

32510 First released in Issue 1. Derived from Issue 1 of the SVID.

32511 **Issue 6**

32512 The following new requirements on POSIX implementations derive from alignment with the  
 32513 Single UNIX Specification:

- 32514 • The [EBADF] optional error condition is added.

32515 **Issue 7**

32516 SD5-XBD-ERN-99 is applied, changing the definition of the [EBADF] error.

32517 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0115 [589] is applied.

32518 **NAME**

32519 flockfile, ftrylockfile, funlockfile — stdio locking functions

32520 **SYNOPSIS**

```
32521 CX #include <stdio.h>
32522 void flockfile(FILE *file);
32523 int ftrylockfile(FILE *file);
32524 void funlockfile(FILE *file);
```

32525 **DESCRIPTION**

32526 These functions shall provide for explicit application-level locking of the locks associated with  
 32527 standard I/O streams (see [Section 2.5](#), on page 521). These functions can be used by a thread to  
 32528 delineate a sequence of I/O statements that are executed as a unit.

32529 The *flockfile()* function shall acquire for a thread ownership of a (**FILE \***) object.

32530 The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE \***) object if the object is  
 32531 available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

32532 The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is  
 32533 undefined if a thread other than the current owner calls the *funlockfile()* function.

32534 The functions shall behave as if there is a lock count associated with each (**FILE \***) object. This  
 32535 count is implicitly initialized to zero when the (**FILE \***) object is created. The (**FILE \***) object is  
 32536 unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE \***)  
 32537 object. When the *flockfile()* function is called, if the count is zero or if the count is positive and  
 32538 the caller owns the (**FILE \***) object, the count shall be incremented. Otherwise, the calling thread  
 32539 shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall  
 32540 decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*)  
 32541 and *funlockfile()* to be nested.

32542 **RETURN VALUE**

32543 None for *flockfile()* and *funlockfile()*.

32544 The *ftrylockfile()* function shall return zero for success and non-zero to indicate that the lock  
 32545 cannot be acquired.

32546 **ERRORS**

32547 No errors are defined.

32548 **EXAMPLES**

32549 None.

32550 **APPLICATION USAGE**

32551 Applications using these functions may be subject to priority inversion, as discussed in XBD  
 32552 [Section 3.275](#) (on page 72).

32553 A call to *exit()* can block until locked streams are unlocked because a thread having ownership  
 32554 of a (**FILE\***) object blocks all function calls that reference that (**FILE\***) object (except those with  
 32555 names ending in *\_unlocked*) from other threads, including calls to *exit()*.

32556 Note: a **FILE** lock is not a file lock (see XBD [Section 3.143](#), on page 51).

32557 **RATIONALE**

32558 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual-exclusion lock for each  
 32559 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a **FILE** lock,  
 32560 analogous to *pthread\_mutex\_trylock()*.

32561 These locks behave as if they are the same as those used internally by *stdio* for thread-safety.

32562 This both provides thread-safety of these functions without requiring a second level of internal  
32563 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

32564 Application developers and implementors should be aware that there are potential deadlock  
32565 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested  
32566 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of  
32567 implementation-defined line-buffered output streams to be flushed. If two threads each hold the  
32568 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring  
32569 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can  
32570 typically be avoided by acquiring locks on input streams before locks on output streams if a  
32571 thread would be acquiring both.

32572 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*  
32573 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where  
32574 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the  
32575 `*_unlocked` functions/macros. This moves the cost/performance tradeoff to the optimal point.

#### 32576 **FUTURE DIRECTIONS**

32577 None.

#### 32578 **SEE ALSO**

32579 [\*exit\(\)\*](#), [\*getc\\_unlocked\(\)\*](#)

32580 XBD Section 3.275 (on page 72), [`<stdio.h>`](#)

#### 32581 **CHANGE HISTORY**

32582 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 32583 **Issue 6**

32584 These functions are marked as part of the Thread-Safe Functions option.

#### 32585 **Issue 7**

32586 The *flockfile()*, *ftrylockfile()*, and *funlockfile()* functions are moved from the Thread-Safe Functions  
32587 option to the Base.

32588 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0140 [118] is applied.

32589 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0116 [611] is applied.

#### 32590 **Issue 8**

32591 Austin Group Defect 1118 is applied, clarifying that a **FILE** lock is not a file lock.

32592 Austin Group Defect 1302 is applied, replacing parts of the text with a reference to [Section 2.5](#)  
32593 (on page 521).

32594 **NAME**

32595 floor, floorf, floorl — floor function

32596 **SYNOPSIS**

```
32597 #include <math.h>
32598 double floor(double x);
32599 float floorf(float x);
32600 long double floorl(long double x);
```

32601 **DESCRIPTION**

32602 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 32603 conflict between the requirements described here and the ISO C standard is unintentional. This  
 32604 volume of POSIX.1-2024 defers to the ISO C standard.

32605 These functions shall compute the largest integral value not greater than  $x$ .

32606 MX These functions may raise the inexact floating-point exception for finite non-integer arguments.

32607 **RETURN VALUE**

32608 MX The returned value shall be independent of the current rounding direction mode and shall have  
 32609 the same sign as  $x$ .

32610 Upon successful completion, these functions shall return the largest integral value not greater  
 32611 than  $x$ , expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the  
 32612 function.

32613 MX If  $x$  is NaN, a NaN shall be returned.

32614 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.

32615 **ERRORS**

32616 No errors are defined.

32617 **EXAMPLES**

32618 None.

32619 **APPLICATION USAGE**

32620 The integral value returned by these functions might not be expressible as an **intmax\_t**. The  
 32621 return value should be tested before assigning it to an integer type to avoid the undefined  
 32622 results of an integer overflow.

32623 **RATIONALE**

32624 None.

32625 **FUTURE DIRECTIONS**

32626 None.

32627 **SEE ALSO**

32628 [ceil\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

32629 [Section 4.23](#) (on page 109), [<math.h>](#)

32630 **CHANGE HISTORY**

32631 First released in Issue 1. Derived from Issue 1 of the SVID.

32632 **Issue 5**

32633 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 32634 text was previously published in the APPLICATION USAGE section.

32635 **Issue 6**32636 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.32637 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
32638 revised to align with the ISO/IEC 9899:1999 standard.32639 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
32640 marked.32641 **Issue 7**

32642 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0141 [346] is applied.

32643 **Issue 8**32644 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
32645 standard.



32646 **NAME**32647 `fma, fmaf, fmal` — floating-point multiply-add32648 **SYNOPSIS**32649 `#include <math.h>`32650 `double fma(double x, double y, double z);`32651 `float fmaf(float x, float y, float z);`32652 `long double fmal(long double x, long double y, long double z);`32653 **DESCRIPTION**32654 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
32655 conflict between the requirements described here and the ISO C standard is unintentional. This  
32656 volume of POSIX.1-2024 defers to the ISO C standard.32657 These functions shall compute  $(x * y) + z$ , rounded as one ternary operation: they shall compute  
32658 the value (as if) to infinite precision and round once to the result format, according to the  
32659 rounding mode characterized by the value of `FLT_ROUNDS`.32660 An application wishing to check for error situations should set `errno` to zero and call  
32661 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
32662 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
32663 zero, an error has occurred.32664 **RETURN VALUE**32665 Upon successful completion, these functions shall return  $(x * y) + z$ , rounded as one ternary  
32666 operation.32667 **MX** If the result overflows or underflows, a range error may occur. On systems that support the IEC  
32668 60559 Floating-Point option, if the result overflows a range error shall occur.32669 If  $x$  or  $y$  are NaN, a NaN shall be returned.32670 If  $x$  multiplied by  $y$  is an exact infinity and  $z$  is also an infinity but with the opposite sign, a  
32671 domain error shall occur, and either a NaN (if supported), or an implementation-defined value  
32672 shall be returned.32673 If one of  $x$  and  $y$  is infinite, the other is zero, and  $z$  is not a NaN, a domain error shall occur, and  
32674 either a NaN (if supported), or an implementation-defined value shall be returned.32675 If one of  $x$  and  $y$  is infinite, the other is zero, and  $z$  is a NaN, a NaN shall be returned and a  
32676 domain error may occur.32677 If  $x*y$  is not  $0*\text{Inf}$  nor  $\text{Inf}*0$  and  $z$  is a NaN, a NaN shall be returned.32678 **ERRORS**

32679 These functions shall fail if:

32680 **MX** **Domain Error** The value of  $x*y+z$  is invalid, or the value  $x*y$  is invalid and  $z$  is not a NaN.32681 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
32682 then `errno` shall be set to [EDOM]. If the integer expression `(math_errhandling`  
32683 `& MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception  
32684 shall be raised.32685 **MX** **Range Error** The result overflows.32686 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
32687 then `errno` shall be set to [ERANGE]. If the integer expression  
32688 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the overflow  
32689 floating-point exception shall be raised.

32690 These functions may fail if:

32691	MX	<b>Domain Error</b>	The value $x*y$ is invalid and $z$ is a NaN.
32692			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
32693			then <i>errno</i> shall be set to [EDOM]. If the integer expression ( <i>math_errhandling</i>
32694			& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
32695			shall be raised.
32696		<b>Range Error</b>	The result underflows.
32697			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
32698			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
32699			( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
32700			floating-point exception shall be raised.
32701		<b>Range Error</b>	The result overflows.
32702			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
32703			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
32704			( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
32705			floating-point exception shall be raised.

#### 32706 EXAMPLES

32707 None.

#### 32708 APPLICATION USAGE

32709 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
32710 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

#### 32711 RATIONALE

32712 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its  
32713 unexpected use by the compiler can undermine carefully written code. The FP\_CONTRACT  
32714 macro can be used to disallow use of floating multiply-add; and the *fma()* function guarantees  
32715 its use where desired. Many current machines provide hardware floating multiply-add  
32716 instructions; software implementation can be used for others.

#### 32717 FUTURE DIRECTIONS

32718 None.

#### 32719 SEE ALSO

32720 [\*feclearexcept\(\)\*](#), [\*fetestexcept\(\)\*](#)

32721 XBD Section 4.23 (on page 109), [<math.h>](#)

#### 32722 CHANGE HISTORY

32723 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

#### 32724 Issue 7

32725 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied,  
32726 adding a “may fail” range error for non-MX systems.

32727 **NAME**

32728 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

32729 **SYNOPSIS**

32730 #include &lt;math.h&gt;

32731 double fmax(double *x*, double *y*);32732 float fmaxf(float *x*, float *y*);32733 long double fmaxl(long double *x*, long double *y*);32734 **DESCRIPTION**32735 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
32736 conflict between the requirements described here and the ISO C standard is unintentional. This  
32737 volume of POSIX.1-2024 defers to the ISO C standard.32738 MX These functions shall determine the maximum numeric value of their arguments. NaN  
32739 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,  
32740 then these functions shall choose the numeric value.32741 **RETURN VALUE**32742 Upon successful completion, these functions shall return the maximum numeric value of their  
32743 arguments.32744 MX The returned value shall be exact and shall be independent of the current rounding direction  
32745 mode.

32746 If just one argument is a NaN, the other argument shall be returned.

32747 If *x* and *y* are NaN, a NaN shall be returned.32748 **ERRORS**

32749 No errors are defined.

32750 **EXAMPLES**

32751 None.

32752 **APPLICATION USAGE**

32753 None.

32754 **RATIONALE**

32755 None.

32756 **FUTURE DIRECTIONS**

32757 None.

32758 **SEE ALSO**32759 *fdim()*, *fmin()*

32760 XBD &lt;math.h&gt;

32761 **CHANGE HISTORY**

32762 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

32763 **Issue 7**

32764 Austin Group Interpretation 1003.1-2001 #007 is applied.

32765 **Issue 8**32766 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
32767 standard.

32768 **NAME**

32769 fmemopen — open a memory buffer stream

32770 **SYNOPSIS**

```
32771 CX #include <stdio.h>
32772 FILE *fmemopen(void *restrict buf, size_t max_size,
32773               const char *restrict mode);
```

32774 **DESCRIPTION**

32775 The *fmemopen()* function shall associate the buffer given by the *buf* and *max\_size* arguments with  
 32776 a stream. The *buf* argument shall be either a null pointer or point to a buffer that is at least  
 32777 *max\_size* bytes long.

32778 The *mode* argument points to a string. If the string is one of the following, the stream shall be  
 32779 opened in the indicated mode. Otherwise, the behavior is undefined.

32780	<i>r</i> or <i>rb</i>	Open the stream for reading.
32781	<i>w</i> or <i>wb</i>	Open the stream for writing.
32782	<i>a</i> or <i>ab</i>	Append; open the stream for writing.
32783	<i>r+</i> or <i>rb+</i> or <i>r+b</i>	Open the stream for update (reading and writing).
32784	<i>w+</i> or <i>wb+</i> or <i>w+b</i>	Open the stream for update (reading and writing).
32785	<i>a+</i> or <i>ab+</i> or <i>a+b</i>	Append; open the stream for update (reading and writing).

32786 If the *mode* argument begins with 'w' and *max\_size* is not zero, the buffer contents shall be  
 32787 truncated by writing a null byte at the beginning. If the *mode* argument includes 'b', the results  
 32788 are implementation-defined.

32789 If a null pointer is specified as the *buf* argument, *fmemopen()* shall allocate *max\_size* bytes of  
 32790 memory as if by a call to *malloc()*. This buffer shall be automatically freed when the stream is  
 32791 closed. Because this feature is only useful when the stream is opened for updating (because  
 32792 there is no way to get a pointer to the buffer) the *fmemopen()* call may fail if the *mode* argument  
 32793 does not include a '+'.  
 32794

32794 When a stream is opened for reading only and *buf* is not a null pointer, the buffer pointed to by  
 32795 *buf* shall not be modified by any operation performed on the stream.

32796 The stream shall maintain a current position in the buffer. This position shall be initially set to  
 32797 either the beginning of the buffer (for *r* and *w* modes) or to the first null byte in the buffer (for *a*  
 32798 modes). If no null byte is found in append mode, the initial position shall be set to one byte after  
 32799 the end of the buffer.

32800 If *buf* is a null pointer, the initial position shall always be set to the beginning of the buffer.

32801 The stream shall also maintain the end position of the current buffer contents; use of *fseek()* or  
 32802 *fseeko()* on the stream with *SEEK\_END* shall seek relative to this end position. If *mode* starts  
 32803 with 'r', the end position shall be set to the value given by the *max\_size* argument and shall not  
 32804 change. Otherwise, the stream is writable and the end position shall be variable; for modes *w*  
 32805 and *w+* the initial end position shall be zero and for modes *a* and *a+* the initial end position shall  
 32806 be:

- 32807 • Zero, if *buf* is a null pointer

- 32808           • The position of the first null byte in the buffer, if one is found
- 32809           • The value of the *max\_size* argument, if *buf* is not a null pointer and no null byte is found

32810           A read operation on the stream shall not advance the current buffer position beyond the current  
 32811           buffer end position. Reaching the buffer end position in a read operation shall count as “end-of-  
 32812           file”. Null bytes in the buffer shall have no special meaning for reads. The read operation shall  
 32813           start at the current buffer position of the stream.

32814           A write operation shall start either at the current position of the stream (if *mode* has not specified  
 32815           'a' as the first character) or at the current end position of the stream (if *mode* had 'a' as the  
 32816           first character). If the current position at the end of the write is larger than the current buffer end  
 32817           position, the current buffer end position shall be set to the current position. A write operation on  
 32818           the stream shall not advance the current buffer end position beyond the size given in the  
 32819           *max\_size* argument.

32820           When a stream open for update (the *mode* argument includes '+') or for writing only is  
 32821           successfully written and the write advances the current buffer end position, a null byte shall be  
 32822           written at the new buffer end position if it fits.

32823           An attempt to seek a memory buffer stream to a negative position or to a position larger than the  
 32824           buffer size given in the *max\_size* argument shall fail.

#### 32825 RETURN VALUE

32826           Upon successful completion, *fmemopen()* shall return a pointer to the object controlling the  
 32827           stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

#### 32828 ERRORS

32829           The *fmemopen()* function shall fail if:

32830           [EMFILE]           {STREAM\_MAX} streams are currently open in the calling process.

32831           The *fmemopen()* function may fail if:

32832           [EINVAL]           The value of the *mode* argument is not valid.

32833           [EINVAL]           The *buf* argument is a null pointer and the *mode* argument does not include a  
 32834           '+' character.

32835           [EINVAL]           The *max\_size* argument specifies a buffer size of zero and the implementation  
 32836           does not support this.

32837           [ENOMEM]          The *buf* argument is a null pointer and the allocation of a buffer of length  
 32838           *max\_size* has failed.

32839           [EMFILE]           {FOPEN\_MAX} streams are currently open in the calling process.

#### 32840 EXAMPLES

```

32841           #include <stdio.h>
32842           #include <string.h>
32843           static char buffer[] = "foobar";
32844           int
32845           main (void)
32846           {
32847               int ch;
32848               FILE *stream;
32849               stream = fmemopen(buffer, strlen (buffer), "r");

```

```

32850         if (stream == NULL)
32851             /* handle error */;
32852         while ((ch = fgetc(stream)) != EOF)
32853             printf("Got %c\n", ch);
32854         fclose(stream);
32855         return (0);
32856     }

```

32857 This program produces the following output:

```

32858     Got f
32859     Got o
32860     Got o
32861     Got b
32862     Got a
32863     Got r

```

### 32864 APPLICATION USAGE

32865 Implementations differ as regards how a 'b' in the *mode* argument affects the behavior. For  
32866 some the 'b' has no effect, as is required for *fopen()*; others distinguish between text and binary  
32867 modes.

32868 Note that *buf* will not be null terminated if *max\_size* bytes are written to the memory stream.  
32869 Applications wanting to guarantee that the buffer will be null terminated need to call  
32870 *fmemopen()* with *max\_size* set to one byte smaller than the actual size of *buf* and set *buf[max\_size]*  
32871 to a null byte.

32872 This standard intentionally leaves the behavior of 'e' and 'x' in the *mode* argument undefined;  
32873 implementations might silently ignore them so that *fmemopen()* may accept the same mode  
32874 strings as *fopen()*, or may reject them as invalid.

### 32875 RATIONALE

32876 This interface has been introduced to eliminate many of the errors encountered in the  
32877 construction of strings, notably overflowing of strings. This interface prevents overflow.

### 32878 FUTURE DIRECTIONS

32879 A future version of this standard may require support of zero-length buffer streams explicitly.

### 32880 SEE ALSO

32881 [\*fdopen\(\)\*](#), [\*fopen\(\)\*](#), [\*freopen\(\)\*](#), [\*fseek\(\)\*](#), [\*malloc\(\)\*](#), [\*open\\_memstream\(\)\*](#)

32882 XBD [\*\*<stdio.h>\*\*](#)

### 32883 CHANGE HISTORY

32884 First released in Issue 7.

32885 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0142 [461], XSH/TC1-2008/0143 [396],  
32886 XSH/TC1-2008/0144 [396], XSH/TC1-2008/0145 [461], XSH/TC1-2008/0146 [461],  
32887 XSH/TC1-2008/0147 [461], XSH/TC1-2008/0148 [461], XSH/TC1-2008/0149 [461], and  
32888 XSH/TC1-2008/0150 [396] are applied.

32889 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0117 [587], XSH/TC2-2008/0118  
32890 [586,818], and XSH/TC2-2008/0119 [818] are applied.

32891 **Issue 8**

32892 Austin Group Defects 456 and 657 are applied, making the behavior implementation-defined  
32893 when the *mode* argument includes 'b'.

32894 Austin Group Defect 1144 is applied, adding a requirement that operations on streams opened  
32895 for reading only do not modify the buffer pointed to by *buf*.

32896 **NAME**

32897 fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

32898 **SYNOPSIS**

32899 #include &lt;math.h&gt;

32900 double fmin(double *x*, double *y*);32901 float fminf(float *x*, float *y*);32902 long double fminl(long double *x*, long double *y*);32903 **DESCRIPTION**32904 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
32905 conflict between the requirements described here and the ISO C standard is unintentional. This  
32906 volume of POSIX.1-2024 defers to the ISO C standard.32907 MX These functions shall determine the minimum numeric value of their arguments. NaN  
32908 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,  
32909 then these functions shall choose the numeric value.32910 **RETURN VALUE**32911 Upon successful completion, these functions shall return the minimum numeric value of their  
32912 arguments.32913 MX The returned value shall be exact and shall be independent of the current rounding direction  
32914 mode.

32915 If just one argument is a NaN, the other argument shall be returned.

32916 If *x* and *y* are NaN, a NaN shall be returned.32917 **ERRORS**

32918 No errors are defined.

32919 **EXAMPLES**

32920 None.

32921 **APPLICATION USAGE**

32922 None.

32923 **RATIONALE**

32924 None.

32925 **FUTURE DIRECTIONS**

32926 None.

32927 **SEE ALSO**32928 *fdim()*, *fmax()*

32929 XBD &lt;math.h&gt;

32930 **CHANGE HISTORY**

32931 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

32932 **Issue 7**

32933 Austin Group Interpretation 1003.1-2001 #008 is applied.

32934 **Issue 8**32935 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
32936 standard.



32937 **NAME**

32938 fmod, fmodf, fmodl — floating-point remainder value function

32939 **SYNOPSIS**

32940 #include &lt;math.h&gt;

32941 double fmod(double *x*, double *y*);32942 float fmodf(float *x*, float *y*);32943 long double fmodl(long double *x*, long double *y*);32944 **DESCRIPTION**32945 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
32946 conflict between the requirements described here and the ISO C standard is unintentional. This  
32947 volume of POSIX.1-2024 defers to the ISO C standard.32948 These functions shall return the floating-point remainder of the division of *x* by *y*.32949 An application wishing to check for error situations should set *errno* to zero and call  
32950 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
32951 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
32952 zero, an error has occurred.32953 **RETURN VALUE**32954 These functions shall return the value  $x - i * y$ , for some integer *i* such that, if *y* is non-zero, the  
32955 result has the same sign as *x* and magnitude less than the magnitude of *y*.32956 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
32957 MXX and *fmod*(), *modf*(), and *fmodl*() shall return 0.0, or (if the IEC 60559 Floating-Point option is not  
32958 supported) an implementation-defined value no greater in magnitude than DBL\_MIN,  
32959 FLT\_MIN, and LDBL\_MIN, respectively.32960 MX If *x* or *y* is NaN, a NaN shall be returned, and none of the conditions below shall be considered.32961 If *y* is zero, a domain error shall occur, and a NaN shall be returned.32962 If *x* is infinite, a domain error shall occur, and a NaN shall be returned.32963 If *x* is  $\pm 0$  and *y* is not zero,  $\pm 0$  shall be returned.32964 If *x* is not infinite and *y* is  $\pm \text{Inf}$ , *x* shall be returned.32965 When subnormal results are supported, the returned value shall be exact and shall be  
32966 independent of the current rounding direction mode.32967 **ERRORS**

32968 These functions shall fail if:

32969 MX **Domain Error** The *x* argument is infinite or *y* is zero.32970 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
32971 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
32972 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
32973 shall be raised.

32974 These functions may fail if:

32975 **Range Error** The result underflows.32976 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
32977 then *errno* shall be set to [ERANGE]. If the integer expression  
32978 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
32979 floating-point exception shall be raised.

**32980 EXAMPLES**

32981 None.

**32982 APPLICATION USAGE**

32983 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
32984 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

**32985 RATIONALE**

32986 None.

**32987 FUTURE DIRECTIONS**

32988 None.

**32989 SEE ALSO**

32990 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

32991 [Section 4.23](#) (on page 109), [<math.h>](#)

**32992 CHANGE HISTORY**

32993 First released in Issue 1. Derived from Issue 1 of the SVID.

**32994 Issue 5**

32995 The DESCRIPTION is updated to indicate how an application should check for an error. This  
32996 text was previously published in the APPLICATION USAGE section.

**32997 Issue 6**

32998 The behavior for when the *y* argument is zero is now defined.

32999 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999  
33000 standard.

33001 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
33002 revised to align with the ISO/IEC 9899:1999 standard.

33003 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
33004 marked.

**33005 Issue 7**

33006 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0151 [68], XSH/TC1-2008/0152 [320],  
33007 and XSH/TC1-2008/0153 [68] are applied.

33008 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0120 [605] is applied.

**33009 Issue 8**

33010 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
33011 standard.

33012 **NAME**33013 `fmtmsg` — display a message in the specified format on standard error and/or a system console33014 **SYNOPSIS**

```
33015 XSI      #include <fmtmsg.h>
33016
33016      int fmtmsg(long classification, const char *label, int severity,
33017                const char *text, const char *action, const char *tag);
```

33018 **DESCRIPTION**33019 The `fmtmsg()` function shall display messages in a specified format instead of the traditional  
33020 `printf()` function.33021 Based on a message's classification component, `fmtmsg()` shall write a formatted message either  
33022 to standard error, to the console, or to both.33023 A formatted message consists of up to five components as defined below. The component  
33024 *classification* is not part of a message displayed to the user, but defines the source of the message  
33025 and directs the display of the formatted message.

33026 *classification*      Contains the sum of identifying values constructed from the constants defined  
33027 below. Any one identifier from a subclass may be used in combination with a  
33028 single identifier from a different subclass. Two or more identifiers from the  
33029 same subclass should not be used together, with the exception of identifiers  
33030 from the display subclass. (Both display subclass identifiers may be used so  
33031 that messages can be displayed to both standard error and the system  
33032 console.)

33033 **Major Classifications**33034 Identifies the source of the condition. Identifiers are: MM\_HARD  
33035 (hardware), MM\_SOFT (software), and MM\_FIRM (firmware).33036 **Message Source Subclassifications**33037 Identifies the type of software in which the problem is detected.  
33038 Identifiers are: MM\_APPL (application), MM\_UTIL (utility), and  
33039 MM OPSYS (operating system).33040 **Display Subclassifications**33041 Indicates where the message is to be displayed. Identifiers are:  
33042 MM\_PRINT to display the message on the standard error stream,  
33043 MM\_CONSOLE to display the message on the system console. One or  
33044 both identifiers may be used.33045 **Status Subclassifications**33046 Indicates whether the application can recover from the condition.  
33047 Identifiers are: MM\_RECOVER (recoverable) and MM\_NRECOV (non-  
33048 recoverable).33049 An additional identifier, MM\_NULLMC, indicates that no classification  
33050 component is supplied for the message.33051 *label*                Identifies the source of the message. The format is two fields separated by a  
33052 <colon>. The first field is up to 10 bytes, the second is up to 14 bytes.33053 *severity*             Indicates the seriousness of the condition. Identifiers for the levels of *severity*  
33054 are:

33055		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
33056			
33057		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
33058			
33059		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
33060			
33061			
33062		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
33063			
33064		MM_NOSEV	Indicates that no severity level is supplied for the message.
33065	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
33066			
33067			
33068	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
33069			
33070			
33071	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
33072			
33073			

33074 The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg()*  
 33075 which message components it is to select when writing messages to standard error. The value of  
 33076 *MSGVERB* shall be a <colon>-separated list of optional keywords. Valid keywords are: *label*,  
 33077 *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the  
 33078 component's value is not the component's null value, *fmtmsg()* shall include that component in  
 33079 the message when writing the message to standard error. If *MSGVERB* does not include a  
 33080 keyword for a message component, that component shall not be included in the display of the  
 33081 message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the  
 33082 null string, if its value is not of the correct format, or if it contains keywords other than the valid  
 33083 ones listed above, *fmtmsg()* shall select all components.

33084 *MSGVERB* shall determine which components are selected for display to standard error. All  
 33085 message components shall be included in console messages.

#### 33086 RETURN VALUE

33087 The *fmtmsg()* function shall return one of the following values:

33088	MM_OK	The function succeeded.
33089	MM_NOTOK	The function failed completely.
33090	MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.
33091		
33092	MM_NOCON	The function was unable to generate a console message, but otherwise succeeded.
33093		

#### 33094 ERRORS

33095 None.

33096 **EXAMPLES**33097 1. The following example of *fmtmsg()*:33098 `fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",`  
33099 `"refer to cat in user's reference manual", "XSI:cat:001")`

33100 produces a complete message in the specified message format:

33101 `XSI:cat: ERROR: illegal option`33102 `TO FIX: refer to cat in user's reference manual XSI:cat:001`33103 2. When the environment variable *MSGVERB* is set as follows:33104 `MSGVERB=severity:text:action`33105 and Example 1 is used, *fmtmsg()* produces:33106 `ERROR: illegal option`33107 `TO FIX: refer to cat in user's reference manual`33108 **APPLICATION USAGE**33109 One or more message components may be systematically omitted from messages generated by  
33110 an application by using the null value of the argument for that component.33111 **RATIONALE**

33112 None.

33113 **FUTURE DIRECTIONS**

33114 None.

33115 **SEE ALSO**33116 [\*fprintf\(\)\*](#)33117 XBD <[fmtmsg.h](#)>33118 **CHANGE HISTORY**

33119 First released in Issue 4, Version 2.

33120 **Issue 5**

33121 Moved from X/OPEN UNIX extension to BASE.

33122 **NAME**

33123 fnmatch — match a filename string or a pathname

33124 **SYNOPSIS**

33125 #include &lt;fnmatch.h&gt;

33126 int fnmatch(const char \*pattern, const char \*string, int flags);

33127 **DESCRIPTION**

33128 The *fnmatch()* function shall match patterns as described in XCU [Section 2.14.1](#) (on page 2523)  
 33129 and [Section 2.14.2](#) (on page 2524). It checks the string specified by the *string* argument to see if it  
 33130 matches the pattern specified by the *pattern* argument.

33131 The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-  
 33132 inclusive OR of zero or more of the flags defined in <fnmatch.h>. If the FNM\_PATHNAME flag  
 33133 is set in *flags*, then a <slash> character (' / ') in *string* shall be explicitly matched by a <slash> in  
 33134 *pattern*; it shall not be matched by either the <asterisk> or <question-mark> special characters,  
 33135 nor by a bracket expression. If the FNM\_PATHNAME flag is not set, the <slash> character shall  
 33136 be treated as an ordinary character.

33137 If FNM\_NOESCAPE is not set in *flags*, a <backslash> character can be used as an escape  
 33138 character as described in XCU [Section 2.14.1](#) (on page 2523). If *pattern* ends with an unescaped  
 33139 <backslash>, the behavior is unspecified. If FNM\_NOESCAPE is set, a <backslash> character  
 33140 shall be treated as an ordinary character.

33141 If FNM\_PERIOD is set in *flags*, then a leading <period> (' . ') in *string* shall match a <period> in  
 33142 *pattern*; as described by rule 2 in XCU [Section 2.14.3](#) (on page 2525) where the location of  
 33143 “leading” is indicated by the value of FNM\_PATHNAME:

- 33144 • If FNM\_PATHNAME is set, a <period> is “leading” if it is the first character in *string* or if  
 33145 it immediately follows a <slash>.
- 33146 • If FNM\_PATHNAME is not set, a <period> is “leading” only if it is the first character of  
 33147 *string*.

33148 If FNM\_PERIOD is not set, then no special restrictions are placed on matching a period.

33149 If FNM\_CASEFOLD or FNM\_IGNORECASE is set, *string* and *pattern* shall be compared in a  
 33150 case-insensitive manner. See XBD [Section 4.1](#) (on page 95).

33151 **RETURN VALUE**

33152 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no  
 33153 match, *fnmatch()* shall return FNM\_NOMATCH, which is defined in <fnmatch.h>. If an error  
 33154 occurs, *fnmatch()* shall return another non-zero value.

33155 **ERRORS**

33156 No errors are defined.

33157 **EXAMPLES**

33158 None.

33159 **APPLICATION USAGE**

33160 The *fnmatch()* function has two major uses. It could be used by an application or utility that  
 33161 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of  
 33162 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that  
 33163 need to match strings in a similar manner.

33164 The name *fnmatch()* is intended to imply *filename* match, rather than *pathname* match. The  
 33165 default action of this function is to match filename strings, rather than pathnames, since it gives  
 33166 no special significance to the <slash> character. With the FNM\_PATHNAME flag, *fnmatch()* does

33167 match pathnames, but without tilde expansion, parameter expansion, or special treatment for a  
33168 <period> at the beginning of a filename.

#### 33169 RATIONALE

33170 This function replaced the REG\_FILENAME flag of *regcomp()* in early proposals of this volume  
33171 of POSIX.1-2024. It provides virtually the same functionality as the *regcomp()* and *regexec()*  
33172 functions using the REG\_FILENAME and REG\_FSLASH flags (the REG\_FSLASH flag was  
33173 proposed for *regcomp()*, and would have had the opposite effect from FNM\_PATHNAME), but  
33174 with a simpler function and less system overhead.

#### 33175 FUTURE DIRECTIONS

33176 None.

#### 33177 SEE ALSO

33178 *glob()*, *wordexp()*

33179 XBD <fnmatch.h>

#### 33180 CHANGE HISTORY

33181 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 33182 Issue 5

33183 Moved from POSIX2 C-language Binding to BASE.

#### 33184 Issue 7

33185 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0154 [291] and XSH/TC1-2008/0155  
33186 [291] are applied.

33187 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0121 [806] is applied.

#### 33188 Issue 8

33189 Austin Group Defect 1031 is applied, adding FNM\_CASEFOLD and FNM\_IGNORECASE.

33190 Austin Group Defect 1287 is applied, changing the description of how <backslash> is handled (if  
33191 FNM\_NOESCAPE is not set) to refer to XCU [Section 2.14.1](#) (on page 2523).

33192 Austin Group Defect 1444 is applied, correcting cross-references to *wordexp()*.

33193 **NAME**

33194 fopen — open a stream

33195 **SYNOPSIS**

33196 #include &lt;stdio.h&gt;

33197 FILE \*fopen(const char \*restrict pathname, const char \*restrict mode);

33198 **DESCRIPTION**

33199 CX Except for the “exclusive access” requirement (see below), the functionality described on this  
 33200 reference page is aligned with the ISO C standard. Any other conflict between the requirements  
 33201 described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to  
 33202 the ISO C standard for all *fopen()* functionality except in relation to “exclusive access”.

33203 The *fopen()* function shall open the file whose pathname is the string pointed to by *pathname*,  
 33204 and associates a stream with it.

33205 The *mode* argument points to a character string. The behavior is unspecified if any character  
 33206 occurs more than once in the string. If the string begins with one of the following characters,  
 33207 then the file shall be opened in the indicated mode. Otherwise, the behavior is undefined.

33208 'r' Open file for reading.

33209 'w' Truncate to zero length or create file for writing.

33210 'a' Append; open or create file for writing at end-of-file.

33211 CX The remainder of the string can contain any of the following characters, in any order, and  
 33212 further affect how the file is opened:

33213 CX 'b' This character shall have no effect, but is allowed for ISO C standard conformance.

33214 CX 'e' The underlying file descriptor shall have the FD\_CLOEXEC flag atomically set.

33215 CX 'x' If the first character of *mode* is 'w' or 'a', then the function shall fail if the file already  
 33216 exists, or cannot be created; if the file does not exist and can be created, it shall be created  
 33217 CX with an implementation-defined form of exclusive (also known as non-shared) access,  
 33218 CX if supported by the underlying file system, provided the resulting file permissions are the  
 33219 same as they would be without the 'x' modifier. If the first character of *mode* is 'r', the  
 33220 effect is implementation-defined.

33221 **Note:** The ISO C standard requires exclusive access “to the extent that the underlying file  
 33222 system supports exclusive access”, but does not define what it means by this. Taken at  
 33223 face value—that systems must do whatever they are capable of, at the file system level,  
 33224 in order to exclude access by others—this would require POSIX.1 systems to set the file  
 33225 permissions in a way that prevents access by other users and groups. Consequently, this  
 33226 volume of POSIX.1-2024 does not defer to the ISO C standard as regards the “exclusive  
 33227 access” requirement.

33228 ' +' The file shall be opened for update (both reading and writing), rather than just reading or  
 33229 just writing.

33230 Opening a file with read mode ('r' as the first character in the *mode* argument) shall fail if the  
 33231 file does not exist or cannot be read.

33232 Opening a file with append mode ('a' as the first character in the *mode* argument) shall cause  
 33233 all subsequent writes to the file to be forced to the then current end-of-file, regardless of  
 33234 intervening calls to *fseek()*.

33235 When a file is opened with update mode ('+' in the *mode* argument), both input and output can  
 33236 be performed on the associated stream. However, the application shall ensure that output is not  
 33237 directly followed by input without an intervening call to *fflush()* or to a file positioning function



33238 (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly followed by output without an  
33239 intervening call to a file positioning function, unless the input operation encounters end-of-file.

33240 When opened, a stream is fully buffered if and only if it can be determined not to refer to an  
33241 interactive device. The error and end-of-file indicators for the stream shall be cleared.

33242 CX If the first character in *mode* is 'w' or 'a' and the file did not previously exist, upon successful  
33243 completion, *fopen()* shall mark for update the last data access, last data modification, and last file  
33244 status change timestamps of the file and the last file status change and last data modification  
33245 timestamps of the parent directory.

33246 If the first character in *mode* is 'w' or 'a' and the file did not previously exist, the *fopen()*  
33247 function shall create a file as if it called the *open()* function with a value appropriate for the *path*  
33248 argument interpreted from *pathname*, a value for the *oflag* argument as specified below, and a  
33249 value of S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH for the third  
33250 argument.

33251 If the first character in *mode* is 'w' and the file did previously exist, upon successful completion,  
33252 *fopen()* shall mark for update the last data modification and last file status change timestamps of  
33253 the file.

33254 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the  
33255 encoding rule shall be cleared, and the associated *mbstate\_t* object shall be set to describe an  
33256 initial conversion state.

33257 CX The file descriptor associated with the opened stream shall be allocated and opened as if by a  
33258 call to *open()* using the following flags:

<i>fopen()</i> Mode First Character	<i>fopen()</i> Mode Includes '+'	Initial <i>open()</i> Flags
'r'	no	O_RDONLY
'w'	no	O_WRONLY   O_CREAT   O_TRUNC
'a'	no	O_WRONLY   O_CREAT   O_APPEND
'r'	yes	O_RDWR
'w'	yes	O_RDWR   O_CREAT   O_TRUNC
'a'	yes	O_RDWR   O_CREAT   O_APPEND

33267 If, and only if, the 'e' *mode* string character is specified, the O\_CLOEXEC flag shall be OR'ed  
33268 into the initial *open()* flags specified in the above table.

33269 If, and only if, the 'x' *mode* string character is specified together with either 'w' or 'a', the  
33270 O\_EXCL flag shall be OR'ed into the initial *open()* flags specified in the above table.

33271 If *mode* includes 'x' and the underlying file system supports exclusive access (see above)  
33272 enabled by the use of implementation-specific flags to *open()*, then the behavior shall be as if  
33273 those flags are also included.

33274 When using *mode* strings specified by this standard, the implementation shall behave as if no  
33275 other flags had been passed to *open()*.

#### 33276 RETURN VALUE

33277 Upon successful completion, *fopen()* shall return a pointer to the object controlling the stream.

33278 CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

33279 **ERRORS**33280 The *fopen()* function shall fail if:

33281	CX	[EACCES]	Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>mode</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.
33282			
33283			
33284			
33285	CX	[EEXIST]	The <i>mode</i> argument begins with <i>w</i> or <i>a</i> and includes <i>x</i> , but the file already exists.
33286			
33287	CX	[EILSEQ]	The <i>mode</i> argument begins with <i>w</i> or <i>a</i> , the file did not previously exist, and the last pathname component is not a portable filename and cannot be created in the target directory.
33288			
33289			
33290	CX	[EINTR]	A signal was caught during <i>fopen()</i> .
33291	CX	[EISDIR]	The named file is a directory and <i>mode</i> requires write access.
33292	CX	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>pathname</i> argument.
33293			
33294	CX	[EMFILE]	All file descriptors available to the process are currently open.
33295	CX	[EMFILE]	{STREAM_MAX} streams are currently open in the calling process.
33296	CX	[ENAMETOOLONG]	
33297			The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
33298			
33299			
33300	CX	[ENFILE]	The maximum allowable number of files is currently open in the system.
33301	CX	[ENOENT]	The <i>mode</i> string begins with 'r' and a component of <i>pathname</i> does not name an existing file, or <i>mode</i> begins with 'w' or 'a' and a component of the path prefix of <i>pathname</i> does not name an existing file, or <i>pathname</i> is an empty string.
33302			
33303			
33304			
33305	CX	[ENOENT] or [ENOTDIR]	
33306			The <i>pathname</i> argument contains at least one non- <i>&lt;slash&gt;</i> character and ends with one or more trailing <i>&lt;slash&gt;</i> characters. If <i>pathname</i> without the trailing <i>&lt;slash&gt;</i> characters would name an existing file, an [ENOENT] error shall not occur.
33307			
33308			
33309			
33310	CX	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and the file was to be created.
33311			
33312	CX	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>pathname</i> argument contains at least one non- <i>&lt;slash&gt;</i> character and ends with one or more trailing <i>&lt;slash&gt;</i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
33313			
33314			
33315			
33316			
33317	CX	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
33318			
33319	CX	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <b>off_t</b> .
33320			

33321 CX [EROFS] The named file resides on a read-only file system and *mode* requires write  
33322 access.

33323 The *fopen()* function may fail if:

33324 CX [EINVAL] The value of the *mode* argument is not valid.

33325 CX [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
33326 resolution of the *pathname* argument.

33327 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

33328 CX [ENAMETOOLONG]

33329 The length of a component of a pathname is longer than {NAME\_MAX}.

33330 CX [ENOMEM] Insufficient storage space is available.

33331 CX [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *mode*  
33332 requires write access.

### 33333 EXAMPLES

#### 33334 Opening a File

33335 The following example tries to open the file named **file** for reading. The *fopen()* function returns  
33336 a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the  
33337 file, it just ignores it.

```
33338 #include <stdio.h>
33339 ...
33340 FILE *fp;
33341 ...
33342 void rgrep(const char *file)
33343 {
33344     ...
33345     if ((fp = fopen(file, "r")) == NULL)
33346         return;
33347     ...
33348 }
```

### 33349 APPLICATION USAGE

33350 If an application needs to create a file in a way that fails if the file already exists, and either  
33351 requires that it does not have exclusive access to the file or does not need exclusive access, it  
33352 should use *open()* with the O\_CREAT and O\_EXCL flags instead of using *fopen()* with an 'x'  
33353 in the mode. A stream can then be created, if needed, by calling *fdopen()* on the file descriptor  
33354 returned by *open()*.

### 33355 RATIONALE

33356 The 'e' mode character is provided as a convenience to avoid a data race in multi-threaded  
33357 applications. Without it, a file descriptor is leaked into a child process created by one thread in  
33358 the window between another thread creating a file descriptor with *fopen()* and then using  
33359 *fileno()* and *fcntl()* to set the FD\_CLOEXEC flag. It is also possible to avoid the race by using  
33360 *open()* with O\_CLOEXEC followed by *fdopen()*, however, there is no safe alternative for the  
33361 *freopen()* function, and consistency dictates that the 'e' modifier should be standardized for all  
33362 functions that accept *mode* strings.

33363 The ISO C standard only recognizes the '+', 'b', and 'x' characters in certain positions of the  
33364 *mode* string, leaving other arrangements as unspecified, and only permits 'x' in *mode* strings

33365 beginning with 'w'. This standard specifically requires support for all characters other than the  
 33366 first in the *mode* string to be recognized in any order. Thus, "wxe" and "wex" behave the same,  
 33367 and while "wx+" is unspecified in the ISO C standard, this standard requires it to have the same  
 33368 behavior as "w+x". This standard also requires that 'x' work for *mode* strings beginning with  
 33369 'a', as well as having implementation-defined behavior for *mode* strings beginning with 'r'.  
 33370 Therefore, while *open()* has undefined behavior if O\_EXCL is specified without O\_CREAT, the  
 33371 same is not true of *fopen()*.

33372 When 'x' is in mode, the ISO C standard requires that the file is created with exclusive access to  
 33373 the extent that the underlying system supports exclusive access. Although POSIX.1 does not  
 33374 specify any method of enabling exclusive access, it allows for the existence of an  
 33375 implementation-specific flag, or flags, that enable it. Note that they should be file creation flags  
 33376 if a file is being created, not file access mode flags (that is, ones that are included in  
 33377 O\_ACCMODE) or file status flags, so that they do not affect the value returned by *fcntl()* with  
 33378 F\_GETFL. On implementations that have such flags, if support for them is file system dependent  
 33379 and exclusive access is requested when using *fopen()* to create a file on a file system that does  
 33380 not support it, the flags must not be used if they would cause *fopen()* to fail.

33381 Some implementations support mandatory file locking as a means of enabling exclusive access  
 33382 to a file. Locks are set in the normal way, but instead of only preventing others from setting  
 33383 conflicting locks they prevent others from accessing the contents of the locked part of the file in a  
 33384 way that conflicts with the lock. However, unless the implementation has a way of setting a  
 33385 whole-file write lock on file creation, this does not satisfy the requirement in the ISO C standard  
 33386 that the file is "created with exclusive access to the extent that the underlying system supports  
 33387 exclusive access". (Having *fopen()* create the file and set a lock on the file as two separate  
 33388 operations is not the same, and it would introduce a race condition whereby another process  
 33389 could open the file and write to it (or set a lock) in between the two operations.) However, on all  
 33390 implementations that support mandatory file locking, its use is discouraged; therefore, it is  
 33391 recommended that implementations which support mandatory file locking do not add a means  
 33392 of creating a file with a whole-file exclusive lock set, so that *fopen()* is not required to enable  
 33393 mandatory file locking in order to conform to the ISO C standard. An implementation that has a  
 33394 means of creating a file with a whole-file exclusive lock set would need to provide a way to  
 33395 change the behavior of *fopen()* depending on whether the calling process is executing in a  
 33396 POSIX.1 conforming environment or an ISO C conforming environment.

33397 The typical implementation-defined behavior for mode "rx" is to ignore the 'x', but the  
 33398 standard developers did not wish to mandate this behavior. For example, an implementation  
 33399 could allow shared access for reading; that is, disallow a file that has been opened this way from  
 33400 also being opened for writing.

33401 Implementations are encouraged to have *fopen()* and *freopen()* report an [EILSEQ] error if *mode*  
 33402 begins with 'w' or 'a', the file did not previously exist, and the last component of *pathname*  
 33403 contains any bytes that have the encoded value of a <newline> character.

#### 33404 FUTURE DIRECTIONS

33405 None.

#### 33406 SEE ALSO

33407 [Section 2.5](#) (on page 521), *creat()*, *fclose()*, *fdopen()*, *fmemopen()*, *freopen()*, *open\_memstream()*

33408 XBD <stdio.h>

33409 **CHANGE HISTORY**

33410 First released in Issue 1. Derived from Issue 1 of the SVID.

33411 **Issue 5**

33412 Large File Summit extensions are added.

33413 **Issue 6**

33414 Extensions beyond the ISO C standard are marked.

33415 The following new requirements on POSIX implementations derive from alignment with the  
33416 Single UNIX Specification:

- 33417 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
33418 file description. This change is to support large files.
- 33419 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
33420 large files.
- 33421 • The [ELOOP] mandatory error condition is added.
- 33422 • The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional  
33423 error conditions are added.

33424 The normative text is updated to avoid use of the term “must” for application requirements.

33425 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 33426 • The prototype for *fopen()* is updated.
- 33427 • The DESCRIPTION is updated to note that if the argument *mode* points to a string other  
33428 than those listed, then the behavior is undefined.

33429 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
33430 [ELOOP] error condition is added.33431 **Issue 7**

33432 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode.

33433 Austin Group Interpretation 1003.1-2001 #143 is applied.

33434 Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set  
33435 on the open file description.

33436 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

33437 SD5-XSH-ERN-149 is applied, changing the {STREAM\_MAX} [EMFILE] error condition from a  
33438 “may fail” to a “shall fail”.

33439 Changes are made related to support for finegrained timestamps.

33440 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0156 [291,433], XSH/TC1-2008/0157  
33441 [146,433], XSH/TC1-2008/0158 [324], and XSH/TC1-2008/0159 [14] are applied.

33442 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0122 [822] is applied.

33443 **Issue 8**33444 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
33445 filenames containing any bytes that have the encoded value of a <newline> character.

33446 Austin Group Defect 293 is applied, adding the [EILSEQ] error.

33447 Austin Group Defects 411 and 1524 are applied, adding the 'e' and 'x' *mode* string characters.

33448 Austin Group Defect 1200 is applied, correcting the argument name in the [ELOOP] errors.

33449  
33450

Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018 standard.

33451 **NAME**

33452 fork — create a new process

33453 **SYNOPSIS**

```
33454 #include <unistd.h>
33455 pid_t fork(void);
33456 pid_t _Fork(void);
```

33457 **DESCRIPTION**

33458 The `fork()` function shall create a new process. The new process (child process) shall be an exact  
 33459 copy of the calling process (parent process) except as detailed below:

- 33460 • The child process shall have a unique process ID.
- 33461 • The child process ID also shall not match any active process group ID.
- 33462 • The child process shall have a different parent process ID, which shall be the process ID of  
 33463 the calling process.
- 33464 • The child process shall have its own copy of the parent's file descriptors, except for those  
 33465 whose `FD_CLOFORK` flag is set (see `fcntl()`). Each of the child's file descriptors shall refer  
 33466 to the same open file description with the corresponding file descriptor of the parent.
- 33467 • The child process shall have its own copy of the parent's open directory streams. Each  
 33468 open directory stream in the child process may share directory stream positioning with the  
 33469 corresponding directory stream of the parent.
- 33470 • The child process shall have its own copy of the parent's message catalog descriptors.
- 33471 • The child process values of `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` shall be set to  
 33472 0.
- 33473 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be  
 33474 canceled; see `alarm()`.
- 33475 XSI • All `semadj` values shall be cleared.
- 33476 • Process-owned file locks set by the parent process shall not be inherited by the child  
 33477 process.
- 33478 • The set of signals pending for the child process shall be initialized to the empty set.
- 33479 XSI • Interval timers shall be reset in the child process.
- 33480 • Any semaphores that are open in the parent process shall also be open in the child process.
- 33481 ML • The child process shall not inherit any address space memory locks established by the  
 33482 parent process via calls to `mlockall()` or `mlock()`.
- 33483 • Memory mappings created in the parent shall be retained in the child process.  
 33484 `MAP_PRIVATE` mappings inherited from the parent shall also be `MAP_PRIVATE`  
 33485 mappings in the child, and any modifications to the data in these mappings made by the  
 33486 parent prior to calling `fork()` shall be visible to the child. Any modifications to the data in  
 33487 `MAP_PRIVATE` mappings made by the parent after `fork()` returns shall be visible only to  
 33488 the parent. Modifications to the data in `MAP_PRIVATE` mappings made by the child shall  
 33489 be visible only to the child.
- 33490 PS • For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the child process shall inherit  
 33491 the policy and priority settings of the parent process during a `fork()` function. For other  
 33492 scheduling policies, the policy and priority settings on `fork()` are implementation-defined.

- 33493 • Per-process timers created by the parent shall not be inherited by the child process.
- 33494 MSG • The child process shall have its own copy of the message queue descriptors of the parent.  
33495 Each of the message descriptors of the child shall refer to the same open message queue  
33496 description as the corresponding message descriptor of the parent.
- 33497 • No asynchronous input or asynchronous output operations shall be inherited by the child  
33498 process. Any use of asynchronous control blocks created by the parent produces undefined  
33499 behavior.
- 33500 • A process shall be created with a single thread. If a multi-threaded process calls *fork()*, the  
33501 new process shall contain a replica of the calling thread and its entire address space,  
33502 possibly including the states of mutexes and other resources. Consequently, the application  
33503 shall ensure that the child process only executes async-signal-safe operations until such  
33504 time as one of the *exec* functions is successful.
- 33505 • Any locks held by any thread in the calling process that have been set to be process-shared  
33506 shall not be held by the child process. For locks held by any thread in the calling process  
33507 that have not been set to be process-shared, any attempt by the child process to perform  
33508 any operation on the lock results in undefined behavior (regardless of whether the calling  
33509 process is single-threaded or multi-threaded).
- 33510 CPT • The initial value of the CPU-time clock of the child process shall be set to zero.
- 33511 TCT • The initial value of the CPU-time clock of the single thread of the child process shall be set  
33512 to zero.

33513 All other process characteristics defined by POSIX.1-2024 shall be the same in the parent and  
33514 child processes. The inheritance of process characteristics not defined by POSIX.1-2024 is  
33515 unspecified by POSIX.1-2024.

33516 After *fork()*, both the parent and the child processes shall be capable of executing independently  
33517 before either one terminates.

33518 The *\_Fork()* function shall be equivalent to *fork()*, except that fork handlers established by means  
33519 of the *pthread\_atfork()* function shall not be called and *\_Fork()* shall be async-signal-safe.

#### 33520 RETURN VALUE

33521 Upon successful completion, *fork()* shall return 0 to the child process and shall return the  
33522 process ID of the child process to the parent process. Both processes shall continue to execute  
33523 from the *fork()* function. Otherwise, *-1* shall be returned to the parent process, no child process  
33524 shall be created, and *errno* shall be set to indicate the error.

#### 33525 ERRORS

33526 These functions shall fail if:

33527 [EAGAIN] The system lacked the necessary resources to create another process, or the  
33528 system-imposed limit on the total number of processes under execution  
33529 system-wide or by a single user {CHILD\_MAX} would be exceeded.

33530 These functions may fail if:

33531 [ENOMEM] Insufficient storage space is available.



33532 **EXAMPLES**

33533 None.

33534 **APPLICATION USAGE**

33535 When a multi-threaded process calls *fork()* or *\_Fork()*, there is no guarantee that thread-specific  
 33536 memory, such as stacks or thread-local storage, associated with threads in the parent other than  
 33537 the calling thread will still be available in the child. This is because threads in the parent other  
 33538 than the calling thread do not exist in the child. Consequently, the implementation of *fork()* or  
 33539 *\_Fork()* could remove that memory from the address space in the child, or reuse it for other  
 33540 purposes before returning or (in the case of *fork()*) calling any of the fork handlers registered by  
 33541 *pthread\_atfork()*. Therefore, applications should avoid using any pointers to thread-specific  
 33542 memory in the child that were passed to the calling thread from other threads in the parent.

33543 **RATIONALE**

33544 Many historical implementations have timing windows where a signal sent to a process group  
 33545 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the  
 33546 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of  
 33547 pending signals. This volume of POSIX.1-2024 does not require, or even permit, this behavior.  
 33548 However, it is pragmatic to expect that problems of this nature may continue to exist in  
 33549 implementations that appear to conform to this volume of POSIX.1-2024 and pass available  
 33550 verification suites. This behavior is only a consequence of the implementation failing to make  
 33551 the interval between signal generation and delivery totally invisible. From the application's  
 33552 perspective, a *fork()* call should appear atomic. A signal that is generated prior to the *fork()*  
 33553 should be delivered prior to the *fork()*. A signal sent to the process group after the *fork()* should  
 33554 be delivered to both parent and child. The implementation may actually initialize internal data  
 33555 structures corresponding to the child's set of pending signals to include signals sent to the  
 33556 process group during the *fork()*. Since the *fork()* call can be considered as atomic from the  
 33557 application's perspective, the set would be initialized as empty and such signals would have  
 33558 arrived after the *fork()*; see also **<signal.h>**.

33559 One approach that has been suggested to address the problem of signal inheritance across *fork()*  
 33560 is to add an [EINTR] error, which would be returned when a signal is detected during the call.  
 33561 While this is preferable to losing signals, it was not considered an optimal solution. Although it  
 33562 is not recommended for this purpose, such an error would be an allowable extension for an  
 33563 implementation.

33564 The [ENOMEM] error value is reserved for those implementations that detect and distinguish  
 33565 such a condition. This condition occurs when an implementation detects that there is not enough  
 33566 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate  
 33567 because there can never be enough memory (either primary or secondary storage) to perform  
 33568 the operation. Since *fork()* duplicates an existing process, this must be a condition where there is  
 33569 sufficient memory for one such process, but not for two. Many historical implementations  
 33570 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally  
 33571 distinct from [EAGAIN] from the perspective of a conforming application.

33572 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it  
 33573 and it should be reserved for the error condition specified there. The condition is not applicable  
 33574 on many implementations.

33575 IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.  
 33576 A system that single-threads processes was clearly not intended and is considered an  
 33577 unacceptable "toy implementation" of this volume of POSIX.1-2024. The only objection  
 33578 anticipated to the phrase "executing independently" is testability, but this assertion should be  
 33579 testable. Such tests require that both the parent and child can block on a detectable action of the  
 33580 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be

33581 possible for the system to conform to the intent of this volume of POSIX.1-2024.

33582 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it  
 33583 occurs or not is not in any practical sense under the control of the application because the  
 33584 condition is usually a consequence of the user's use of the system, not of the application's code.  
 33585 Thus, no application can or should rely upon its occurrence under any circumstances, nor  
 33586 should the exact semantics of what concept of "user" is used be of concern to the application  
 33587 developer. Validation writers should be cognizant of this limitation.

33588 There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread  
 33589 of control within the same program (which was originally only possible in POSIX by creating a  
 33590 new process); the other is to create a new process running a different program. In the latter case,  
 33591 the call to *fork()* is soon followed by a call to one of the *exec* functions.

33592 The general problem with making *fork()* work in a multi-threaded world is what to do with all  
 33593 of the threads. There are two alternatives. One is to copy all of the threads into the new process.  
 33594 This causes the programmer or implementation to deal with threads that are suspended on  
 33595 system calls or that might be about to execute system calls that should not be executed in the  
 33596 new process. The other alternative is to copy only the thread that calls *fork()*. This creates the  
 33597 difficulty that the state of process-local resources is usually held in process memory. If a thread  
 33598 that is not calling *fork()* holds a resource, that resource is never released in the child process  
 33599 because the thread whose job it is to release the resource does not exist in the child process.

33600 When a programmer is writing a multi-threaded program, the first described use of *fork()*,  
 33601 creating new threads in the same program, is provided by the *pthread\_create()* function. The  
 33602 *fork()* function is thus used only to run new programs, and the effects of calling functions that  
 33603 require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

33604 The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()*  
 33605 function lets all the threads in the parent be duplicated in the child. This essentially duplicates  
 33606 the state of the parent in the child. This allows threads in the child to continue processing and  
 33607 allows locks and the state to be preserved without explicit *pthread\_atfork()* code. The calling  
 33608 process has to ensure that the threads processing state that is shared between the parent and  
 33609 child (that is, file descriptors or MAP\_SHARED memory) behaves properly after *forkall()*. For  
 33610 example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two  
 33611 threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*.  
 33612 If this is not desired behavior, the parent process has to synchronize with such threads before  
 33613 calling *forkall()*.

33614 When *forkall()* is called, threads, other than the calling thread, that are in functions that can  
 33615 return with an [EINTR] error may have those functions return [EINTR] if the implementation  
 33616 cannot ensure that the function behaves correctly in the parent and child. In particular,  
 33617 *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, and *pthread\_cond\_wait()* need to return in  
 33618 order to ensure that the condition has not changed. These functions can be awakened by a  
 33619 spurious condition wakeup rather than returning [EINTR].

33620 **FUTURE DIRECTIONS**

33621 None.

33622 **SEE ALSO**

33623 *alarm()*, *exec*, *fcntl()*, *pthread\_atfork()*, *semop()*, *signal()*, *times()*

33624 XBD Section 4.15.2 (on page 104), <sys/types.h>, <unistd.h>

33625 **CHANGE HISTORY**

33626 First released in Issue 1. Derived from Issue 1 of the SVID.

33627 **Issue 5**33628 The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX  
33629 Threads Extension.33630 **Issue 6**33631 The following new requirements on POSIX implementations derive from alignment with the  
33632 Single UNIX Specification:

- 33633 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
33634 required for conforming implementations of previous POSIX specifications, it was not  
33635 required for UNIX applications.

33636 The following changes were made to align with the IEEE P1003.1a draft standard:

- 33637 • The effect of `fork()` on a pending alarm call in the child process is clarified.

33638 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

33639 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

33640 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the  
33641 DESCRIPTION and RATIONALE relating to fork handlers registered by the `pthread_atfork()`  
33642 function and async-signal safety.33643 **Issue 7**33644 Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous  
33645 input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.33646 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers,  
33647 and Threads options is moved to the Base.

33648 Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

33649 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0123 [858] is applied.

33650 **Issue 8**33651 Austin Group Defects 62, 1361, and 1383 are applied, adding the `_Fork()` function and removing  
33652 the requirement for `fork()` to be async-signal-safe.

33653 Austin Group Defect 768 is applied, adding OFD-owned file locks.

33654 Austin Group Defect 1112 is applied, clarifying the requirements for a child of a multi-threaded  
33655 process and for process-shared and non-process-shared locks held by any thread in the calling  
33656 process.

33657 Austin Group Defect 1114 is applied, changing the APPLICATION USAGE section.

33658 Austin Group Defect 1216 is applied, adding `pthread_cond_clockwait()`.33659 Austin Group Defect 1318 is applied, adding `FD_CLOFORK`.

33660 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

33661 **NAME**

33662 `fpathconf`, `pathconf` — get configurable pathname variables

33663 **SYNOPSIS**

```
33664 #include <unistd.h>
33665 long fpathconf(int fildes, int name);
33666 long pathconf(const char *path, int name);
```

33667 **DESCRIPTION**

33668 The `fpathconf()` and `pathconf()` functions shall determine the current value of a configurable limit  
33669 or option (*variable*) that is associated with a file or directory.

33670 For `pathconf()`, the *path* argument points to the pathname of a file or directory.

33671 For `fpathconf()`, the *filde*s argument is an open file descriptor.

33672 The *name* argument represents the variable to be queried relative to that file or directory.  
33673 Implementations shall support all of the variables listed in the following table and may support  
33674 others. The variables in the following table come from `<limits.h>` or `<unistd.h>` and the  
33675 symbolic constants, defined in `<unistd.h>`, are the corresponding values used for *name*.

33676	Variable	Value of <i>name</i>	Requirements
33677	{FILESIZEBITS}	_PC_FILESIZEBITS	4,7
33678	{LINK_MAX}	_PC_LINK_MAX	1
33679	{MAX_CANON}	_PC_MAX_CANON	2
33680	{MAX_INPUT}	_PC_MAX_INPUT	2
33681	{NAME_MAX}	_PC_NAME_MAX	3,4
33682	{PATH_MAX}	_PC_PATH_MAX	4,5
33683	{PIPE_BUF}	_PC_PIPE_BUF	6
33684	{POSIX2_SYMLINKS}	_PC_2_SYMLINKS	4
33685	{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	10
33686	{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	10
33687	{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	10
33688	{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	10
33689	{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	10
33690	{SYMLINK_MAX}	_PC_SYMLINK_MAX	4,9
33691	{TEXTDOMAIN_MAX}	_PC_TEXTDOMAIN_MAX	3,4
33692	_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
33693	_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
33694	_POSIX_VDISABLE	_PC_VDISABLE	2
33695	_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
33696	_POSIX_FALLOC	_PC_FALLOC	8
33697	_POSIX_PRIO_IO	_PC_PRIO_IO	8
33698	_POSIX_SYNC_IO	_PC_SYNC_IO	8
33699	_POSIX_TIMESTAMP_RESOLUTION	_PC_TIMESTAMP_RESOLUTION	1

33700 **Requirements**

- 33701 1. If *path* or *fildev* refers to a directory, the value returned shall apply to the directory itself.
- 33702 2. If *path* or *fildev* does not refer to a terminal file, it is unspecified whether an  
33703 implementation supports an association of the variable name with the specified file.
- 33704 3. If *path* or *fildev* refers to a directory, the value returned shall apply to filenames within the  
33705 directory.
- 33706 4. If *path* or *fildev* does not refer to a directory, it is unspecified whether an implementation  
33707 supports an association of the variable name with the specified file.
- 33708 5. If *path* or *fildev* refers to a directory, the value returned shall be the maximum length of a  
33709 relative pathname that would not cross any mount points when the specified directory is  
33710 the working directory.
- 33711 6. If *path* refers to a FIFO, or *fildev* refers to a pipe or FIFO, the value returned shall apply to  
33712 the referenced object. If *path* or *fildev* refers to a directory, the value returned shall apply to  
33713 any FIFO that exists or can be created within the directory. If *path* or *fildev* refers to any  
33714 other type of file, it is unspecified whether an implementation supports an association of  
33715 the variable name with the specified file.
- 33716 7. If *path* or *fildev* refers to a directory, the value returned shall apply to any files, other than  
33717 directories, that exist or can be created within the directory.
- 33718 8. If *path* or *fildev* refers to a directory, it is unspecified whether an implementation supports  
33719 an association of the variable name with the specified file.
- 33720 9. If *path* or *fildev* refers to a directory, the value returned shall be the maximum length of the  
33721 string that a symbolic link in that directory can contain.
- 33722 10. If *path* or *fildev* does not refer to a regular file, it is unspecified whether an  
33723 implementation supports an association of the variable name with the specified file. If an  
33724 implementation supports such an association for other than a regular file, the value  
33725 returned is unspecified.

33726 **RETURN VALUE**

33727 If *name* is an invalid value, both *pathconf()* and *fpathconf()* shall return  $-1$  and set *errno* to  
33728 indicate the error.

33729 If the variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum  
33730 value and the variable has no limit for the path or file descriptor, both *pathconf()* and *fpathconf()*  
33731 shall return  $-1$  without changing *errno*. Note that indefinite limits do not imply infinite limits;  
33732 see **<limits.h>**.

33733 If the implementation needs to use *path* to determine the value of *name* and the implementation  
33734 does not support the association of *name* with the file specified by *path*, or if the process did not  
33735 have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf()*  
33736 shall return  $-1$  and set *errno* to indicate the error.

33737 If the implementation needs to use *fildev* to determine the value of *name* and the implementation  
33738 does not support the association of *name* with the file specified by *fildev*, or if *fildev* is an invalid  
33739 file descriptor, *fpathconf()* shall return  $-1$  and set *errno* to indicate the error.

33740 Otherwise, *pathconf()* or *fpathconf()* shall return the current variable value for the file or  
33741 directory without changing *errno*. The value returned shall not be more restrictive than the  
33742 corresponding value available to the application when it was compiled with the  
33743 implementation's **<limits.h>** or **<unistd.h>**.

33744 If the variable corresponding to *name* is dependent on an unsupported option, the results are  
33745 unspecified.

### 33746 ERRORS

33747 The *pathconf()* function shall fail if:

33748 [EINVAL] The value of *name* is not valid.

33749 [EOVERFLOW] The value of *name* is `_PC_TIMESTAMP_RESOLUTION` and the resolution is  
33750 larger than `{LONG_MAX}`.

33751 The *pathconf()* function may fail if:

33752 [EACCES] Search permission is denied for a component of the path prefix.

33753 [EINVAL] The implementation does not support an association of the variable *name* with  
33754 the specified file.

33755 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
33756 argument.

33757 [ELOOP] More than `{SYMLOOP_MAX}` symbolic links were encountered during  
33758 resolution of the *path* argument.

33759 [ENAMETOOLONG]

33760 The length of a component of a pathname is longer than `{NAME_MAX}`.

33761 [ENAMETOOLONG]

33762 The length of a pathname exceeds `{PATH_MAX}`, or pathname resolution of a  
33763 symbolic link produced an intermediate result with a length that exceeds  
33764 `{PATH_MAX}`.

33765 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

33766 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
33767 directory nor a symbolic link to a directory, or the *path* argument contains at  
33768 least one non-`<slash>` character and ends with one or more trailing `<slash>`  
33769 characters and the last pathname component names an existing file that is  
33770 neither a directory nor a symbolic link to a directory.

33771 The *fpathconf()* function shall fail if:

33772 [EINVAL] The value of *name* is not valid.

33773 [EOVERFLOW] The value of *name* is `_PC_TIMESTAMP_RESOLUTION` and the resolution is  
33774 larger than `{LONG_MAX}`.

33775 The *fpathconf()* function may fail if:

33776 [EBADF] The *fdes* argument is not a valid file descriptor.

33777 [EINVAL] The implementation does not support an association of the variable *name* with  
33778 the specified file.

33779 **EXAMPLES**

33780 None.

33781 **APPLICATION USAGE**

33782 Application developers should check whether an option, such as `_POSIX_ADVISORY_INFO`, is  
 33783 supported prior to obtaining and using values for related variables such as  
 33784 `{POSIX_ALLOC_SIZE_MIN}`.

33785 **RATIONALE**

33786 The `fpathconf()` function was proposed immediately after the `sysconf()` function when it was  
 33787 realized that some configurable values may differ across file system, directory, or device  
 33788 boundaries.

33789 For example, `{NAME_MAX}` frequently changes between System V and BSD-based file systems;  
 33790 System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file  
 33791 systems, an application would be forced to limit all pathname components to 14 bytes, as this  
 33792 would be the value specified in `<limits.h>` on such a system.

33793 Therefore, various useful values can be queried on any pathname or file descriptor, assuming  
 33794 that appropriate privileges are in place.

33795 The value returned for the variable `{PATH_MAX}` indicates the longest relative pathname that  
 33796 could be given if the specified directory is the current working directory of the process. A  
 33797 process may not always be able to generate a name that long and use it if a subdirectory in the  
 33798 pathname crosses into a more restrictive file system. Note that implementations are allowed to  
 33799 accept pathnames longer than `{PATH_MAX}` bytes long, but are not allowed to return  
 33800 pathnames longer than this unless the user specifies a larger buffer using a function that  
 33801 provides a buffer size argument.

33802 The value returned for the variable `_POSIX_CHOWN_RESTRICTED` also applies to directories  
 33803 that do not have file systems mounted on them. The value may change when crossing a mount  
 33804 point, so applications that need to know should check for each directory. (An even easier check  
 33805 is to try the `chown()` function and look for an error in case it happens.)

33806 Unlike the values returned by `sysconf()`, the pathname-oriented variables are potentially more  
 33807 volatile and are not guaranteed to remain constant throughout the lifetime of the process. For  
 33808 example, in between two calls to `fpathconf()`, the file system in question may have been  
 33809 unmounted and remounted with different characteristics.

33810 Also note that most of the errors are optional. If one of the variables always has the same value  
 33811 on an implementation, the implementation need not look at `path` or `fildev` to return that value and  
 33812 is, therefore, not required to detect any of the errors except the meaning of `[EINVAL]` that  
 33813 indicates that the value of `name` is not valid for that variable, and the `[EOVERFLOW]` error that  
 33814 indicates the value to be returned is larger than `{LONG_MAX}`.

33815 If the value of any of the limits is unspecified (logically infinite), they will not be defined in  
 33816 `<limits.h>` and the `fpathconf()` and `fpathconf()` functions return `-1` without changing `errno`. This  
 33817 can be distinguished from the case of giving an unrecognized `name` argument because `errno` is set  
 33818 to `[EINVAL]` in this case.

33819 Since `-1` is a valid return value for the `fpathconf()` and `fpathconf()` functions, applications should  
 33820 set `errno` to zero before calling them and check `errno` only if the return value is `-1`.

33821 For the case of `{SYMLINK_MAX}`, since both `fpathconf()` and `open()` follow symbolic links, there  
 33822 is no way that `path` or `fildev` could refer to a symbolic link.

33823 It was the intention of IEEE Std 1003.1d-1999 that the following variables:

33824 {POSIX\_ALLOC\_SIZE\_MIN}  
 33825 {POSIX\_REC\_INCR\_XFER\_SIZE}  
 33826 {POSIX\_REC\_MAX\_XFER\_SIZE}  
 33827 {POSIX\_REC\_MIN\_XFER\_SIZE}  
 33828 {POSIX\_REC\_XFER\_ALIGN}

33829 only applied to regular files, but Note 10 also permits implementation of the advisory semantics  
 33830 on other file types unique to an implementation (for example, a character special device).

33831 The [EOVERFLOW] error for \_PC\_TIMESTAMP\_RESOLUTION cannot occur on POSIX-  
 33832 compliant file systems because POSIX requires a timestamp resolution no larger than one  
 33833 second. Even on 32-bit systems, this can be represented without overflow.

#### 33834 FUTURE DIRECTIONS

33835 None.

#### 33836 SEE ALSO

33837 *chown()*, *confstr()*, *sysconf()*

33838 XBD <limits.h>, <unistd.h>

33839 XCU *getconf*

#### 33840 CHANGE HISTORY

33841 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 33842 Issue 5

33843 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

33844 Large File Summit extensions are added.

#### 33845 Issue 6

33846 The following new requirements on POSIX implementations derive from alignment with the  
 33847 Single UNIX Specification:

- 33848 • The DESCRIPTION is updated to include {FILESIZEBITS}.
- 33849 • The [ELOOP] mandatory error condition is added.
- 33850 • A second [ENAMETOOLONG] is added as an optional error condition.

33851 The following changes were made to align with the IEEE P1003.1a draft standard:

- 33852 • The \_PC\_SYMLINK\_MAX entry is added to the table in the DESCRIPTION.

33853 The following *pathconf()* variables and their associated names are added for alignment with  
 33854 IEEE Std 1003.1d-1999:

33855 {POSIX\_ALLOC\_SIZE\_MIN}  
 33856 {POSIX\_REC\_INCR\_XFER\_SIZE}  
 33857 {POSIX\_REC\_MAX\_XFER\_SIZE}  
 33858 {POSIX\_REC\_MIN\_XFER\_SIZE}  
 33859 {POSIX\_REC\_XFER\_ALIGN}

33860 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth  
 33861 paragraph of the DESCRIPTION and removing shading and margin markers from the table.  
 33862 This change is needed since implementations are required to support all of these symbols.

33863 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/34 is applied, adding the table entry for  
 33864 POSIX2\_SYMLINKS in the DESCRIPTION.



33865 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/35 is applied, updating the  
33866 DESCRIPTION and RATIONALE sections to clarify behavior for the following variables:

33867 {POSIX\_ALLOC\_SIZE\_MIN}  
33868 {POSIX\_REC\_INCR\_XFER\_SIZE}  
33869 {POSIX\_REC\_MAX\_XFER\_SIZE}  
33870 {POSIX\_REC\_MIN\_XFER\_SIZE}  
33871 {POSIX\_REC\_XFER\_ALIGN}

33872 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/36 is applied, updating the RETURN  
33873 VALUE and APPLICATION USAGE sections to state that the results are unspecified if a variable  
33874 is dependent on an unsupported option, and advising application developers to check for  
33875 supported options prior to obtaining and using such values.

33876 **Issue 7**

33877 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.

33878 Changes are made related to support for finegrained timestamps.

33879 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
33880 pathname exists but is not a directory or a symbolic link to a directory.

33881 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0160 [256,428], XSH/TC1-2008/0161  
33882 [256,428], and XSH/TC1-2008/0162 [324] are applied.

33883 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0124 [651] and XSH/TC2-2008/0125  
33884 [651] are applied.

33885 **Issue 8**

33886 Austin Group Defect 687 is applied, adding \_POSIX\_FALLOC and \_PC\_FALLOC.

33887 Austin Group Defect 1122 is applied, adding {TEXTDOMAIN\_MAX}.

33888 **NAME**

33889 fpclassify — classify real floating type

33890 **SYNOPSIS**

33891 #include &lt;math.h&gt;

33892 int fpclassify(real-floating x);

33893 **DESCRIPTION**

33894 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
33895 conflict between the requirements described here and the ISO C standard is unintentional. This  
33896 volume of POSIX.1-2024 defers to the ISO C standard.

33897 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,  
33898 zero, or into another implementation-defined category. First, an argument represented in a  
33899 format wider than its semantic type is converted to its semantic type. Then classification is based  
33900 on the type of the argument.

33901 **RETURN VALUE**

33902 The *fpclassify()* macro shall return the value of the number classification macro appropriate to  
33903 the value of its argument.

33904 **ERRORS**

33905 No errors are defined.

33906 **EXAMPLES**

33907 None.

33908 **APPLICATION USAGE**

33909 None.

33910 **RATIONALE**

33911 None.

33912 **FUTURE DIRECTIONS**

33913 None.

33914 **SEE ALSO**33915 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

33916 XBD &lt;math.h&gt;

33917 **CHANGE HISTORY**

33918 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

33919 **NAME**

33920 asprintf, dprintf, fprintf, printf, snprintf, sprintf — print formatted output

33921 **SYNOPSIS**

33922 #include &lt;stdio.h&gt;

```

33923 CX int asprintf(char **restrict ptr, const char *restrict format, ...);
33924 int dprintf(int fildes, const char *restrict format, ...);
33925 int fprintf(FILE *restrict stream, const char *restrict format, ...);
33926 int printf(const char *restrict format, ...);
33927 int snprintf(char *restrict s, size_t n,
33928             const char *restrict format, ...);
33929 int sprintf(char *restrict s, const char *restrict format, ...);

```

33930 **DESCRIPTION**

33931 CX Except for *asprintf()*, *dprintf()*, and the behavior of the %lc conversion when passed a null wide  
 33932 character, the functionality described on this reference page is aligned with the ISO C standard.  
 33933 Any other conflict between the requirements described here and the ISO C standard is  
 33934 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard for all *fprintf()*,  
 33935 *printf()*, *snprintf()*, and *sprintf()* functionality except in relation to the %lc conversion when  
 33936 passed a null wide character.

33937 The *fprintf()* function shall place output on the named output *stream*. The *printf()* function shall  
 33938 place output on the standard output stream *stdout*. The *sprintf()* function shall place output  
 33939 followed by the null byte, '\0', in consecutive bytes starting at *\*s*; it is the user's responsibility  
 33940 to ensure that enough space is available.

33941 CX The *asprintf()* function shall be equivalent to *sprintf()*, except that the output string shall be  
 33942 written to dynamically allocated memory, allocated as if by a call to *malloc()*, of sufficient length  
 33943 to hold the resulting string, including a terminating null byte. If the call to *asprintf()* is  
 33944 successful, the address of this dynamically allocated string shall be stored in the location  
 33945 referenced by *ptr*.

33946 The *dprintf()* function shall be equivalent to the *fprintf()* function, except that *dprintf()* shall  
 33947 write output to the file associated with the file descriptor specified by the *fildes* argument rather  
 33948 than place output on a stream.

33949 The *snprintf()* function shall be equivalent to *sprintf()*, with the addition of the *n* argument  
 33950 which limits the number of bytes written to the buffer referred to by *s*. If *n* is zero, nothing shall  
 33951 be written and *s* may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be  
 33952 discarded instead of being written to the array, and a null byte is written at the end of the bytes  
 33953 actually written into the array.

33954 If copying takes place between objects that overlap as a result of a call to *sprintf()* or *snprintf()*,  
 33955 the results are undefined.

33956 Each of these functions converts, formats, and prints its arguments under control of the *format*.  
 33957 The application shall ensure that the format is a character string, beginning and ending in its  
 33958 initial shift state, if any. The format is composed of zero or more directives: *ordinary characters*,  
 33959 which are simply copied to the output stream, and *conversion specifications*, each of which shall  
 33960 result in the fetching of zero or more arguments. The results are undefined if there are  
 33961 insufficient arguments for the format. If the format is exhausted while arguments remain, the  
 33962 excess arguments shall be evaluated but are otherwise ignored.

33963 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 33964 to the next unused argument. In this case, the conversion specifier character % (see below) is  
 33965 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}],

33966 giving the position of the argument in the argument list. This feature provides for the definition  
 33967 of format strings that select arguments in an order appropriate to specific languages (see the  
 33968 EXAMPLES section).

33969 The *format* can contain either numbered argument conversion specifications (that is, those  
 33970 introduced by "%n\$" and optionally containing the "\*m\$" forms of field width and precision),  
 33971 or unnumbered argument conversion specifications (that is, those introduced by the % character  
 33972 and optionally containing the \* form of field width and precision), but not both. The only  
 33973 exception to this is that %% can be mixed with the "%n\$" form. The results of mixing numbered  
 33974 and unnumbered argument specifications in a format string are undefined. When numbered  
 33975 argument specifications are used, specifying the Nth argument requires that all the leading  
 33976 arguments, from the first to the (N-1)th, are specified in the format string.

33977 In format strings containing the "%n\$" form of conversion specification, numbered arguments  
 33978 in the argument list can be referenced from the format string as many times as required.

33979 In format strings containing the % form of conversion specification, each conversion specification  
 33980 uses the first unused argument in the argument list.

33981 CX All forms of the *fprintf()* functions allow for the insertion of a language-dependent radix  
 33982 character in the output string. The radix character is defined in the current locale (category  
 33983 LC\_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the  
 33984 radix character shall default to a <period> ('.').

33985 CX Each conversion specification is introduced by the '%' character or by the character sequence  
 33986 "%n\$", after which the following appear in sequence:

- 33987 • Zero or more *flags* (in any order), which modify the meaning of the conversion  
 33988 specification.
- 33989 • An optional minimum *field width*. If the converted value has fewer bytes than the field  
 33990 width, it shall be padded with <space> characters by default on the left; it shall be padded  
 33991 on the right if the left-adjustment flag ('-'), described below, is given to the field width.  
 33992 CX The field width takes the form of an <asterisk> ('\*'), or in conversion specifications  
 33993 introduced by "%n\$" the "\*m\$" string, described below, or a decimal integer.
- 33994 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,  
 33995 x, and X conversion specifiers; the number of digits to appear after the radix character for  
 33996 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for  
 33997 the g and G conversion specifiers; or the maximum number of bytes to be printed from a  
 33998 XSI string in the s and S conversion specifiers. The precision takes the form of a <period>  
 33999 CX ('.') followed either by an <asterisk> ('\*'), or in conversion specifications introduced  
 34000 by "%n\$" the "\*m\$" string, described below, or an optional decimal digit string, where a  
 34001 null digit string is treated as zero. If a precision appears with any other conversion  
 34002 specifier, the behavior is undefined.
- 34003 • An optional length modifier that specifies the size of the argument.
- 34004 • A *conversion specifier* character that indicates the type of conversion to be applied.

34005 A field width, or precision, or both, may be indicated by an <asterisk> ('\*'). In this case an  
 34006 argument of type **int** supplies the field width or precision. Applications shall ensure that  
 34007 arguments specifying field width, or precision, or both appear in that order before the argument,  
 34008 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field  
 34009 CX width. A negative precision is taken as if the precision were omitted. In format strings  
 34010 containing conversion specifications introduced by "%n\$", in addition to being indicated by the  
 34011 decimal digit string, a field width may be indicated by the sequence "\*m\$" and precision by the

```

34012 sequence ".*m$", where m is a decimal integer in the range [1,{NL_ARGMAX}] giving the
34013 position in the argument list (after the format argument) of an integer argument containing the
34014 field width or precision, for example:
34015 printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);

```

34016 The flag characters and their meanings are:

- 34017 CX ' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d,  
34018 %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For  
34019 other conversions the behavior is undefined. The non-monetary grouping character is  
34020 used.
- 34021 - The result of the conversion shall be left-justified within the field. The conversion is  
34022 right-justified if this flag is not specified.
- 34023 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The  
34024 conversion shall begin with a sign only when a negative value is converted if this flag is  
34025 not specified.
- 34026 <space> If the first character of a signed conversion is not a sign or if a signed conversion results  
34027 in no characters, a <space> shall be prefixed to the result. This means that if the  
34028 <space> and '+' flags both appear, the <space> flag shall be ignored.
- 34029 # Specifies that the value is to be converted to an alternative form. For o conversion, it  
34030 shall increase the precision, if and only if necessary, to force the first digit of the result  
34031 to be a zero (if the value and precision are both 0, a single 0 is printed). For x or X  
34032 conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,  
34033 E, f, F, g, and G conversion specifiers, the result shall always contain a radix character,  
34034 even if no digits follow the radix character. Without this flag, a radix character appears  
34035 in the result of these conversions only if a digit follows it. For g and G conversion  
34036 specifiers, trailing zeros shall *not* be removed from the result as they normally are. For  
34037 other conversion specifiers, the behavior is undefined.
- 34038 0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros  
34039 (following any indication of sign or base) are used to pad to the field width rather than  
34040 performing space padding, except when converting an infinity or NaN. If the '0' and  
34041 '-' flags both appear, the '0' flag is ignored. For d, i, o, u, x, and X conversion  
34042 CX specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and  
34043 <apostrophe> flags both appear, the grouping characters are inserted before zero  
34044 padding. For other conversions, the behavior is undefined.

34045 The length modifiers and their meanings are:

- 34046 hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char**  
34047 or **unsigned char** argument (the argument will have been promoted according to the  
34048 integer promotions, but its value shall be converted to **signed char** or **unsigned char**  
34049 before printing); or that a following n conversion specifier applies to a pointer to a  
34050 **signed char** argument.
- 34051 h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or  
34052 **unsigned short** argument (the argument will have been promoted according to the  
34053 integer promotions, but its value shall be converted to **short** or **unsigned short** before  
34054 printing); or that a following n conversion specifier applies to a pointer to a **short**  
34055 argument.

- 34056 1 (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long** or  
 34057 **unsigned long** argument; that a following `n` conversion specifier applies to a pointer to  
 34058 a **long** argument; that a following `c` conversion specifier applies to a **wint\_t** argument;  
 34059 that a following `s` conversion specifier applies to a pointer to a **wchar\_t** argument; or  
 34060 has no effect on a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier.
- 34061 1.1 (ell-ell)  
 34062 Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long long**  
 34063 or **unsigned long long** argument; or that a following `n` conversion specifier applies to a  
 34064 pointer to a **long long** argument.
- 34065 `j` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to an **intmax\_t**  
 34066 or **uintmax\_t** argument; or that a following `n` conversion specifier applies to a pointer  
 34067 to an **intmax\_t** argument.
- 34068 `z` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **size\_t** or the  
 34069 corresponding signed integer type argument; or that a following `n` conversion specifier  
 34070 applies to a pointer to a signed integer type corresponding to a **size\_t** argument.
- 34071 `t` Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **ptrdiff\_t** or  
 34072 the corresponding **unsigned** type argument; or that a following `n` conversion specifier  
 34073 applies to a pointer to a **ptrdiff\_t** argument.
- 34074 `L` Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to a **long**  
 34075 **double** argument.
- 34076 If a length modifier appears with any conversion specifier other than as specified above, the  
 34077 behavior is undefined.
- 34078 The conversion specifiers and their meanings are:
- 34079 `d`, `i` The **int** argument shall be converted to a signed decimal in the style "`[-] dddd`". The  
 34080 precision specifies the minimum number of digits to appear; if the value being  
 34081 converted can be represented in fewer digits, it shall be expanded with leading zeros.  
 34082 The default precision is 1. The result of converting zero with an explicit precision of  
 34083 zero shall be no characters.
- 34084 `o` The **unsigned** argument shall be converted to unsigned octal format in the style  
 34085 "`ddd`". The precision specifies the minimum number of digits to appear; if the value  
 34086 being converted can be represented in fewer digits, it shall be expanded with leading  
 34087 zeros. The default precision is 1. The result of converting zero with an explicit precision  
 34088 of zero shall be no characters.
- 34089 `u` The **unsigned** argument shall be converted to unsigned decimal format in the style  
 34090 "`ddd`". The precision specifies the minimum number of digits to appear; if the value  
 34091 being converted can be represented in fewer digits, it shall be expanded with leading  
 34092 zeros. The default precision is 1. The result of converting zero with an explicit precision  
 34093 of zero shall be no characters.
- 34094 `x` The **unsigned** argument shall be converted to unsigned hexadecimal format in the style  
 34095 "`ddd`"; the letters "`abcdef`" are used. The precision specifies the minimum number  
 34096 of digits to appear; if the value being converted can be represented in fewer digits, it  
 34097 shall be expanded with leading zeros. The default precision is 1. The result of  
 34098 converting zero with an explicit precision of zero shall be no characters.
- 34099 `X` Equivalent to the `x` conversion specifier, except that letters "`ABCDEF`" are used instead  
 34100 of "`abcdef`".

34101	f, F	The <b>double</b> argument shall be converted to decimal notation in the style	
34102		" <code>[-]ddd.ddd</code> ", where the number of digits after the radix character is equal to the	
34103		precision specification. If the precision is missing, it shall be taken as 6; if the precision	
34104		is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix	
34105		character appears, at least one digit appears before it. The low-order digit shall be	
34106		rounded in an implementation-defined manner.	
34107		A <b>double</b> argument representing an infinity shall be converted in one of the styles	
34108		" <code>[-]inf</code> " or " <code>[-]infinity</code> "; which style is implementation-defined. A <b>double</b>	
34109		argument representing a NaN shall be converted in one of the styles " <code>[-]nan(<i>n-char-sequence</i>)</code> "	
34110		or " <code>[-]nan</code> "; which style, and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The F	
34111		conversion specifier produces "INF",	
34112		"INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.	
34113	e, E	The <b>double</b> argument shall be converted in the style " <code>[-]d.ddde±dd</code> ", where there is	
34114		one digit before the radix character (which is non-zero if the argument is non-zero) and	
34115		the number of digits after it is equal to the precision; if the precision is missing, it shall	
34116		be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall	
34117		appear. The low-order digit shall be rounded in an implementation-defined manner.	
34118		The E conversion specifier shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the	
34119	value is zero, the exponent shall be zero.		
34120		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of	
34121		an f or F conversion specifier.	
34122			
34123		g, G	The <b>double</b> argument representing a floating-point number shall be converted in the
34124			style f or e (or in the style F or E in the case of a G conversion specifier), depending on
34125			the value converted and the precision. Let P equal the precision if non-zero, 6 if the
34126	precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E		
34127	would have an exponent of X:		
34128	— If $P > X \geq -4$ , the conversion shall be with style f (or F) and precision $P - (X + 1)$ .		
34129	— Otherwise, the conversion shall be with style e (or E) and precision $P - 1$ .		
34130		Finally, unless the '#' flag is used, any trailing zeros shall be removed from the	
34131		fractional portion of the result and the decimal-point character shall be removed if there	
34132		is no fractional portion remaining.	
34133		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of	
34134		an f or F conversion specifier.	
34135	a, A	A <b>double</b> argument representing a floating-point number shall be converted in the	
34136		style " <code>[-]0xh.hhhhp±d</code> ", where there is one hexadecimal digit (which shall be non-	
34137		zero if the argument is a normalized floating-point number and is otherwise	
34138		unspecified) before the decimal-point character and the number of hexadecimal digits	
34139		after it is equal to the precision; if the precision is missing and FLT_RADIX is a power	
34140		of 2, then the precision shall be sufficient for an exact representation of the value; if the	
34141		precision is missing and FLT_RADIX is not a power of 2, then the precision shall be	
34142		sufficient to distinguish values of type <b>double</b> , except that trailing zeros may be	
34143		omitted; if the precision is zero and the '#' flag is not specified, no decimal-point	
34144		character shall appear. The letters "abcdef" shall be used for a conversion and the	
34145		letters "ABCDEF" for A conversion. The A conversion specifier produces a number with	
34146		'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one	
34147		digit, and only as many more digits as necessary to represent the decimal exponent of	

34148		2. If the value is zero, the exponent shall be zero.
34149		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
34150		
34151	c	The <b>int</b> argument shall be converted to an <b>unsigned char</b> , and the resulting byte shall be written.
34152		
34153	CX	If an <code>l</code> (ell) qualifier is present, the <b>wint_t</b> argument shall be converted to a multi-byte sequence as if by a call to <code>wcrtomb()</code> with a pointer to storage of at least <code>MB_CUR_MAX</code> bytes, the <b>wint_t</b> argument converted to <b>wchar_t</b> , and an initial shift state, and the resulting byte(s) written.
34154		
34155		
34156		
34157	s	The argument shall be a pointer to an array of <b>char</b> . Bytes from the array shall be written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes shall be written. If the precision is not specified or is greater than the size of the array, the application shall ensure that the array contains a null byte.
34158		
34159		
34160		
34161		
34162		If an <code>l</code> (ell) qualifier is present, the argument shall be a pointer to an array of type <b>wchar_t</b> . Wide characters from the array shall be converted to characters (each as if by a call to the <code>wcrtomb()</code> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial character be written.
34163		
34164		
34165		
34166		
34167		
34168		
34169		
34170		
34171		
34172		
34173		
34174	p	The argument shall be a pointer to <b>void</b> . The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.
34175		
34176	n	The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the <code>fprintf()</code> functions. No argument is converted.
34177		
34178		
34179	XSI	<b>C</b> Equivalent to <code>lc</code> .
34180	XSI	<b>S</b> Equivalent to <code>ls</code> .
34181	%	Write a <code>'%'</code> character; no argument shall be converted. The application shall ensure that the complete conversion specification is <code>%%</code> .
34182		
34183		If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.
34184		
34185		
34186		In no case shall a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <code>fprintf()</code> and <code>printf()</code> are printed as if <code>fputc()</code> had been called.
34187		
34188		
34189		For the <code>a</code> and <code>A</code> conversion specifiers, if <code>FLT_RADIX</code> is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.
34190		
34191		For <code>a</code> and <code>A</code> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in
34192		



34193 hexadecimal floating style with the given precision, with the extra stipulation that the error  
34194 should have a correct sign for the current rounding direction.

34195 For the *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers, if the number of significant decimal digits is at  
34196 most `DECIMAL_DIG`, then the result should be correctly rounded. If the number of significant  
34197 decimal digits is more than `DECIMAL_DIG` but the source value is exactly representable with  
34198 `DECIMAL_DIG` digits, then the result should be an exact representation with trailing zeros.  
34199 Otherwise, the source value is bounded by two adjacent decimal strings  $L < U$ , both having  
34200 `DECIMAL_DIG` significant digits; the value of the resultant decimal string  $D$  should satisfy  $L \leq$   
34201  $D \leq U$ , with the extra stipulation that the error should have a correct sign for the current  
34202 rounding direction.

34203 CX The last data modification and last file status change timestamps of the file shall be marked for  
34204 update:

- 34205 1. Between the call to a successful execution of `fprintf()` or `printf()` and the next successful  
34206 completion of a call to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`
- 34207 2. Upon successful completion of a call to `dprintf()`

### 34208 RETURN VALUE

34209 CX Upon successful completion, the `dprintf()`, `fprintf()`, and `printf()` functions shall return the  
34210 number of bytes transmitted.

34211 CX Upon successful completion, the `asprintf()` function shall return the number of bytes written to  
34212 the allocated string stored in the location referenced by `ptr`, excluding the terminating null byte.

34213 Upon successful completion, the `sprintf()` function shall return the number of bytes written to `s`,  
34214 excluding the terminating null byte.

34215 Upon successful completion, the `snprintf()` function shall return the number of bytes that would  
34216 be written to `s` had `n` been sufficiently large excluding the terminating null byte.

34217 CX If an error was encountered, these functions shall return a negative value and set `errno` to  
34218 indicate the error. For `asprintf()`, if memory allocation was not possible, or if some other error  
34219 occurs, the function shall return a negative value, and the contents of the location referenced by  
34220 `ptr` are undefined, but shall not refer to allocated memory.

34221 If the value of `n` is zero on a call to `snprintf()`, nothing shall be written, the number of bytes that  
34222 would have been written had `n` been sufficiently large excluding the terminating null shall be  
34223 returned, and `s` may be a null pointer.

### 34224 ERRORS

34225 CX For the conditions under which `dprintf()`, `fprintf()`, and `printf()` fail and may fail, refer to `fputc()`  
34226 or `fputwc()`.

34227 In addition, all forms of `fprintf()` shall fail if:

34228 CX [EILSEQ] A wide-character code that does not correspond to a valid character has been  
34229 detected.

34230 CX [EOVERFLOW] The value to be returned is greater than `{INT_MAX}`.

34231 CX The `asprintf()` function shall fail if:

34232 [ENOMEM] Insufficient storage space is available.

34233 The `dprintf()` function may fail if:

34234 [EBADF] The *fildest* argument is not a valid file descriptor.

34235 CX The *dprintf()*, *fprintf()*, and *printf()* functions may fail if:

34236 CX [ENOMEM] Insufficient storage space is available.

### 34237 EXAMPLES

#### 34238 Printing Language-Independent Date and Time

34239 The following statement can be used to print date and time using a language-independent  
34240 format:

```
34241 printf(format, weekday, month, day, hour, min);
```

34242 For American usage, *format* could be a pointer to the following string:

```
34243 "%s, %s %d, %d:%.2d\n"
```

34244 This example would produce the following message:

```
34245 Sunday, July 3, 10:02
```

34246 For German usage, *format* could be a pointer to the following string:

```
34247 "%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

34248 This definition of *format* would produce the following message:

```
34249 Sonntag, 3. Juli, 10:02
```

#### 34250 Printing File Information

34251 The following example prints information about the type, permissions, and number of links of a  
34252 specific file in a directory.

34253 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined  
34254 *strperm()* function shall return a string similar to the one at the beginning of the output for the  
34255 following command:

```
34256 ls -l
```

34257 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*  
34258 function shall return a **passwd** structure from which the name of the user is extracted. If the user  
34259 name is not found, the program instead prints out the numeric value of the user ID.

34260 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar  
34261 to *getpwuid()* except that it shall return group information based on the group number. Once  
34262 again, if the group is not found, the program prints the numeric value of the group for the entry.

34263 The final call to *printf()* prints the size of the file.

```
34264 #include <stdio.h>
34265 #include <sys/types.h>
34266 #include <pwd.h>
34267 #include <grp.h>

34268 char *strperm (mode_t);
34269 ...
34270 struct stat statbuf;
34271 struct passwd *pwd;
```

```

34272     struct group *grp;
34273     ...
34274     printf("%10.10s", strperm (statbuf.st_mode));
34275     printf("%4d", statbuf.st_nlink);

34276     if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
34277         printf(" %-8.8s", pwd->pw_name);
34278     else
34279         printf(" %-8ld", (long) statbuf.st_uid);

34280     if ((grp = getgrgid(statbuf.st_gid)) != NULL)
34281         printf(" %-8.8s", grp->gr_name);
34282     else
34283         printf(" %-8ld", (long) statbuf.st_gid);

34284     printf("%9jd", (intmax_t) statbuf.st_size);
34285     ...

```

### 34286 **Printing a Localized Date String**

34287 The following example gets a localized date string. The *nl\_langinfo()* function shall return the  
34288 localized date string, which specifies the order and layout of the date. The *strftime()* function  
34289 takes this information and, using the **tm** structure for values, places the date and time  
34290 information into *datestring*. The *printf()* function then outputs *datestring* and the name of the  
34291 entry.

```

34292     #include <stdio.h>
34293     #include <time.h>
34294     #include <langinfo.h>
34295     ...
34296     struct dirent *dp;
34297     struct tm *tm;
34298     char datestring[256];
34299     ...
34300     strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);

34301     printf(" %s %s\n", datestring, dp->d_name);
34302     ...

```

### 34303 **Printing Error Information**

34304 The following example uses *fprintf()* to write error information to standard error.

34305 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If  
34306 the file already exists, this is an error, as indicated by the **O\_EXCL** flag on the *open()* function. If  
34307 the call fails, the program assumes that someone else is updating the password file, and the  
34308 program exits.

34309 The next group of calls saves a new password file as the current password file by creating a link  
34310 between **LOCKFILE** and the new password file **PASSWDFILE**.

```

34311     #include <sys/types.h>
34312     #include <sys/stat.h>
34313     #include <fcntl.h>
34314     #include <stdio.h>
34315     #include <stdlib.h>

```

```

34316     #include <unistd.h>
34317     #include <string.h>
34318     #include <errno.h>

34319     #define LOCKFILE "/etc/ptmp"
34320     #define PASSWDFILE "/etc/passwd"
34321     ...
34322     int pfd;
34323     ...
34324     if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
34325         S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
34326     {
34327         fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
34328         exit(1);
34329     }
34330     ...
34331     if (link(LOCKFILE, PASSWDFILE) == -1) {
34332         fprintf(stderr, "Link error: %s\n", strerror(errno));
34333         exit(1);
34334     }
34335     ...

```

### 34336 Printing Usage Information

34337 The following example checks to make sure the program has the necessary arguments, and uses  
34338 *fprintf()* to print usage information if the expected number of arguments is not present.

```

34339     #include <stdio.h>
34340     #include <stdlib.h>
34341     ...
34342     char *Options = "hdbt1";
34343     ...
34344     if (argc < 2) {
34345         fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
34346     }
34347     ...

```

### 34348 Formatting a Decimal String

34349 The following example prints a key and data pair on *stdout*. Note use of the <asterisk> ('\*') in  
34350 the format string; this ensures the correct number of decimal places for the element based on the  
34351 number of elements requested.

```

34352     #include <stdio.h>
34353     ...
34354     long i;
34355     char *keyst;
34356     int elementlen, len;
34357     ...
34358     while (len < elementlen) {
34359         ...
34360         printf("%s Element%0*ld\n", keyst, elementlen, i);
34361         ...

```

```
34362     }
```

### 34363 **Creating a Pathname**

34364 The following example creates a pathname using information from a previous *getpwnam()*  
34365 function that returned the password database entry of the user.

```
34366     #include <stdint.h>
34367     #include <stdio.h>
34368     #include <stdlib.h>
34369     #include <string.h>
34370     #include <sys/types.h>
34371     #include <unistd.h>
34372     ...
34373     char *pathname;
34374     struct passwd *pw;
34375     size_t len;
34376     ...
34377     // digits required for pid_t is number of bits times
34378     // log2(10) = approx 10/33
34379     len = strlen(pw->pw_dir) + 1 + 1+(sizeof(pid_t)*80+32)/33 +
34380         sizeof ".out";
34381     pathname = malloc(len);
34382     if (pathname != NULL)
34383     {
34384         snprintf(pathname, len, "%s/%jd.out", pw->pw_dir,
34385             (intmax_t) getpid());
34386         ...
34387     }
```

### 34388 **Reporting an Event**

34389 The following example loops until an event has timed out. The *pause()* function waits forever  
34390 unless it receives a signal. The *fprintf()* statement should never occur due to the possible return  
34391 values of *pause()*.

```
34392     #include <stdio.h>
34393     #include <unistd.h>
34394     #include <string.h>
34395     #include <errno.h>
34396     ...
34397     while (!event_complete) {
34398         ...
34399         if (pause() != -1 || errno != EINTR)
34400             fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
34401     }
34402     ...
```

34403 **Printing Monetary Information**

34404 The following example uses *strfmon()* to convert a number and store it as a formatted monetary  
 34405 string named *convbuf*. If the first number is printed, the program prints the format and the  
 34406 description; otherwise, it just prints the number.

```

34407 #include <monetary.h>
34408 #include <stdio.h>
34409 ...
34410 struct tblfmt {
34411     char *format;
34412     char *description;
34413 };
34414
34415 struct tblfmt table[] = {
34416     { "%n", "default formatting" },
34417     { "%11n", "right align within an 11 character field" },
34418     { "%#5n", "aligned columns for values up to 99999" },
34419     { "%=*#5n", "specify a fill character" },
34420     { "%=0#5n", "fill characters do not use grouping" },
34421     { "%^#5n", "disable the grouping separator" },
34422     { "%^#5.0n", "round off to whole units" },
34423     { "%^#5.4n", "increase the precision" },
34424     { "%(#5n", "use an alternative pos/neg style" },
34425     { "%!(#5n", "disable the currency symbol" },
34426 };
34427 ...
34428 float input[3];
34429 int i, j;
34430 char convbuf[100];
34431 ...
34432 strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
34433
34434 if (j == 0) {
34435     printf("%s%s%s\n", table[i].format,
34436           convbuf, table[i].description);
34437 }
34438 else {
34439     printf("%s\n", convbuf);
34440 }
34441 ...
  
```

34440 **Printing Wide Characters**

34441 The following example prints a series of wide characters. Suppose that "L`@`" expands to three  
 34442 bytes:

```

34443 wchar_t wz [3] = L"@@";           // Zero-terminated
34444 wchar_t wn [3] = L"@@";           // Unterminated
34445
34446 fprintf (stdout, "%ls", wz);       // Outputs 6 bytes
34447 fprintf (stdout, "%ls", wn);       // Undefined because wn has no terminator
34448 fprintf (stdout, "%4ls", wz);       // Outputs 3 bytes
34449 fprintf (stdout, "%4ls", wn);       // Outputs 3 bytes; no terminator needed
34450 fprintf (stdout, "%9ls", wz);       // Outputs 6 bytes
  
```

```

34450 fprintf (stdout, "%9ls", wn); // Outputs 9 bytes; no terminator needed
34451 fprintf (stdout, "%10ls", wz); // Outputs 6 bytes
34452 fprintf (stdout, "%10ls", wn); // Undefined because wn has no terminator

```

34453 In the last line of the example, after processing three characters, nine bytes have been output.  
 34454 The fourth character must then be examined to determine whether it converts to one byte or  
 34455 more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth  
 34456 character in the array, the behavior is undefined.

#### 34457 APPLICATION USAGE

34458 If the application calling *fprintf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
 34459 the **<wchar.h>** header to have these objects defined.

34460 The space allocated by a successful call to *asprintf()* should be subsequently freed by a call to  
 34461 *free()*.

#### 34462 RATIONALE

34463 If an implementation detects that there are insufficient arguments for the format, it is  
 34464 recommended that the function should fail and report an [EINVAL] error.

34465 The behavior specified for the `%lc` conversion differs slightly from the specification in the ISO C  
 34466 standard, in that printing the null wide character produces a null byte instead of 0 bytes of  
 34467 output as would be required by a strict reading of the ISO C standard's direction to behave as if  
 34468 applying the `%ls` specifier to a **wchar\_t** array whose first element is the null wide character.  
 34469 Requiring a multi-byte output for every possible wide character, including the null character,  
 34470 matches historical practice, and provides consistency with `%c` in *fprintf()* and with both `%c` and  
 34471 `%lc` in *fwprintf()*. It is anticipated that a future edition of the ISO C standard will change to  
 34472 match the behavior specified here.

#### 34473 FUTURE DIRECTIONS

34474 None.

#### 34475 SEE ALSO

34476 [Section 2.5](#) (on page 521), *fputc()*, *fscanf()*, *setlocale()*, *strfmon()*, *strlcat()*, *wcrtomb()*, *wcslcat()*

34477 [XBD Chapter 7](#) (on page 127), **<inttypes.h>**, **<stdio.h>**, **<wchar.h>**

#### 34478 CHANGE HISTORY

34479 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 34480 Issue 5

34481 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier can  
 34482 now be used with `c` and `s` conversion specifiers.

34483 The *snprintf()* function is new in Issue 5.

#### 34484 Issue 6

34485 Extensions beyond the ISO C standard are marked.

34486 The normative text is updated to avoid use of the term “must” for application requirements.

34487 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 34488 • The prototypes for *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* are updated, and the XSI  
 34489 shading is removed from *snprintf()*.

- 34490 • The description of *snprintf()* is aligned with the ISO C standard. Note that this supersedes  
 34491 the *snprintf()* description in The Open Group Base Resolution bwg98-006, which changed  
 34492 the behavior from Issue 5.

- 34493           • The DESCRIPTION is updated.
- 34494           The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.
- 34495
- 34496           ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.
- 34497           An example of printing wide characters is added.
- 34498   **Issue 7**
- 34499           Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.
- 34500
- 34501           Austin Group Interpretation 1003.1-2001 #170 is applied.
- 34502           ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of g and G.
- 34503
- 34504           SD5-XSH-ERN-174 is applied.
- 34505           The *dprintf()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.
- 34506
- 34507           Functionality relating to the %n\$ form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.
- 34508
- 34509           Changes are made related to support for finegrained timestamps.
- 34510           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0163 [302], XSH/TC1-2008/0164 [316], XSH/TC1-2008/0165 [316], XSH/TC1-2008/0166 [451,291], and XSH/TC1-2008/0167 [14] are applied.
- 34511
- 34512
- 34513           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0126 [894], XSH/TC2-2008/0127 [557], and XSH/TC2-2008/0128 [936] are applied.
- 34514
- 34515   **Issue 8**
- 34516           Austin Group Defect 986 is applied, adding *strlcat()* and *wcslcat()* to the SEE ALSO section.
- 34517           Austin Group Defect 1020 is applied, clarifying that the *snprintf()* argument *n* limits the number of bytes written to *s*; it is not necessarily the same as the size of *s*.
- 34518
- 34519           Austin Group Defect 1021 is applied, changing “output error” to “error” in the RETURN VALUE section.
- 34520
- 34521           Austin Group Defect 1137 is applied, clarifying the use of “%n\$” and “\*m\$” in conversion specifications.
- 34522
- 34523           Austin Group Defect 1205 is applied, changing the description of the % conversion specifier.
- 34524           Austin Group Defect 1219 is applied, removing the *snprintf()*-specific [Eoverflow] error.
- 34525           Austin Group Defect 1496 is applied, adding the *asprintf()* function.
- 34526           Austin Group Defect 1562 is applied, clarifying that it is the application’s responsibility to ensure that the format is a character string, beginning and ending in its initial shift state, if any.
- 34527
- 34528           Austin Group Defect 1647 is applied, changing the description of the c conversion specifier and updating the statement that this volume of POSIX.1-2024 defers to the ISO C standard so that it excludes the %lc conversion when passed a null wide character.
- 34529
- 34530



34531 **NAME**

34532           fputc — put a byte on a stream

34533 **SYNOPSIS**

34534           #include &lt;stdio.h&gt;

34535           int fputc(int *c*, FILE \**stream*);34536 **DESCRIPTION**34537 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
34538 conflict between the requirements described here and the ISO C standard is unintentional. This  
34539 volume of POSIX.1-2024 defers to the ISO C standard.34540       The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the  
34541 output stream pointed to by *stream*, at the position indicated by the associated file-position  
34542 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file  
34543 cannot support positioning requests, or if the stream was opened with append mode, the byte  
34544 shall be appended to the output stream.34545 CX       The last data modification and last file status change timestamps of the file shall be marked for  
34546 update between the successful execution of *fputc()* and the next successful completion of a call  
34547 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.34548 **RETURN VALUE**34549       Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall  
34550 CX       return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the  
34551 error.34552 **ERRORS**34553       The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be  
34554 flushed, and:34555 CX       [EAGAIN]       The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and  
34556                       the thread would be delayed in the write operation.34557 CX       [EBADF]       The file descriptor underlying *stream* is not a valid file descriptor open for  
34558                       writing.

34559 CX       [EFBIG]       An attempt was made to write to a file that exceeds the maximum file size.

34560 CX       [EFBIG]       An attempt was made to write to a file that exceeds the file size limit of the  
34561                       process.  
34562 XSI       A SIGXFSZ signal shall also be generated for the thread.34563 CX       [EFBIG]       The file is a regular file and an attempt was made to write at or beyond the  
34564                       offset maximum.34565 CX       [EINTR]       The write operation was terminated due to the receipt of a signal, and no data  
34566                       was transferred.34567 CX       [EIO]        A physical I/O error has occurred, or the process is a member of a background  
34568                       process group attempting to write to its controlling terminal, TOSTOP is set,  
34569                       the calling thread is not blocking SIGTTOU, the process is not ignoring  
34570                       SIGTTOU, and the process group of the process is orphaned. This error may  
34571                       also be returned under implementation-defined conditions.

34572 CX       [ENOSPC]       There was no free space remaining on the device containing the file.

34573 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
 34574 any process. A SIGPIPE signal shall also be sent to the thread.

34575 The *fputc()* function may fail if:

34576 CX [ENOMEM] Insufficient storage space is available.

34577 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
 34578 capabilities of the device.

#### 34579 EXAMPLES

34580 None.

#### 34581 APPLICATION USAGE

34582 None.

#### 34583 RATIONALE

34584 None.

#### 34585 FUTURE DIRECTIONS

34586 None.

#### 34587 SEE ALSO

34588 [Section 2.5](#) (on page 521), *ferror()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*

34589 XBD [<stdio.h>](#)

#### 34590 CHANGE HISTORY

34591 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 34592 Issue 5

34593 Large File Summit extensions are added.

#### 34594 Issue 6

34595 Extensions beyond the ISO C standard are marked.

34596 The following new requirements on POSIX implementations derive from alignment with the  
 34597 Single UNIX Specification:

- 34598 • The [EIO] and [EFBIG] mandatory error conditions are added.
- 34599 • The [ENOMEM] and [ENXIO] optional error conditions are added.

34600 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/37 is applied, updating the [EAGAIN]  
 34601 error in the ERRORS section from “the process would be delayed” to “the thread would be  
 34602 delayed”.

#### 34603 Issue 7

34604 Changes are made related to support for finegrained timestamps.

34605 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0168 [79] and XSH/TC1-2008/0169  
 34606 [14] are applied.

#### 34607 Issue 8

34608 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

34609 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
 34610 relating to the file size limit for the process.

34611 **NAME**

34612 fputs — put a string on a stream

34613 **SYNOPSIS**

34614 #include &lt;stdio.h&gt;

34615 int fputs(const char \*restrict s, FILE \*restrict stream);

34616 **DESCRIPTION**

34617 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 34618 conflict between the requirements described here and the ISO C standard is unintentional. This  
 34619 volume of POSIX.1-2024 defers to the ISO C standard.

34620 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed  
 34621 to by *stream*. The terminating null byte shall not be written.

34622 CX The last data modification and last file status change timestamps of the file shall be marked for  
 34623 update between the successful execution of *fputs()* and the next successful completion of a call  
 34624 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

34625 **RETURN VALUE**

34626 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall  
 34627 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

34628 **ERRORS**34629 Refer to *fputc()*.34630 **EXAMPLES**34631 **Printing to Standard Output**

34632 The following example gets the current time, converts it to a string using *localtime()* and  
 34633 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 34634 an event for which it is waiting.

```
34635 #include <time.h>
34636 #include <stdio.h>
34637 ...
34638 time_t now;
34639 int minutes_to_event;
34640 ...
34641 time(&now);
34642 printf("The time is ");
34643 fputs(asctime(localtime(&now)), stdout);
34644 printf("There are still %d minutes to the event.\n",
34645     minutes_to_event);
34646 ...
```

34647 **APPLICATION USAGE**34648 The *puts()* function appends a <newline> while *fputs()* does not.

34649 This volume of POSIX.1-2024 requires that successful completion simply return a non-negative  
 34650 integer. There are at least three known different implementation conventions for this  
 34651 requirement:

- 34652 • Return a constant value.

- 34653 • Return the last character written.
- 34654 • Return the number of bytes written. Note that this implementation convention cannot be
- 34655 adhered to for strings longer than {INT\_MAX} bytes as the value would not be
- 34656 representable in the return type of the function. For backwards-compatibility,
- 34657 implementations can return the number of bytes for strings of up to {INT\_MAX} bytes, and
- 34658 return {INT\_MAX} for all longer strings.

#### 34659 RATIONALE

34660 The *fputs()* function is one whose source code was specified in the referenced *The C Programming*  
34661 *Language*. In the original edition, the function had no defined return value, yet many practical  
34662 implementations would, as a side-effect, return the value of the last character written as that was  
34663 the value remaining in the accumulator used as a return value. In the second edition of the book,  
34664 either the fixed value 0 or EOF would be returned depending upon the return value of *ferror()*;  
34665 however, for compatibility with extant implementations, several implementations would, upon  
34666 success, return a positive value representing the last byte written.

#### 34667 FUTURE DIRECTIONS

34668 None.

#### 34669 SEE ALSO

34670 [Section 2.5](#) (on page 521), *fopen()*, *putc()*, *puts()*

34671 XBD [<stdio.h>](#)

#### 34672 CHANGE HISTORY

34673 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 34674 Issue 6

34675 Extensions beyond the ISO C standard are marked.

34676 The *fputs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

#### 34677 Issue 7

34678 Changes are made related to support for finegrained timestamps.

34679 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0170 [174,412], XSH/TC1-2008/0171  
34680 [412], and XSH/TC1-2008/0172 [14] are applied.

34681 **NAME**34682 `fputwc` — put a wide-character code on a stream34683 **SYNOPSIS**34684 `#include <stdio.h>`34685 `#include <wchar.h>`34686 `wint_t fputwc(wchar_t wc, FILE *stream);`34687 **DESCRIPTION**

34688 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 34689 conflict between the requirements described here and the ISO C standard is unintentional. This  
 34690 volume of POSIX.1-2024 defers to the ISO C standard.

34691 The `fputwc()` function shall write the character corresponding to the wide-character code `wc` to  
 34692 the output stream pointed to by `stream`, at the position indicated by the associated file-position  
 34693 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot  
 34694 support positioning requests, or if the stream was opened with append mode, the character is  
 34695 appended to the output stream. If an error occurs while writing the character, the shift state of  
 34696 the output file is left in an undefined state.

34697 CX The last data modification and last file status change timestamps of the file shall be marked for  
 34698 update between the successful execution of `fputwc()` and the next successful completion of a call  
 34699 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.

34700 The `fputwc()` function shall not change the setting of `errno` if successful.

34701 **RETURN VALUE**

34702 CX Upon successful completion, `fputwc()` shall return `wc`. Otherwise, it shall return WEOF, `errno`  
 34703 shall be set to indicate the error, and for errors other than [EILSEQ] the error indicator for the  
 34704 stream shall be set; the error indicator for the stream shall also be set for [EILSEQ] errors.

34705 **ERRORS**

34706 The `fputwc()` function shall fail if either the stream is unbuffered or data in the `stream`'s buffer  
 34707 needs to be written, and:

34708 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor underlying `stream` and  
 34709 the thread would be delayed in the write operation.

34710 CX [EBADF] The file descriptor underlying `stream` is not a valid file descriptor open for  
 34711 writing.

34712 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

34713 CX [EFBIG] An attempt was made to write to a file that exceeds the file size limit of the  
 34714 process.

34715 XSI A SIGXFSZ signal shall also be generated for the thread.

34716 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 34717 offset maximum associated with the corresponding stream.

34718 [EILSEQ] The wide-character code `wc` does not correspond to a valid character.

34719 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data  
 34720 was transferred.

34721 CX [EIO] A physical I/O error has occurred, or the process is a member of a background  
 34722 process group attempting to write to its controlling terminal, TOSTOP is set,  
 34723 the calling thread is not blocking SIGTTOU, the process is not ignoring  
 34724 SIGTTOU, and the process group of the process is orphaned. This error may

34725 also be returned under implementation-defined conditions.

34726 CX [ENOSPC] There was no free space remaining on the device containing the file.

34727 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
34728 any process. A SIGPIPE signal shall also be sent to the thread.

34729 The *fputc()* function may fail if:

34730 CX [ENOMEM] Insufficient storage space is available.

34731 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
34732 capabilities of the device.

**34733 EXAMPLES**

34734 None.

**34735 APPLICATION USAGE**

34736 None.

**34737 RATIONALE**

34738 The requirement to set the error indicator for the stream on [EILSEQ] errors is CX shaded  
34739 because the ISO C standard does not require it to be set for *fputc()* encoding errors, although it  
34740 does for *fgetc()*. The next revision of the ISO C standard is expected to address this  
34741 inconsistency by requiring the error indicator for the stream to be set for *fputc()* encoding  
34742 errors.

**34743 FUTURE DIRECTIONS**

34744 None.

**34745 SEE ALSO**

34746 [Section 2.5](#) (on page 521), *ferror()*, *fopen()*, *getrlimit()*, *setbuf()*

34747 XBD [<stdio.h>](#), [<wchar.h>](#)

**34748 CHANGE HISTORY**

34749 First released in Issue 4. Derived from the MSE working draft.

**34750 Issue 5**

34751 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
34752 is changed from **wint\_t** to **wchar\_t**.

34753 The Optional Header (OH) marking is removed from [<stdio.h>](#).

34754 Large File Summit extensions are added.

**34755 Issue 6**

34756 Extensions beyond the ISO C standard are marked.

34757 The following new requirements on POSIX implementations derive from alignment with the  
34758 Single UNIX Specification:

- 34759 • The [EFBIG] and [EIO] mandatory error conditions are added.
- 34760 • The [ENOMEM] and [ENXIO] optional error conditions are added.

34761 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/38 is applied, updating the [EAGAIN]  
34762 error in the ERRORS section from “the process would be delayed” to “the thread would be  
34763 delayed”.

**34764 Issue 7**

34765 Changes are made related to support for finegrained timestamps.

34766 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0173 [105], XSH/TC1-2008/0174 [79],  
34767 and XSH/TC1-2008/0175 [14] are applied.

**34768 Issue 8**

34769 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

34770 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
34771 relating to the file size limit for the process.

34772 Austin Group Defect 1769 is applied, changing the CX shading in the RETURN VALUE section.

34773 **NAME**34774 `fputws` — put a wide-character string on a stream34775 **SYNOPSIS**34776 `#include <stdio.h>`34777 `#include <wchar.h>`34778 `int fputws(const wchar_t *restrict ws, FILE *restrict stream);`34779 **DESCRIPTION**34780 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
34781 conflict between the requirements described here and the ISO C standard is unintentional. This  
34782 volume of POSIX.1-2024 defers to the ISO C standard.34783 The `fputws()` function shall write a character string corresponding to the (null-terminated) wide-  
34784 character string pointed to by `ws` to the stream pointed to by `stream`. No character corresponding  
34785 to the terminating null wide-character code shall be written.34786 CX The last data modification and last file status change timestamps of the file shall be marked for  
34787 update between the successful execution of `fputws()` and the next successful completion of a call  
34788 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.34789 **RETURN VALUE**34790 Upon successful completion, `fputws()` shall return a non-negative number. Otherwise, it shall  
34791 CX return `-1`, set an error indicator for the stream, and set `errno` to indicate the error.34792 **ERRORS**34793 Refer to `fputwc()`.34794 **EXAMPLES**

34795 None.

34796 **APPLICATION USAGE**34797 The `fputws()` function does not append a `<newline>`.34798 This volume of POSIX.1-2024 requires that successful completion simply return a non-negative  
34799 integer. There are at least three known different implementation conventions for this  
34800 requirement:

- 34801
- Return a constant value.
  - Return the last character written.
  - Return the number of bytes written. Note that this implementation convention cannot be  
34804 adhered to for strings longer than `{INT_MAX}` bytes as the value would not be  
34805 representable in the return type of the function. For backwards-compatibility,  
34806 implementations can return the number of bytes for strings of up to `{INT_MAX}` bytes, and  
34807 return `{INT_MAX}` for all longer strings.

34808 **RATIONALE**

34809 None.

34810 **FUTURE DIRECTIONS**

34811 None.

34812 **SEE ALSO**34813 [Section 2.5](#) (on page 521), `fopen()`34814 XBD `<stdio.h>`, `<wchar.h>`



34815 **CHANGE HISTORY**

34816 First released in Issue 4. Derived from the MSE working draft.

34817 **Issue 5**

34818 The Optional Header (OH) marking is removed from `<stdio.h>`.

34819 **Issue 6**

34820 Extensions beyond the ISO C standard are marked.

34821 The `fputws()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

34822 **Issue 7**

34823 Changes are made related to support for finegrained timestamps.

34824 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0176 [412] and XSH/TC1-2008/0177  
34825 [14] are applied.

34826 **NAME**

34827 fread — binary input

34828 **SYNOPSIS**

```
34829 #include <stdio.h>
34830 size_t fread(void *restrict ptr, size_t size, size_t nitems,
34831 FILE *restrict stream);
```

34832 **DESCRIPTION**

34833 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 34834 conflict between the requirements described here and the ISO C standard is unintentional. This  
 34835 volume of POSIX.1-2024 defers to the ISO C standard.

34836 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size  
 34837 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall  
 34838 be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**  
 34839 **char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be  
 34840 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the  
 34841 file position indicator for the stream is unspecified. If a partial element is read, its value is  
 34842 unspecified.

34843 CX The *fread()* function may mark the last data access timestamp of the file associated with *stream*  
 34844 for update. The last data access timestamp shall be marked for update by the first successful  
 34845 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, or *scanf()* using  
 34846 *stream* that returns data not supplied by a prior call to *ungetc()*.

34847 **RETURN VALUE**

34848 The *fread()* function shall return the number of elements successfully read, which shall be less  
 34849 than *nitems* only if an error or end-of-file is encountered, or *size* is zero. If *size* or *nitems* is 0,  
 34850 *fread()* shall return 0 and the contents of the array and the state of the stream shall remain  
 34851 CX unchanged. Otherwise, if an error occurs, the error indicator for the stream shall be set, and  
 34852 *errno* shall be set to indicate the error.

34853 **ERRORS**34854 Refer to *fgetc()*.34855 **EXAMPLES**34856 **Reading from a Stream**

34857 The following example transfers a single 100-byte fixed length record from the *fp* stream into the  
 34858 array pointed to by *buf*.

```
34859 #include <stdio.h>
34860 ...
34861 size_t elements_read;
34862 char buf[100];
34863 FILE *fp;
34864 ...
34865 elements_read = fread(buf, sizeof(buf), 1, fp);
34866 ...
```

34867 If a read error occurs, *elements\_read* will be zero but the number of bytes read from the stream  
 34868 could be anything from zero to *sizeof(buf)*-1.

34869 The following example reads multiple single-byte elements from the *fp* stream into the array  
 34870 pointed to by *buf*.

```

34871     #include <stdio.h>
34872     ...
34873     size_t bytes_read;
34874     char buf[100];
34875     FILE *fp;
34876     ...
34877     bytes_read = fread(buf, 1, sizeof(buf), fp);
34878     ...

```

34879 If a read error occurs, *bytes\_read* will contain the number of bytes read from the stream.

#### 34880 APPLICATION USAGE

34881 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an  
 34882 end-of-file condition.

34883 Because of possible differences in element length and byte ordering, files written using *fwrite()*  
 34884 are application-dependent, and possibly cannot be read using *fread()* by a different application  
 34885 or by the same application on a different processor.

#### 34886 RATIONALE

34887 None.

#### 34888 FUTURE DIRECTIONS

34889 None.

#### 34890 SEE ALSO

34891 [Section 2.5](#) (on page 521), *feof()*, *ferror()*, *fgetc()*, *fopen()*, *fscanf()*, *getc()*

34892 XBD [<stdio.h>](#)

#### 34893 CHANGE HISTORY

34894 First released in Issue 1. Derived from Issue 1 of the SVID.

##### 34895 Issue 6

34896 Extensions beyond the ISO C standard are marked.

34897 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 34898 • The *fread()* prototype is updated.
- 34899 • The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

##### 34900 Issue 7

34901 Changes are made related to support for finegrained timestamps.

34902 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0178 [232] and XSH/TC1-2008/0179  
 34903 [14] are applied.

34904 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0129 [926] is applied.

##### 34905 Issue 8

34906 Austin Group Defect 1196 is applied, clarifying the RETURN VALUE section.

34907 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

34908 Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

34909 **NAME**

34910 free — free allocated memory

34911 **SYNOPSIS**

34912 #include &lt;stdlib.h&gt;

34913 void free(void \*ptr);

34914 **DESCRIPTION**

34915 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 34916 conflict between the requirements described here and the ISO C standard is unintentional. This  
 34917 volume of POSIX.1-2024 defers to the ISO C standard.

34918 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made  
 34919 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the  
 34920 argument does not match a pointer earlier returned by *aligned\_alloc()*, *calloc()*, *malloc()*,  
 34921 ADV *posix\_memalign()*, *realloc()*,  
 34922 CX *reallocarray()*, or a function in POSIX.1-2024 that allocates memory as if by *malloc()*, or if the  
 34923 CX space has been deallocated by a call to *free()*, *reallocarray()*, or *realloc()*, the behavior is  
 34924 undefined.

34925 Any use of a pointer that refers to freed space results in undefined behavior.

34926 CX The *free()* function shall not modify *errno* if *ptr* is a null pointer or a pointer previously returned  
 34927 as if by *malloc()* and not yet deallocated.

34928 For purposes of determining the existence of a data race, *free()* shall behave as though it  
 34929 accessed only memory locations accessible through its argument and not other static duration  
 34930 storage. The function may, however, visibly modify the storage that it deallocates. Calls to  
 34931 ADV *aligned\_alloc()*, *calloc()*, *free()*, *malloc()*, *posix\_memalign()*,  
 34932 CX *reallocarray()*, and *realloc()* that allocate or deallocate a particular region of memory shall occur  
 34933 in a single total order (see XBD Section 4.15.1, on page 100), and each such deallocation call shall  
 34934 synchronize with the next allocation (if any) in this order.

34935 **RETURN VALUE**34936 The *free()* function shall not return a value.34937 **ERRORS**

34938 No errors are defined.

34939 **EXAMPLES**

34940 None.

34941 **APPLICATION USAGE**

34942 There is now no requirement for the implementation to support the inclusion of &lt;malloc.h&gt;.

34943 Because the *free()* function does not modify *errno* for valid pointers, it is safe to use it in cleanup  
 34944 code without corrupting earlier errors, such as in this example code:

```
34945 // buf was obtained by malloc(buflen)
34946 ret = write(fd, buf, buflen);
34947 if (ret < 0) {
34948     free(buf);
34949     return ret;
34950 }
```

34951 However, earlier versions of this standard did not require this, and the same example had to be  
 34952 written as:

```
34953 // buf was obtained by malloc(buflen)
```

```
34954         ret = write(fd, buf, buflen);
34955         if (ret < 0) {
34956             int save = errno;
34957             free(buf);
34958             errno = save;
34959             return ret;
34960         }
```

**34961 RATIONALE**

34962 None.

**34963 FUTURE DIRECTIONS**

34964 None.

**34965 SEE ALSO**

34966 [\*aligned\\_alloc\(\)\*](#), [\*calloc\(\)\*](#), [\*malloc\(\)\*](#), [\*posix\\_memalign\(\)\*](#), [\*realloc\(\)\*](#)

34967 XBD <[\*stdlib.h\*](#)>

**34968 CHANGE HISTORY**

34969 First released in Issue 1. Derived from Issue 1 of the SVID.

**34970 Issue 6**

34971 Reference to the *valloc()* function is removed.

**34972 Issue 7**

34973 The DESCRIPTION is updated to clarify that if the pointer returned is not by a function that  
34974 allocates memory as if by *malloc()*, then the behavior is undefined.

**34975 Issue 8**

34976 Austin Group Defect 385 is applied, adding a requirement that *free()* does not modify *errno*  
34977 when passed a pointer to an object than can be freed.

34978 Austin Group Defect 1218 is applied, adding *reallocarray()*.

34979 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
34980 standard.

34981 **NAME**

34982 freeaddrinfo, getaddrinfo — get address information

34983 **SYNOPSIS**

```

34984 #include <sys/socket.h>
34985 #include <netdb.h>

34986 void freeaddrinfo(struct addrinfo *ai);
34987 int getaddrinfo(const char *restrict nodename,
34988               const char *restrict servname,
34989               const struct addrinfo *restrict hints,
34990               struct addrinfo **restrict res);

```

34991 **DESCRIPTION**

34992 The `freeaddrinfo()` function shall free one or more **addrinfo** structures returned by `getaddrinfo()`,  
 34993 along with any additional storage associated with those structures. If the `ai_next` field of the  
 34994 structure is not null, the entire list of structures shall be freed. The `freeaddrinfo()` function shall  
 34995 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by `getaddrinfo()`.  
 34996 The `freeaddrinfo()` function shall not modify `errno` if `ai` is a sublist previously returned by  
 34997 `getaddrinfo()` and not yet freed.

34998 The `getaddrinfo()` function shall translate the name of a service location (for example, a host  
 34999 name) and/or a service name and shall return a set of socket addresses and associated  
 35000 information to be used in creating a socket with which to address the specified service.

35001 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,  
 35002 RFC 1035, and RFC 3596.

35003 The `freeaddrinfo()` and `getaddrinfo()` functions shall be thread-safe.

35004 The `nodename` and `servname` arguments are either null pointers or pointers to null-terminated  
 35005 strings. One or both of these two arguments shall be supplied by the application as a non-null  
 35006 pointer.

35007 The format of a valid name depends on the address family or families. If a specific family is not  
 35008 given and the name could be interpreted as valid within multiple supported families, the  
 35009 implementation shall attempt to resolve the name in all supported families and, in absence of  
 35010 errors, one or more results shall be returned.

35011 If the `nodename` argument is not null, it can be a descriptive name or can be an address string. If  
 35012 IP6 the specified address family is AF\_INET, AF\_INET6, or AF\_UNSPEC, valid descriptive names  
 35013 include host names. If the specified address family is AF\_INET or AF\_UNSPEC, address strings  
 35014 using Internet standard dot notation as specified in `inet_ntop()` are valid.

35015 IP6 If the specified address family is AF\_INET6 or AF\_UNSPEC, standard IPv6 text forms described  
 35016 in `inet_ntop()` are valid.

35017 If `nodename` is not null, the requested service location is named by `nodename`; otherwise, the  
 35018 requested service location is local to the caller.

35019 If `servname` is null, the call shall return network-level addresses for the specified `nodename`. If  
 35020 `servname` is not null, it is a null-terminated character string identifying the requested service.  
 35021 This can be either a descriptive name or a numeric representation suitable for use with the  
 35022 IP6 address family or families. If the specified address family is AF\_INET, AF\_INET6, or  
 35023 AF\_UNSPEC, the service can be specified as a string specifying a decimal port number.

35024 If the `hints` argument is not null, it refers to a structure containing input values that directs the  
 35025 operation by providing options and by limiting the returned information to a specific socket  
 35026 type, address family, and/or protocol, as described below. The application shall ensure that each

35027 of the *ai\_addrlen*, *ai\_addr*, *ai\_canonname*, and *ai\_next* members, as well as each of the non-standard  
 35028 additional members, if any, of this *hints* structure is initialized. If any of these members has a  
 35029 value other than the value that would result from default initialization, the behavior is  
 35030 implementation-defined. A value of AF\_UNSPEC for *ai\_family* means that the caller shall accept  
 35031 any address family. A value of zero for *ai\_socktype* means that the caller shall accept any socket  
 35032 type. A value of zero for *ai\_protocol* means that the caller shall accept any protocol. If *hints* is a  
 35033 null pointer, the behavior shall be as if it referred to a structure containing the value zero for the  
 35034 *ai\_flags*, *ai\_socktype*, and *ai\_protocol* fields, and AF\_UNSPEC for the *ai\_family* field.

35035 The *ai\_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-  
 35036 inclusive OR of one or more of the values AI\_PASSIVE, AI\_CANONNAME,  
 35037 AI\_NUMERICHOST, AI\_NUMERICSERV, AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG.

35038 If the AI\_PASSIVE flag is specified, the returned address information shall be suitable for use in  
 35039 binding a socket for accepting incoming connections for the specified service. In this case, if the  
 35040 *nodename* argument is null, then the IP address portion of the socket address structure shall be  
 35041 set to INADDR\_ANY for an IPv4 address or IN6ADDR\_ANY\_INIT for an IPv6 address. If the  
 35042 AI\_PASSIVE flag is not specified, the returned address information shall be suitable for a call to  
 35043 *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a  
 35044 connectionless protocol). In this case, if the *nodename* argument is null, then the IP address  
 35045 portion of the socket address structure shall be set to the loopback address. The AI\_PASSIVE  
 35046 flag shall be ignored if the *nodename* argument is not null.

35047 If the AI\_CANONNAME flag is specified and the *nodename* argument is not null, the function  
 35048 shall attempt to determine the canonical name corresponding to *nodename* (for example, if  
 35049 *nodename* is an alias or shorthand notation for a complete name).

35050 **Note:** Since different implementations use different conceptual models, the terms “canonical name”  
 35051 and “alias” cannot be precisely defined for the general case. However, Domain Name System  
 35052 implementations are expected to interpret them as they are used in RFC 1034.

35053 A numeric host address string is not a “name”, and thus does not have a “canonical name”  
 35054 form; no address to host name translation is performed. See below for handling of the case  
 35055 where a canonical name cannot be obtained.

35056 If the AI\_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a  
 35057 numeric host address string. Otherwise, an [EAI\_NONAME] error is returned. This flag shall  
 35058 prevent any type of name resolution service (for example, the DNS) from being invoked.

35059 If the AI\_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a  
 35060 numeric port string. Otherwise, an [EAI\_NONAME] error shall be returned. This flag shall  
 35061 prevent any type of name resolution service (for example, NIS+) from being invoked.

35062 IP6 By default, with an *ai\_family* of AF\_INET6, *getaddrinfo()* shall return only IPv6 addresses. If the  
 35063 AI\_V4MAPPED flag is specified along with an *ai\_family* of AF\_INET6, then *getaddrinfo()* shall  
 35064 return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses. The  
 35065 AI\_V4MAPPED flag shall be ignored unless *ai\_family* equals AF\_INET6. If the AI\_ALL flag is  
 35066 used with the AI\_V4MAPPED flag, then *getaddrinfo()* shall return all matching IPv6 and IPv4  
 35067 addresses. The AI\_ALL flag without the AI\_V4MAPPED flag shall be ignored.

35068 If the AI\_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4  
 35069 IP6 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6  
 35070 address is configured on the local system.

35071 The *ai\_socktype* field to which argument *hints* points specifies the socket type for the service, as  
 35072 defined in Section 2.10.6 (on page 550). If a specific socket type is not given (for example, a  
 35073 value of zero) and the service name could be interpreted as valid with multiple supported socket

35074 types, the implementation shall attempt to resolve the service name for all supported socket  
 35075 types and, in the absence of errors, all possible results shall be returned. A non-zero socket type  
 35076 value shall limit the returned information to values with the specified socket type.

35077 If the *ai\_family* field to which *hints* points has the value AF\_UNSPEC, addresses shall be  
 35078 returned for use with any address family that can be used with the specified *nodename* and/or  
 35079 *servname*. Otherwise, addresses shall be returned for use only with the specified address family.  
 35080 If *ai\_family* is not AF\_UNSPEC and *ai\_protocol* is not zero, then addresses shall be returned for  
 35081 use only with the specified address family and protocol; the value of *ai\_protocol* shall be  
 35082 interpreted as in a call to the *socket()* function with the corresponding values of *ai\_family* and  
 35083 *ai\_protocol*.

#### 35084 RETURN VALUE

35085 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value  
 35086 indicates failure. The possible values for the failures are listed in the ERRORS section.

35087 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list  
 35088 of **addrinfo** structures, each of which shall specify a socket address and information for use in  
 35089 creating a socket with which to use that socket address. The list shall include at least one  
 35090 **addrinfo** structure. The *ai\_next* field of each structure contains a pointer to the next structure on  
 35091 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall  
 35092 include values for use with a call to the *socket()* function, and a socket address for use with the  
 35093 *connect()* function or, if the AI\_PASSIVE flag was specified, for use with the *bind()* function. The  
 35094 fields *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be usable as the arguments to the *socket()*  
 35095 function to create a socket suitable for use with the returned address. The fields *ai\_addr* and  
 35096 *ai\_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,  
 35097 according to the AI\_PASSIVE flag.

35098 If *nodename* is not null, and if requested by the AI\_CANONNAME flag, the *ai\_canonname* field of  
 35099 the first returned **addrinfo** structure shall point to a null-terminated string containing the  
 35100 canonical name corresponding to the input *nodename*; if the canonical name is not available, then  
 35101 *ai\_canonname* shall refer to the *nodename* argument or a string with the same contents. The  
 35102 contents of the *ai\_flags* field of the returned structures are undefined.

35103 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an  
 35104 explicit argument (for example, *sin6\_flowinfo*) shall be set to zero.

35105 **Note:** This makes it easier to compare socket address structures.

#### 35106 ERRORS

35107 The *getaddrinfo()* function shall fail and return the corresponding error value if:

35108 [EAI\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

35109 [EAI\_BADFLAGS]

35110 The *flags* parameter had an invalid value.

35111 [EAI\_FAIL] A non-recoverable error occurred when attempting to resolve the name.

35112 [EAI\_FAMILY] The address family was not recognized.

35113 [EAI\_MEMORY] There was a memory allocation failure when trying to allocate storage for the  
 35114 return value.

35115 [EAI\_NONAME] The name does not resolve for the supplied parameters.

35116 Neither *nodename* nor *servname* were supplied. At least one of these shall be  
 35117 supplied.



35118 [EAI\_SERVICE] The service passed was not recognized for the specified socket type.

35119 [EAI\_SOCKTYPE]

35120 The intended socket type was not recognized.

35121 [EAI\_SYSTEM] A system error occurred; the error code can be found in *errno*.

### 35122 EXAMPLES

35123 The following (incomplete) program demonstrates the use of *getaddrinfo()* to obtain the socket  
35124 address structure(s) for the service named in the program's command-line argument. The  
35125 program then loops through each of the address structures attempting to create and bind a  
35126 socket to the address, until it performs a successful *bind()*.

```

35127 #include <stdio.h>
35128 #include <stdlib.h>
35129 #include <unistd.h>
35130 #include <string.h>
35131 #include <sys/socket.h>
35132 #include <netdb.h>

35133 int
35134 main(int argc, char *argv[])
35135 {
35136     struct addrinfo *result, *rp;
35137     int sfd, s;

35138     if (argc != 2) {
35139         fprintf(stderr, "Usage: %s port\n", argv[0]);
35140         exit(EXIT_FAILURE);
35141     }

35142     struct addrinfo hints = {0};
35143     hints.ai_family = AF_UNSPEC;
35144     hints.ai_socktype = SOCK_DGRAM;
35145     hints.ai_flags = AI_PASSIVE;
35146     hints.ai_protocol = 0;

35147     s = getaddrinfo(NULL, argv[1], &hints, &result);
35148     if (s != 0) {
35149         fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
35150         exit(EXIT_FAILURE);
35151     }

35152     /* getaddrinfo() returns a list of address structures.
35153     Try each address until a successful bind().
35154     If socket(2) (or bind(2)) fails, close the socket
35155     and try the next address. */

35156     for (rp = result; rp != NULL; rp = rp->ai_next) {
35157         sfd = socket(rp->ai_family, rp->ai_socktype,
35158                   rp->ai_protocol);
35159         if (sfd == -1)
35160             continue;

35161         if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
35162             break;          /* Success */

35163         close(sfd);

```

```

35164     }
35165     if (rp == NULL) {           /* No address succeeded */
35166         fprintf(stderr, "Could not bind\n");
35167         exit(EXIT_FAILURE);
35168     }
35169     freeaddrinfo(result);      /* No longer needed */
35170                               /* ... use socket bound to sfd ... */
35171 }

```

### 35172 APPLICATION USAGE

35173 If the caller handles only TCP and not UDP, for example, then the *ai\_protocol* member of the *hints*  
 35174 structure should be set to IPPROTO\_TCP when *getaddrinfo()* is called.

35175 If the caller handles only IPv4 and not IPv6, then the *ai\_family* member of the *hints* structure  
 35176 should be set to AF\_INET when *getaddrinfo()* is called.

35177 The *hints* structure can be initialized using `memset(&hints, 0, sizeof hints)` or by  
 35178 default initialization (see the APPLICATION USAGE for XBD <[netdb.h](#)>).

35179 The term “canonical name” is misleading; it is taken from the Domain Name System (RFC 2181).  
 35180 It should be noted that the canonical name is a result of alias processing, and not necessarily a  
 35181 unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this  
 35182 in the Domain Name System context.

35183 The *ai\_socktype* field pointed to by *hints* is just the socket type; not the socket type and flags that  
 35184 can be specified when the socket is created. Passing in socket creation flags will cause a failure  
 35185 with [EAI\_SOCKTYPE].

### 35186 RATIONALE

35187 None.

### 35188 FUTURE DIRECTIONS

35189 None.

### 35190 SEE ALSO

35191 [connect\(\)](#), [endservent\(\)](#), [gai\\_strerror\(\)](#), [getnameinfo\(\)](#), [socket\(\)](#)

35192 XBD <[netdb.h](#)>, <[sys/socket.h](#)>

### 35193 CHANGE HISTORY

35194 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35195 The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the  
 35196 ISO/IEC 9899:1999 standard.

35197 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the  
 35198 DESCRIPTION and adding text to the APPLICATION USAGE related to the term “canonical  
 35199 name”. A reference to RFC 2181 is also added to the Informative References.

35200 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for  
 35201 alignment with IPv6. These include the following:

- 35202 • Adding AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG to the allowed values for the  
 35203 *ai\_flags* field
- 35204 • Adding a description of AI\_ADDRCONFIG

- 35205           • Adding a description of the consequences of ignoring the AI\_PASSIVE flag.
- 35206           IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/39 is applied, changing “corresponding  
35207           value” to “corresponding error value” in the ERRORS section.
- 35208   **Issue 7**
- 35209           Austin Group Interpretation 1003.1-2001 #013 is applied.
- 35210           Austin Group Interpretation 1003.1-2001 #146 is applied, updating the DESCRIPTION.
- 35211           An example is added.
- 35212           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0130 [939], XSH/TC2-2008/0131 [979],  
35213           XSH/TC2-2008/0132 [918], and XSH/TC2-2008/0133 [934] are applied.
- 35214   **Issue 8**
- 35215           Austin Group Defect 385 is applied, adding a requirement that *freeaddrinfo()* does not modify  
35216           *errno* when passed a sublist that can be freed.
- 35217           Austin Group Defect 411 is applied, changing the “socket type” reference and adding a  
35218           paragraph about *hints->ai\_socktype* to the APPLICATION USAGE section.
- 35219           Austin Group Defect 940 is applied, changing text in the APPLICATION USAGE section relating  
35220           to initialization of the *hints* structure.
- 35221           Austin Group Defect 1102 is applied, replacing a reference to the *inet\_addr()* page with one to the  
35222           *inet\_ntop()* page.
- 35223           Austin Group Defect 1685 is applied, updating RFC references.

35224 **NAME**

35225           freelocale — free resources allocated for a locale object

35226 **SYNOPSIS**

```
35227 CX       #include <locale.h>
35228       void freelocale(locale_t locobj);
```

35229 **DESCRIPTION**

35230       The *freelocale()* function shall cause the resources allocated for a locale object returned by a call  
35231       to the *newlocale()* or *duplocale()* functions to be released. The *freelocale()* function shall not  
35232       modify *errno* if *locobj* is a locale object previously returned by *newlocale()* or *duplocale()* and not  
35233       yet released by *freelocale()* or *newlocale()*.

35234       The behavior is undefined if the *locobj* argument is the special locale object  
35235       LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

35236       Any use of a locale object that has been freed results in undefined behavior.

35237 **RETURN VALUE**

35238       None.

35239 **ERRORS**

35240       None.

35241 **EXAMPLES**35242       **Freeing Up a Locale Object**

35243       The following example shows a code fragment to free a locale object created by *newlocale()*:

```
35244       #include <locale.h>
35245       ...
35246       /* Every locale object allocated with newlocale() should be
35247        * freed using freelocale():
35248        */
35249       locale_t loc;
35250       /* Get the locale. */
35251       loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
35252       /* ... Use the locale object ... */
35253       ...
35254       /* Free the locale object resources. */
35255       freelocale (loc);
```

35256 **APPLICATION USAGE**

35257       None.

35258 **RATIONALE**

35259       None.

35260 **FUTURE DIRECTIONS**

35261       None.

35262 **SEE ALSO**35263 *duplocale()*, *getlocalename\_l()*, *newlocale()*, *uselocale()*

35264 XBD &lt;locale.h&gt;

35265 **CHANGE HISTORY**

35266 First released in Issue 7.

35267 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0180 [283] is applied.

35268 **Issue 8**35269 Austin Group Defect 385 is applied, adding a requirement that *freelocale()* does not modify *errno*  
35270 when passed a locale object than can be freed.35271 Austin Group Defect 1220 is applied, adding *getlocalename\_l()* to the SEE ALSO section.

## 35272 NAME

35273 freopen — open a stream

## 35274 SYNOPSIS

35275 #include &lt;stdio.h&gt;

```
35276 FILE *freopen(const char *restrict pathname, const char *restrict mode,
35277 FILE *restrict stream);
```

## 35278 DESCRIPTION

35279 CX Except for the “exclusive access” requirement (see *fopen()*), the functionality described on this  
 35280 reference page is aligned with the ISO C standard. Any other conflict between the requirements  
 35281 described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to  
 35282 the ISO C standard for all *freopen()* functionality except in relation to “exclusive access”.

35283 The *freopen()* function shall first attempt to flush the stream associated with *stream* as if by a call  
 35284 to *fflush(stream)*. Failure to flush the stream successfully shall be ignored. If *pathname* is not a  
 35285 null pointer, *freopen()* shall close any file descriptor associated with *stream*. Failure to close the  
 35286 file descriptor successfully shall be ignored. The error and end-of-file indicators for the stream  
 35287 shall be cleared.

35288 The *freopen()* function shall open the file whose pathname is the string pointed to by *pathname*  
 35289 and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in  
 35290 *fopen()*.

35291 The original stream shall be closed regardless of whether the subsequent open succeeds.

35292 If *pathname* is a null pointer, the *freopen()* function shall attempt to change the mode of the  
 35293 stream to that specified by *mode*, as if the name of the file currently associated with the stream  
 35294 had been used. In this case, the file descriptor associated with the stream need not be closed if  
 35295 the call to *freopen()* succeeds. It is implementation-defined which changes of mode are permitted  
 35296 (if any), and under what circumstances.

35297 XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the  
 35298 encoding rule shall be cleared, and the associated *mbstate\_t* object shall be set to describe an  
 35299 initial conversion state.

35300 CX If *pathname* is not a null pointer, or if *pathname* is a null pointer and the specified mode change  
 35301 necessitates the file descriptor associated with the stream to be closed and reopened, the file  
 35302 descriptor associated with the reopened stream shall be allocated and opened as if by a call to  
 35303 *open()* with the flags specified for *fopen()* with the same *mode* argument.

## 35304 RETURN VALUE

35305 Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer  
 35306 CX shall be returned, and *errno* shall be set to indicate the error.

## 35307 ERRORS

35308 The *freopen()* function shall fail if:

35309 CX [EACCES] Search permission is denied on a component of the path prefix, or the file  
 35310 exists and the permissions specified by *mode* are denied, or the file does not  
 35311 exist and write permission is denied for the parent directory of the file to be  
 35312 created.

35313 CX [EBADF] The file descriptor underlying the stream is not a valid file descriptor when  
 35314 *pathname* is a null pointer.

35315	CX	[EILSEQ]	The <i>mode</i> argument begins with <i>w</i> or <i>a</i> , the file did not previously exist, and the last pathname component is not a portable filename and cannot be created in the target directory.
35316			
35317			
35318	CX	[EEXIST]	The <i>mode</i> argument begins with <i>w</i> or <i>a</i> and includes <i>x</i> , but the file already exists.
35319			
35320	CX	[EINTR]	A signal was caught during <i>freopen()</i> .
35321	CX	[EISDIR]	The named file is a directory and <i>mode</i> requires write access.
35322	CX	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>pathname</i> argument.
35323			
35324	CX	[EMFILE]	All file descriptors available to the process are currently open.
35325	CX	[ENAMETOOLONG]	
35326			The length of a component of a pathname is longer than {NAME_MAX}.
35327	CX	[ENFILE]	The maximum allowable number of files is currently open in the system.
35328	CX	[ENOENT]	The <i>mode</i> string begins with 'r' and a component of <i>pathname</i> does not name an existing file, or <i>mode</i> begins with 'w' or 'a' and a component of the path prefix of <i>pathname</i> does not name an existing file, or <i>pathname</i> is an empty string.
35329			
35330			
35331			
35332	CX	[ENOENT] or [ENOTDIR]	
35333			The <i>pathname</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters. If <i>pathname</i> without the trailing <code>&lt;slash&gt;</code> characters would name an existing file, an [ENOENT] error shall not occur.
35334			
35335			
35336			
35337	CX	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.
35338			
35339	CX	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>pathname</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
35340			
35341			
35342			
35343			
35344	CX	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
35345			
35346	CX	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .
35347			
35348	CX	[EROFS]	The named file resides on a read-only file system and <i>mode</i> requires write access.
35349			
35350			The <i>freopen()</i> function may fail if:
35351	CX	[EBADF]	The mode with which the file descriptor underlying the stream was opened does not support the requested mode when <i>pathname</i> is a null pointer.
35352			
35353	CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
35354	CX	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>pathname</i> argument.
35355			

35356	CX	[ENAMETOOLONG]	
35357			The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
35358			symbolic link produced an intermediate result with a length that exceeds
35359			{PATH_MAX}.
35360	CX	[ENOMEM]	Insufficient storage space is available.
35361	CX	[ENXIO]	A request was made of a nonexistent device, or the request was outside the
35362			capabilities of the device.
35363	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i>
35364			requires write access.

### 35365 EXAMPLES

#### 35366 Directing Standard Output to a File

35367 The following example logs all standard output to the `/tmp/logfile` file.

```
35368 #include <stdio.h>
35369 ...
35370 FILE *fp;
35371 ...
35372 fp = freopen ("/tmp/logfile", "a+", stdout);
35373 ...
```

### 35374 APPLICATION USAGE

35375 The `freopen()` function is typically used to attach the pre-opened *streams* associated with *stdin*,

35376 *stdout*, and *stderr* to other files.

35377 Since implementations are not required to support any stream mode changes when the *pathname*

35378 argument is NULL, portable applications cannot rely on the use of `freopen()` to change the stream

35379 mode, and use of this feature is discouraged. The feature was originally added to the ISO C

35380 standard in order to facilitate changing *stdin* and *stdout* to binary mode. Since a 'b' character in

35381 the mode has no effect on POSIX systems, this use of the feature is unnecessary in POSIX

35382 applications. However, even though the 'b' is ignored, a successful call to `freopen(NULL, "wb",`

35383 `stdout)` does have an effect. In particular, for regular files it truncates the file and sets the file-

35384 position indicator for the stream to the start of the file. It is possible that these side-effects are an

35385 unintended consequence of the way the feature was specified in the ISO/IEC 9899:1999

35386 standard (and still is in the current standard), but unless or until the ISO C standard is changed,

35387 applications which successfully call `freopen(NULL, "wb", stdout)` will behave in unexpected ways

35388 on conforming systems in situations such as:

```
35389 { appl file1; appl file2; } > file3
```

35390 which will result in `file3` containing only the output from the second invocation of `appl`.

35391 See also the APPLICATION USAGE for `fopen()`.

### 35392 RATIONALE

35393 See the RATIONALE for `fopen()`.

### 35394 FUTURE DIRECTIONS

35395 None.



35396 **SEE ALSO**

35397 Section 2.5 (on page 521), *fclose()*, *fdopen()*, *fflush()*, *fmemopen()*, *fopen()*, *mbsinit()*, *open()*,  
 35398 *open\_memstream()*

35399 XBD <stdio.h>

35400 **CHANGE HISTORY**

35401 First released in Issue 1. Derived from Issue 1 of the SVID.

35402 **Issue 5**

35403 The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the  
 35404 conversion state of the stream is set to an initial conversion state by a successful call to the  
 35405 *freopen()* function.

35406 Large File Summit extensions are added.

35407 **Issue 6**

35408 Extensions beyond the ISO C standard are marked.

35409 The following new requirements on POSIX implementations derive from alignment with the  
 35410 Single UNIX Specification:

35411 • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
 35412 file description. This change is to support large files.

35413 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
 35414 large files.

35415 • The [ELOOP] mandatory error condition is added.

35416 • A second [ENAMETOOLONG] is added as an optional error condition.

35417 • The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

35418 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

35419 • The *freopen()* prototype is updated.

35420 • The DESCRIPTION is updated.

35421 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 35422 [ELOOP] error condition is added.

35423 The DESCRIPTION is updated regarding failure to close, changing the “file” to “file descriptor”.

35424 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/40 is applied, adding the following  
 35425 sentence to the DESCRIPTION: “In this case, the file descriptor associated with the stream need  
 35426 not be closed if the call to *freopen()* succeeds.”.

35427 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/41 is applied, adding an mandatory  
 35428 [EBADF] error, and an optional [EBADF] error to the ERRORS section.

35429 **Issue 7**

35430 Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the *freopen()* function  
 35431 allocates a file descriptor as per *open()*.

35432 Austin Group Interpretation 1003.1-2001 #143 is applied.

35433 Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set  
 35434 on the open file description.

35435 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

- 35436 SD5-XSH-ERN-150 and SD5-XSH-ERN-219 are applied.
- 35437 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0181 [291,433], XSH/TC1-2008/0182  
35438 [146,433], XSH/TC1-2008/0183 [324], and XSH/TC1-2008/0184 [14] are applied.
- 35439 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0134 [822] is applied.
- 35440 **Issue 8**
- 35441 Austin Group Defect 293 is applied, adding the [EILSEQ] error.
- 35442 Austin Group Defect 411 is applied, adding the *e* and *x* mode string characters.
- 35443 Austin Group Defect 1200 is applied, correcting the argument name in the [ELOOP] errors.
- 35444 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
35445 standard.

35446 **NAME**

35447 frexp, frexpf, frexpl — extract significand and exponent from a double precision number

35448 **SYNOPSIS**

35449 #include &lt;math.h&gt;

35450 double frexp(double *num*, int \**exp*);35451 float frexpf(float *num*, int \**exp*);35452 long double frexpl(long double *num*, int \**exp*);35453 **DESCRIPTION**

35454 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 35455 conflict between the requirements described here and the ISO C standard is unintentional. This  
 35456 volume of POSIX.1-2024 defers to the ISO C standard.

35457 These functions shall break a floating-point number *num* into a normalized fraction and an  
 35458 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*; if the  
 35459 integer exponent is outside the range of **int**, the results are unspecified.

35460 **RETURN VALUE**

35461 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the  
 35462 interval  $[\frac{1}{2}, 1)$  or 0, and *num* equals *x* times 2 raised to the power \**exp*.

35463 MX When the radix of the argument is a power of 2, the returned value shall be exact and shall be  
 35464 independent of the current rounding direction mode.

35465 If *num* is NaN, a NaN shall be returned, and the value of \**exp* is unspecified.

35466 If *num* is  $\pm 0$ ,  $\pm 0$  shall be returned, and the value of \**exp* shall be 0.

35467 If *num* is  $\pm\text{Inf}$ , *num* shall be returned, and the value of \**exp* is unspecified.

35468 **ERRORS**

35469 No errors are defined.

35470 **EXAMPLES**

35471 None.

35472 **APPLICATION USAGE**

35473 None.

35474 **RATIONALE**

35475 None.

35476 **FUTURE DIRECTIONS**

35477 None.

35478 **SEE ALSO**35479 [isnan\(\)](#), [ldexp\(\)](#), [modf\(\)](#)35480 XBD [<math.h>](#)35481 **CHANGE HISTORY**

35482 First released in Issue 1. Derived from Issue 1 of the SVID.

35483 **Issue 5**

35484 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 35485 text was previously published in the APPLICATION USAGE section.

35486 **Issue 6**

35487 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
35488 standard.

35489 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
35490 revised to align with the ISO/IEC 9899:1999 standard.

35491 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
35492 marked.

35493 **Issue 8**

35494 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
35495 standard.

35496 Austin Group Defect 1753 is applied, changing the NAME section.

35497 **NAME**

35498 fscanf, scanf, sscanf — convert formatted input

35499 **SYNOPSIS**

```
35500 #include <stdio.h>
35501 int fscanf(FILE *restrict stream, const char *restrict format, ...);
35502 int scanf(const char *restrict format, ...);
35503 int sscanf(const char *restrict s, const char *restrict format, ...);
```

35504 **DESCRIPTION**

35505 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 35506 conflict between the requirements described here and the ISO C standard is unintentional. This  
 35507 volume of POSIX.1-2024 defers to the ISO C standard.

35508 The *fscanf()* function shall read from the named input *stream*. The *scanf()* function shall read  
 35509 from the standard input stream *stdin*. The *sscanf()* function shall read from the string *s*. Each  
 35510 function reads bytes, interprets them according to a format, and stores the results in its  
 35511 arguments. Each expects, as arguments, a control string *format* described below, and a set of  
 35512 *pointer* arguments indicating where the converted input should be stored. The result is  
 35513 undefined if there are insufficient arguments for the format. If the format is exhausted while  
 35514 arguments remain, the excess arguments shall be evaluated but otherwise ignored.

35515 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 35516 to the next unused argument. In this case, the conversion specifier character % (see below) is  
 35517 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}].  
 35518 This feature provides for the definition of format strings that select arguments in an order  
 35519 appropriate to specific languages. In format strings containing the "%n\$" form of conversion  
 35520 specifications, it is unspecified whether numbered arguments in the argument list can be  
 35521 referenced from the format string more than once.

35522 The format can contain either form of a conversion specification—that is, % or "%n\$"—but the  
 35523 two forms cannot be mixed within a single format string. The only exception to this is that %% or  
 35524 %\* can be mixed with the "%n\$" form. When numbered argument specifications are used,  
 35525 specifying the *N*th argument requires that all the leading arguments, from the first to the  
 35526 (*N*−1)th, are pointers.

35527 The *fscanf()* function in all its forms shall allow detection of a language-dependent radix  
 35528 character in the input string. The radix character is defined in the current locale (category  
 35529 LC\_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the  
 35530 radix character shall default to a <period> ('.').

35531 The application shall ensure that the format is a character string, beginning and ending in its  
 35532 initial shift state, if any, composed of zero or more directives. Each directive is composed of one  
 35533 of the following: one or more white-space bytes; an ordinary character (neither '%' nor a white-  
 35534 space byte); or a conversion specification. Each conversion specification is introduced by the  
 35535 CX character '%' or the character sequence "%n\$", after which the following appear in sequence:

- 35536 • An optional assignment-suppressing character '\*'.
- 35537 • An optional non-zero decimal integer that specifies the maximum field width.
- 35538 CX • An optional assignment-allocation character 'm'.
- 35539 • An option length modifier that specifies the size of the receiving object.
- 35540 • A *conversion specifier* character that specifies the type of conversion to be applied. The valid  
 35541 conversion specifiers are described below.

35542 The *fscanf()* functions shall execute each directive of the format in turn. When all directives have  
 35543 been executed, or if a directive fails (as detailed below), the function shall return. Failures are  
 35544 described as input failures (due to the unavailability of input bytes) or matching failures (due to  
 35545 inappropriate input).

35546 A directive composed of one or more white-space bytes shall be executed by reading input up to  
 35547 the first non-white-space byte, which shall remain unread, or until no more bytes can be read.  
 35548 The directive shall never fail.

35549 A directive that is an ordinary character shall be executed as follows: the next byte shall be read  
 35550 from the input and compared with the byte that comprises the directive; if the comparison  
 35551 shows that they are not equivalent, the directive shall fail, and the differing and subsequent  
 35552 bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a  
 35553 character from being read, the directive shall fail.

35554 A directive that is a conversion specification defines a set of matching input sequences, as  
 35555 described below for each conversion character. A conversion specification shall be executed in  
 35556 the following steps.

35557 Input white-space bytes shall be skipped, unless the conversion specification includes a `[`, `c`, `C`,  
 35558 or `n` conversion specifier.

35559 An item shall be read from the input, unless the conversion specification includes an `n`  
 35560 conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to  
 35561 any specified maximum field width, which may be measured in characters or bytes dependent  
 35562 on the conversion specifier) which is an initial subsequence of a matching sequence. The first  
 35563 byte, if any, after the input item shall remain unread. If the length of the input item is 0, the  
 35564 execution of the conversion specification shall fail; this condition is a matching failure, unless  
 35565 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is  
 35566 an input failure.

35567 Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion  
 35568 specification, the count of input bytes) shall be converted to a type appropriate to the conversion  
 35569 character. If the input item is not a matching sequence, the execution of the conversion  
 35570 specification fails; this condition is a matching failure. Unless assignment suppression was  
 35571 indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the  
 35572 first argument following the *format* argument that has not already received a conversion result if  
 35573 the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the  
 35574 character sequence `"%n$"`. If this object does not have an appropriate type, or if the result of the  
 35575 conversion cannot be represented in the space provided, the behavior is undefined.

35576 CX The `c`, `s`, and `[` conversion specifiers shall accept an optional assignment-allocation character  
 35577 `'m'`, which shall cause a memory buffer to be allocated to hold the conversion results. If the  
 35578 conversion specifier is `s` or `[`, the allocated buffer shall include space for a terminating null  
 35579 character (or wide character). In such a case, the argument corresponding to the conversion  
 35580 specifier should be a reference to a pointer variable that will receive a pointer to the allocated  
 35581 buffer. The system shall allocate a buffer as if *malloc()* had been called. The application shall be  
 35582 responsible for freeing the memory after usage. If there is insufficient memory to allocate a  
 35583 buffer, the function shall set *errno* to `[ENOMEM]` and a conversion error shall result. If the  
 35584 function returns EOF, any memory successfully allocated for parameters using assignment-  
 35585 allocation character `'m'` by this call shall be freed before the function returns.

- 35586 The length modifiers and their meanings are:
- 35587 hh Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
35588 argument with type pointer to **signed char** or **unsigned char**.
- 35589 h Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
35590 argument with type pointer to **short** or **unsigned short**.
- 35591 l (ell) Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
35592 argument with type pointer to **long** or **unsigned long**; that a following a, A, e, E, f, F,  
35593 g, or G conversion specifier applies to an argument with type pointer to **double**; or that  
35594 a following c, s, or [ conversion specifier applies to an argument with type pointer to  
35595 CX **wchar\_t**. If the 'm' assignment-allocation character is specified, the conversion  
35596 applies to an argument with the type pointer to a pointer to **wchar\_t**.
- 35597 ll (ell-ell)  
35598 Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
35599 argument with type pointer to **long long** or **unsigned long long**.
- 35600 j Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
35601 argument with type pointer to **intmax\_t** or **uintmax\_t**.
- 35602 z Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
35603 argument with type pointer to **size\_t** or the corresponding signed integer type.
- 35604 t Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
35605 argument with type pointer to **ptrdiff\_t** or the corresponding **unsigned** type.
- 35606 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an  
35607 argument with type pointer to **long double**.
- 35608 If a length modifier appears with any conversion specifier other than as specified above, the  
35609 behavior is undefined.
- 35610 The following conversion specifiers are valid:
- 35611 d Matches an optionally signed decimal integer, whose format is the same as expected for  
35612 the subject sequence of *strtol()* with the value 10 for the *base* argument. In the absence  
35613 of a size modifier, the application shall ensure that the corresponding argument is a  
35614 pointer to **int**.
- 35615 i Matches an optionally signed integer, whose format is the same as expected for the  
35616 subject sequence of *strtol()* with 0 for the *base* argument. In the absence of a size  
35617 modifier, the application shall ensure that the corresponding argument is a pointer to  
35618 **int**.
- 35619 o Matches an optionally signed octal integer, whose format is the same as expected for  
35620 the subject sequence of *strtoul()* with the value 8 for the *base* argument. In the absence  
35621 of a size modifier, the application shall ensure that the corresponding argument is a  
35622 pointer to **unsigned**.
- 35623 u Matches an optionally signed decimal integer, whose format is the same as expected for  
35624 the subject sequence of *strtoul()* with the value 10 for the *base* argument. In the absence  
35625 of a size modifier, the application shall ensure that the corresponding argument is a  
35626 pointer to **unsigned**.
- 35627 x Matches an optionally signed hexadecimal integer, whose format is the same as  
35628 expected for the subject sequence of *strtoul()* with the value 16 for the *base* argument. In  
35629 the absence of a size modifier, the application shall ensure that the corresponding

35630 argument is a pointer to **unsigned**.

35631 a, e, f, g

35632 Matches an optionally signed floating-point number, infinity, or NaN, whose format is

35633 the same as expected for the subject sequence of *strtod*(). In the absence of a size

35634 modifier, the application shall ensure that the corresponding argument is a pointer to

35635 **float**.

35636 If the *fprintf*() family of functions generates character string representations for infinity

35637 and NaN (a symbolic entity encoded in floating-point format) to support

35638 IEEE Std 754-1985, the *fscanf*() family of functions shall recognize them as input.

35639 s

35640 Matches a sequence of bytes that are not white-space bytes. If the 'm' assignment-

35641 allocation character is not specified, the application shall ensure that the corresponding

35642 argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned**

35643 **char** large enough to accept the sequence and a terminating null character code, which

35644 **CX** shall be added automatically. Otherwise, the application shall ensure that the

corresponding argument is a pointer to a pointer to a **char**.

35645 If an l (ell) qualifier is present, the input is a sequence of characters that begins in the

35646 initial shift state. Each character shall be converted to a wide character as if by a call to

35647 the *mbrtowc*() function, with the conversion state described by an **mbstate\_t** object

35648 initialized to zero before the first character is converted. If the 'm' assignment-

35649 allocation character is not specified, the application shall ensure that the corresponding

35650 argument is a pointer to an array of **wchar\_t** large enough to accept the sequence and

35651 **CX** the terminating null wide character, which shall be added automatically. Otherwise,

35652 the application shall ensure that the corresponding argument is a pointer to a pointer to

35653 a **wchar\_t**.

35654 [

35655 Matches a non-empty sequence of bytes from a set of expected bytes (the *scanset*). The

35656 normal skip over white-space bytes shall be suppressed in this case. If the 'm'

35657 assignment-allocation character is not specified, the application shall ensure that the

35658 corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**,

35659 **CX** or **unsigned char** large enough to accept the sequence and a terminating null byte,

35660 which shall be added automatically. Otherwise, the application shall ensure that the

corresponding argument is a pointer to a pointer to a **char**.

35661 If an l (ell) qualifier is present, the input is a sequence of characters that begins in the

35662 initial shift state. Each character in the sequence shall be converted to a wide character

35663 as if by a call to the *mbrtowc*() function, with the conversion state described by an

35664 **mbstate\_t** object initialized to zero before the first character is converted. If the 'm'

35665 assignment-allocation character is not specified, the application shall ensure that the

35666 corresponding argument is a pointer to an array of **wchar\_t** large enough to accept the

35667 sequence and the terminating null wide character, which shall be added automatically.

35668 **CX** Otherwise, the application shall ensure that the corresponding argument is a pointer to

35669 a pointer to a **wchar\_t**.

35670 The conversion specification includes all subsequent bytes in the format string up to

35671 and including the matching <right-square-bracket> (' ] '). The bytes between the

35672 square brackets (the *scanlist*) comprise the *scanset*, unless the byte after the <left-

35673 square-bracket> is a <circumflex> (' ^ '), in which case the *scanset* contains all bytes

35674 that do not appear in the *scanlist* between the <circumflex> and the <right-square-

35675 bracket>. If the conversion specification begins with "[ ]" or "[ ^ ]", the <right-

35676 square-bracket> is included in the *scanlist* and the next <right-square-bracket> is the

35677 matching <right-square-bracket> that ends the conversion specification; otherwise, the



35678		first <right-square-bracket> is the one that ends the conversion specification. If a '-' is in the scanlist and is not the first character, nor the second where the first character is a '^', nor the last character, the behavior is implementation-defined.
35679		
35680		
35681	c	Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). No null byte is added. The normal skip over white-space bytes shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .
35682		
35683		
35684		
35685		
35686	CX	
35687		
35688		If an l (ell) qualifier is present, the input shall be a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. No null wide character is added. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the resulting sequence of wide characters. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
35689		
35690		
35691		
35692		
35693		
35694	CX	
35695		
35696		
35697	p	Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding <i>fprintf()</i> functions. The application shall ensure that the corresponding argument is a pointer to a pointer to <b>void</b> . The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion specification is undefined.
35698		
35699		
35700		
35701		
35702		
35703		
35704	n	No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the input so far by this call to the <i>fscanf()</i> functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.
35705		
35706		
35707		
35708		
35709		
35710		
35711	XSI	<b>C</b> Equivalent to <b>lc</b> .
35712	XSI	<b>S</b> Equivalent to <b>ls</b> .
35713	%	Matches a single '%' character; no conversion or assignment occurs. The complete conversion specification shall be %%.
35714		
35715		If a conversion specification is invalid, the behavior is undefined.
35716		The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to a, e, f, g, and x, respectively.
35717		
35718		If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for %n) have been read (other than leading white-space bytes, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.
35719		
35720		
35721		
35722		
35723		

35724 Reaching the end of the string in *sscanf()* shall be equivalent to encountering end-of-file for  
35725 *fscanf()*.

35726 If conversion terminates on a conflicting input, the offending input is left unread in the input.  
35727 Any trailing white-space bytes (including <newline> characters) shall be left unread unless  
35728 matched by a conversion specification. The success of literal matches and suppressed  
35729 assignments is only directly determinable via the %n conversion specification.

35730 CX The *fscanf()* and *scanf()* functions may mark the last data access timestamp of the file associated  
35731 with *stream* for update. The last data access timestamp shall be marked for update by the first  
35732 successful execution of *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *fscanf()*, or  
35733 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

#### 35734 RETURN VALUE

35735 Upon successful completion, these functions shall return the number of successfully matched  
35736 and assigned input items; this number can be zero in the event of an early matching failure. If  
35737 the input ends before the first conversion (if any) has completed, and without a matching failure  
35738 having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has  
35739 CX completed, and without a matching failure having occurred, EOF shall be returned and *errno*  
35740 shall be set to indicate the error. If an error occurs, the error indicator for the stream shall be set.

#### 35741 ERRORS

35742 For the conditions under which the *fscanf()* functions fail and may fail, refer to *fgetc()* or  
35743 *fgetwc()*.

35744 In addition, the *fscanf()* function shall fail if:

35745 CX [EILSEQ] Input byte sequence does not form a valid character.

35746 [ENOMEM] Insufficient storage space is available.

35747 In addition, the *fscanf()* function may fail if:

35748 CX [EINVAL] There are insufficient arguments.

#### 35749 EXAMPLES

35750 The call:

```
35751 int i, n; float x; char name[50];
35752 n = scanf("%d%f%s", &i, &x, name);
```

35753 with the input line:

```
35754 25 54.32E-1 Hamster
```

35755 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
35756 "Hamster".

35757 The call:

```
35758 int i; float x; char name[50];
35759 (void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

35760 with input:

```
35761 56789 0123 56a72
```

35762 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to  
35763 *getchar()* shall return the character 'a'.

35764 **Reading Data into an Array**

35765 The following call uses *fscanf()* to read three floating-point numbers from standard input into  
 35766 the *input* array.

```
35767 float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

35768 **APPLICATION USAGE**

35769 If the application calling *fscanf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include  
 35770 the **<wchar.h>** header to have these objects defined.

35771 For functions that allocate memory as if by *malloc()*, the application should release such memory  
 35772 when it is no longer required by a call to *free()*. For *fscanf()*, this is memory allocated via use of  
 35773 the 'm' assignment-allocation character.

35774 **RATIONALE**

35775 The set of characters allowed in a scanset is limited to single-byte characters. In other similar  
 35776 places, multi-byte characters have been permitted, but for alignment with the ISO C standard, it  
 35777 has not been done here. Applications needing this could use the corresponding wide-character  
 35778 functions to achieve the desired results.

35779 **FUTURE DIRECTIONS**

35780 None.

35781 **SEE ALSO**

35782 [Section 2.5](#) (on page 521), *fprintf()*, *getc()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*

35783 [XBD Chapter 7](#) (on page 127), **<inttypes.h>**, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

35784 **CHANGE HISTORY**

35785 First released in Issue 1. Derived from Issue 1 of the SVID.

35786 **Issue 5**

35787 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the **l** (ell) qualifier is  
 35788 now defined for the **c**, **s**, and **[** conversion specifiers.

35789 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the  
 35790 *fprintf()* family of functions, then they are recognized by the *fscanf()* family.

35791 **Issue 6**

35792 The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several  
 35793 occurrences of “characters” in the text which have been replaced with the term “bytes”.

35794 The normative text is updated to avoid use of the term “must” for application requirements.

35795 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 35796 • The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.
- 35797 • The DESCRIPTION is updated.
- 35798 • The **hh**, **ll**, **j**, **t**, and **z** length modifiers are added.
- 35799 • The **a**, **A**, and **F** conversion characters are added.

35800 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
 35801 specification” consistently.

35802 **Issue 7**

- 35803 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 35804 SD5-XSH-ERN-9 is applied, correcting *fscanf()* to *scanf()* in the DESCRIPTION.
- 35805 SD5-XSH-ERN-132 is applied, adding the assignment-allocation character 'm'.
- 35806 Functionality relating to the %n\$ form of conversion specification is moved from the XSI option  
35807 to the Base.
- 35808 Changes are made related to support for finegrained timestamps.
- 35809 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
35810 *malloc()*.
- 35811 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0185 [302], XSH/TC1-2008/0186 [90],  
35812 and XSH/TC1-2008/0187 [14] are applied. XSH/TC1-2008/0186 [90] changes the second  
35813 sentence in the RETURN VALUE section to align with expected wording changes in the next  
35814 revision of the ISO C standard.
- 35815 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0135 [936] is applied.

35816 **Issue 8**

- 35817 Austin Group Defect 1163 is applied, clarifying the handling of white space in the format string.
- 35818 Austin Group Defect 1173 is applied, clarifying the description of the assignment-allocation  
35819 character 'm'.
- 35820 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
35821 standard.
- 35822 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 35823 Austin Group Defect 1375 is applied, changing “terminating null character” to “terminating null  
35824 character (or wide character)”.
- 35825 Austin Group Defect 1562 is applied, clarifying that it is the application’s responsibility to  
35826 ensure that the format is a character string, beginning and ending in its initial shift state, if any.
- 35827 Austin Group Defect 1624 is applied, changing the RETURN VALUE section.

35828 **NAME**35829 `fseek, fseeko` — reposition a file-position indicator in a stream35830 **SYNOPSIS**35831 `#include <stdio.h>`35832 `int fseek(FILE *stream, long offset, int whence);`35833 CX `int fseeko(FILE *stream, off_t offset, int whence);`35834 **DESCRIPTION**35835 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
35836 conflict between the requirements described here and the ISO C standard is unintentional. This  
35837 volume of POSIX.1-2024 defers to the ISO C standard.35838 The `fseek()` function shall set the file-position indicator for the stream pointed to by `stream`. If a  
35839 read or write error occurs, the error indicator for the stream shall be set and `fseek()` fails.35840 CX The new position, measured in bytes from the beginning of the file, except in the case of streams  
35841 opened with `open_wmemstream()` for which the position shall be measured in wide characters,  
35842 shall be obtained by adding `offset` to the position specified by `whence`. The specified point is the  
35843 beginning of the file for `SEEK_SET`, the current value of the file-position indicator for  
35844 `SEEK_CUR`, or end-of-file for `SEEK_END`.35845 If the stream is to be used with wide-character input/output functions, the application shall  
35846 ensure that `offset` is either 0 or a value returned by an earlier call to `ftell()` on the same stream and  
35847 `whence` is `SEEK_SET`.35848 A successful call to `fseek()` shall clear the end-of-file indicator for the stream and undo any effects  
35849 of `ungetc()` and `ungetwc()` on the same stream. After an `fseek()` call, the next operation on an  
35850 update stream may be either input or output.35851 CX If the most recent operation, other than `ftell()`, on a given stream is `fflush()`, the file offset in the  
35852 underlying open file description shall be adjusted to reflect the location specified by `fseek()`.35853 The `fseek()` function shall allow the file-position indicator to be set beyond the end of existing  
35854 data in the file. If data is later written at this point, subsequent reads of data in the gap shall  
35855 return bytes with the value 0 until data is actually written into the gap.35856 The behavior of `fseek()` on devices which are incapable of seeking is implementation-defined.  
35857 The value of the file offset associated with such a device is undefined.35858 If the stream has an underlying file description and is writable, and buffered data had not been  
35859 written to the underlying file, `fseek()` shall cause the unwritten data to be written to the file and  
35860 shall mark the last data modification and last file status change timestamps of the file for update.35861 If the stream was created by `fmemopen()`, `open_memstream()`, or `open_wmemstream()` and the  
35862 stream is writable, and if the stream is buffered and data in the stream's buffer has not been  
35863 written to the underlying memory buffer, `fseek()` shall cause the unwritten data to be written to  
35864 the underlying memory buffer.35865 In a locale with state-dependent encoding, whether `fseek()` restores the stream's shift state is  
35866 implementation-defined.35867 The `fseeko()` function shall be equivalent to the `fseek()` function except that the `offset` argument is  
35868 of type `off_t`.

35869 **RETURN VALUE**

35870 CX The `fseek()` and `fseeko()` functions shall return 0 if they succeed.

35871 CX Otherwise, they shall return `-1` and set `errno` to indicate the error.

35872 **ERRORS**

35873 CX The `fseek()` and `fseeko()` functions shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to be flushed, and the call to `fseek()` or `fseeko()` causes an underlying `lseek()` or `write()` to be invoked, and:

35876 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor and the thread would be delayed in the write operation.

35878 CX **[EBADF]** The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.

35880 CX **[EFBIG]** An attempt was made to write a file that exceeds the maximum file size.

35881 CX **[EFBIG]** An attempt was made to write a file that exceeds the file size limit of the process.

35882  
35883 XSI A `SIGXFSZ` signal shall also be generated for the thread.

35884 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

35886 CX **[EINTR]** The write operation was terminated due to the receipt of a signal, and no data was transferred.

35888 CX **[EINVAL]** The *whence* argument is invalid, the resulting file-position indicator would be set to a negative value, or the stream was created by `fmemopen()` and the resulting file-position indicator would be beyond the end of the underlying memory buffer.

35892 CX **[EIO]** A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a `write()` to its controlling terminal, `TOSTOP` is set, the calling thread is not blocking `SIGTTOU`, the process is not ignoring `SIGTTOU`, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

35897 CX **[ENOMEM]** The stream was created by `open_memstream()` or `open_wmemstream()` and insufficient memory is available.

35899 CX **[ENOSPC]** There was no free space remaining on the device containing the file or in the buffer used by the `fmemopen()` function.

35901 CX **[EOVERFLOW]** For `fseek()`, the resulting file offset would be a value which cannot be represented correctly in an object of type **long**.

35903 CX **[EOVERFLOW]** For `fseeko()`, the resulting file offset would be a value which cannot be represented correctly in an object of type **off\_t**.

35905 CX **[EPIPE]** An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a `SIGPIPE` signal shall also be sent to the thread.

35907 CX **[ESPIPE]** The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

35908 CX The `fseek()` and `fseeko()` functions may fail if:

35909 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
 35910 capabilities of the device.

### 35911 EXAMPLES

35912 None.

### 35913 APPLICATION USAGE

35914 None.

### 35915 RATIONALE

35916 When the stream was created by *fmemopen()*, *fseek()* fails if an attempt is made to seek beyond  
 35917 the end of the underlying memory buffer. This is different than *fseek()* on a file when a file size  
 35918 limit is in effect because the size specified to *fmemopen()* is a fixed, absolute limit whereas a file  
 35919 size limit is artificial and can be changed. With a file size limit, it is possible to seek past the  
 35920 limit, then raise the limit and successfully write at the new position; there is no equivalent  
 35921 possibility with the buffer size specified to *fmemopen()*.

35922 See also the rationale for *ftell()*.

### 35923 FUTURE DIRECTIONS

35924 None.

### 35925 SEE ALSO

35926 Section 2.5 (on page 521), *fopen()*, *fmemopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*,  
 35927 *open\_memstream()*, *rewind()*, *ungetc()*, *write()*

35928 XBD <stdio.h>

### 35929 CHANGE HISTORY

35930 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 35931 Issue 5

35932 Normative text previously in the APPLICATION USAGE section is moved to the  
 35933 DESCRIPTION.

35934 Large File Summit extensions are added.

#### 35935 Issue 6

35936 Extensions beyond the ISO C standard are marked.

35937 The following new requirements on POSIX implementations derive from alignment with the  
 35938 Single UNIX Specification:

- 35939 • The *fseeko()* function is added.
- 35940 • The [EFBIG], [Eoverflow], and [ENXIO] mandatory error conditions are added.

35941 The following change is incorporated for alignment with the FIPS requirements:

- 35942 • The [EINTR] error is no longer an indication that the implementation does not report  
 35943 partial transfers.

35944 The normative text is updated to avoid use of the term “must” for application requirements.

35945 The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and  
 35946 then on error the error indicator is set and *fseek()* fails. This is for alignment with the  
 35947 ISO/IEC 9899:1999 standard.

35948 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/42 is applied, updating the [EAGAIN]  
 35949 error in the ERRORS section from “the process would be delayed” to “the thread would be  
 35950 delayed”.

35951 **Issue 7**

35952 Changes are made related to support for finegrained timestamps.

35953 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0188 [79], XSH/TC1-2008/0189 [122],  
35954 XSH/TC1-2008/0190 [225], and XSH/TC1-2008/0191 [14] are applied.

35955 **Issue 8**

35956 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

35957 Austin Group Defect 1027 is applied, specifying that for streams opened with  
35958 *open\_wmemstream()* the position is measured in wide characters, not bytes.

35959 Austin Group Defect 1225 is applied, clarifying the behavior for memory streams.

35960 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
35961 relating to the file size limit for the process.



35962 **NAME**

35963 fsetpos — set current file position

35964 **SYNOPSIS**

35965 #include &lt;stdio.h&gt;

35966 int fsetpos(FILE \*stream, const fpos\_t \*pos);

35967 **DESCRIPTION**

35968 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 35969 conflict between the requirements described here and the ISO C standard is unintentional. This  
 35970 volume of POSIX.1-2024 defers to the ISO C standard.

35971 The *fsetpos()* function shall set the **mbstate\_t** object (if any) and file position indicator for the  
 35972 stream pointed to by *stream* according to the value of the object pointed to by *pos*, which the  
 35973 application shall ensure is a value obtained from an earlier call to *fgetpos()* on the same stream. If  
 35974 a read or write error occurs, the error indicator for the stream shall be set and *fsetpos()* fails.

35975 A successful call to the *fsetpos()* function shall undo any effects of the *ungetc()* function on the  
 35976 stream, clear the end-of-file indicator for the stream, and then establish the new parse state and  
 35977 position. After a successful *fsetpos()* call, the next operation on an update stream can be either  
 35978 input or output.

35979 CX The behavior of *fsetpos()* on devices which are incapable of seeking is implementation-defined.  
 35980 The value of the file offset associated with such a device is undefined.

35981 If the stream has an underlying file description and is writable, and buffered data has not been  
 35982 written to the underlying file, *fsetpos()* shall cause the unwritten data to be written to the file and  
 35983 shall mark the last data modification and last file status change timestamps of the file for update.

35984 If the stream was created by *fmemopen()*, *open\_memstream()*, or *open\_wmemstream()* and the  
 35985 stream is writable, and if the stream is buffered and data in the stream's buffer has not been  
 35986 written to the underlying memory buffer, *fsetpos()* shall cause the unwritten data to be written to  
 35987 the underlying memory buffer.

35988 The *fsetpos()* function shall not change the setting of *errno* if successful.

35989 **RETURN VALUE**

35990 The *fsetpos()* function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and  
 35991 set *errno* to indicate the error.

35992 **ERRORS**

35993 CX The *fsetpos()* function shall fail if, either the *stream* is unbuffered or the *stream's* buffer needed to  
 35994 be flushed, and the call to *fsetpos()* causes an underlying *lseek()* or *write()* to be invoked, and:

35995 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the thread would be  
 35996 delayed in the write operation.

35997 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the  
 35998 stream's buffer needed to be flushed and the file is not open.

35999 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

36000 CX [EFBIG] An attempt was made to write a file that exceeds the file size limit of the  
 36001 process.

36002 XSI A SIGXFSZ signal shall also be generated for the thread.

36003 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the  
 36004 offset maximum associated with the corresponding stream.

36005	CX	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
36006			
36007	CX	[EIO]	A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
36008			
36009			
36010			
36011			
36012	CX	[ENOMEM]	The stream was created by <i>open_memstream()</i> or <i>open_wmemstream()</i> and insufficient memory is available.
36013			
36014	CX	[ENOSPC]	There was no free space remaining on the device containing the file or in the buffer used by the <i>fmemopen()</i> function.
36015			
36016	CX	[EPIPE]	An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.
36017			
36018	CX	[ESPIPE]	The file descriptor underlying <i>stream</i> is associated with a pipe, FIFO, or socket.
36019			The <i>fsetpos()</i> function may fail if:
36020	CX	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
36021			

**36022 EXAMPLES**

36023 None.

**36024 APPLICATION USAGE**

36025 None.

**36026 RATIONALE**

36027 The ERRORS section does not include an [EINVAL] error equivalent to the one for *fseek()*  
 36028 because applications are required to obtain the **fpos\_t** value using *fgetpos()*, in which case the  
 36029 file position to be set will always be valid. Directly manipulating the **fpos\_t** object to set a  
 36030 position results in undefined behavior. However, if an implementation detects that the  
 36031 requested file position would be a negative value, or would be beyond the end of the underlying  
 36032 memory buffer of a stream that was created by *fmemopen()*, it is recommended that *fsetpos()*  
 36033 returns a non-zero value and sets *errno* to [EINVAL].

**36034 FUTURE DIRECTIONS**

36035 None.

**36036 SEE ALSO**

36037 [Section 2.5](#) (on page 521), *fopen()*, *fmemopen()*, *ftell()*, *lseek()*, *open\_memstream()*, *rewind()*,  
 36038 *ungetc()*, *write()*

36039 XBD [<stdio.h>](#)

**36040 CHANGE HISTORY**

36041 First released in Issue 4. Derived from the ISO C standard.

**36042 Issue 6**

36043 Extensions beyond the ISO C standard are marked.

36044 The normative text is updated to avoid use of the term “must” for application requirements.

36045 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or  
 36046 write error. This is for alignment with the ISO/IEC 9899:1999 standard.

36047 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous  
36048 [EINVAL] error case from the ERRORS section.

36049 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/43 is applied, updating the [EAGAIN]  
36050 error in the ERRORS section from ``the process would be delayed'' to ``the thread would be  
36051 delayed''.

36052 **Issue 7**

36053 SD5-XSH-ERN-220 is applied.

36054 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0192 [105], XSH/TC1-2008/0193 [79],  
36055 XSH/TC1-2008/0194 [225], XSH/TC1-2008/0195 [450], XSH/TC1-2008/0196 [450], and  
36056 XSH/TC1-2008/0197 [14] are applied.

36057 **Issue 8**

36058 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

36059 Austin Group Defect 1225 is applied, aligning the CX requirements with *fseek()*.

36060 Austin Group Defect 1249 is applied, correcting some text mismatches with the ISO C standard.

36061 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
36062 relating to the file size limit for the process.

36063 **NAME**36064 `fstat` — get file status36065 **SYNOPSIS**36066 `#include <sys/stat.h>`36067 `int fstat(int fildev, struct stat *buf);`36068 **DESCRIPTION**36069 The `fstat()` function shall obtain information about an open file associated with the file  
36070 descriptor *fildev*, and shall write it to the area pointed to by *buf*.36071 SHM If *fildev* references a shared memory object, the implementation shall update in the **stat** structure  
36072 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
36073 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
36074 valid. The implementation may update other fields and flags.36075 TYM If *fildev* references a typed memory object, the implementation shall update in the **stat** structure  
36076 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
36077 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
36078 valid. The implementation may update other fields and flags.36079 The *buf* argument is a pointer to a **stat** structure, as defined in `<sys/stat.h>`, into which  
36080 information is placed concerning the file.36081 For all other file types defined in this volume of POSIX.1-2024, the structure members *st\_mode*,  
36082 *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the  
36083 value of the *st\_nlink* member shall be set to the number of links to the file.36084 An implementation that provides additional or alternative file access control mechanisms may,  
36085 under implementation-defined conditions, cause `fstat()` to fail.36086 The `fstat()` function shall update any time-related fields (as described in XBD [Section 4.12](#), on  
36087 page 98), before writing into the **stat** structure.36088 **RETURN VALUE**36089 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
36090 indicate the error.36091 **ERRORS**36092 The `fstat()` function shall fail if:36093 [EBADF] The *fildev* argument is not a valid file descriptor.

36094 [EIO] An I/O error occurred while reading from the file system.

36095 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
36096 serial number cannot be represented correctly in the structure pointed to by  
36097 *buf*.36098 The `fstat()` function may fail if:36099 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*  
36100 argument.

36101 **EXAMPLES**36102 **Obtaining File Status Information**

36103 The following example shows how to obtain file status information for a file named  
 36104 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The  
 36105 **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file  
 36106 descriptor *fildev*.

```
36107 #include <sys/types.h>
36108 #include <sys/stat.h>
36109 #include <fcntl.h>

36110 struct stat buffer;
36111 int      status;
36112 ...
36113 fildev = open("/home/cnd/mod1", O_RDWR);
36114 status = fstat(fildev, &buffer);
```

36115 **APPLICATION USAGE**

36116 None.

36117 **RATIONALE**

36118 None.

36119 **FUTURE DIRECTIONS**

36120 None.

36121 **SEE ALSO**

36122 [fstatat\(\)](#)

36123 XBD Section 4.12 (on page 98), [<sys/stat.h>](#), [<sys/types.h>](#)

36124 **CHANGE HISTORY**

36125 First released in Issue 1. Derived from Issue 1 of the SVID.

36126 **Issue 5**

36127 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

36128 Large File Summit extensions are added.

36129 **Issue 6**

36130 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

36131 The following new requirements on POSIX implementations derive from alignment with the  
 36132 Single UNIX Specification:

- 36133 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 36134 required for conforming implementations of previous POSIX specifications, it was not  
 36135 required for UNIX applications.
- 36136 • The [EIO] mandatory error condition is added.
- 36137 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 36138 files.
- 36139 • The [EOVERFLOW] optional error condition is added.

36140 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 36141 shared memory object semantics apply to typed memory objects.

36142 **Issue 7**

36143 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st\_nlink*  
36144 applies.

36145 Changes are made related to support for finegrained timestamps.

36146 **NAME**

36147 fstatat, lstat, stat — get file status

36148 **SYNOPSIS**

36149 OH #include &lt;fcntl.h&gt;

36150 #include &lt;sys/stat.h&gt;

36151 int fstatat(int fd, const char \*restrict path,

36152 struct stat \*restrict buf, int flag);

36153 int lstat(const char \*restrict path, struct stat \*restrict buf);

36154 int stat(const char \*restrict path, struct stat \*restrict buf);

36155 **DESCRIPTION**

36156 The *stat()* function shall obtain information about the named file and write it to the area pointed  
 36157 to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or  
 36158 execute permission of the named file is not required. An implementation that provides  
 36159 additional or alternate file access control mechanisms may, under implementation-defined  
 36160 conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file  
 36161 specified by *path*.

36162 If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using  
 36163 the contents of the symbolic link, and shall return information pertaining to the resulting file if  
 36164 the file exists.

36165 The *buf* argument is a pointer to a **stat** structure, as defined in the <sys/stat.h> header, into  
 36166 which information is placed concerning the file.

36167 The *stat()* function shall update any time-related fields (as described in XBD Section 4.12, on  
 36168 page 98), before writing into the **stat** structure.

36169 SHM If the named file is a shared memory object, the implementation shall update in the **stat** structure  
 36170 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
 36171 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
 36172 valid. The implementation may update other fields and flags.

36173 TYM If the named file is a typed memory object, the implementation shall update in the **stat** structure  
 36174 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
 36175 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
 36176 valid. The implementation may update other fields and flags.

36177 For all other file types defined in this volume of POSIX.1-2024, the structure members *st\_mode*,  
 36178 *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the  
 36179 value of the member *st\_nlink* shall be set to the number of hard links to the file.

36180 The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In  
 36181 that case *lstat()* shall return information about the link, while *stat()* shall return information  
 36182 about the file the link references.

36183 For symbolic links, the *st\_mode* member shall contain meaningful information when used with  
 36184 the file type macros. The file mode bits in *st\_mode* are unspecified. The structure members *st\_ino*,  
 36185 *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the value of  
 36186 the *st\_nlink* member shall be set to the number of hard links to the symbolic link. The value of  
 36187 the *st\_size* member shall be set to the length of the pathname contained in the symbolic link not  
 36188 including any terminating null byte.

36189 The *fstatat()* function shall be equivalent to the *stat()* or *lstat()* function, depending on the value  
 36190 of *flag* (see below), except in the case where *path* specifies a relative path. In this case the status  
 36191 shall be retrieved from a file relative to the directory associated with the file descriptor *fd* instead

36192 of the current working directory. If the access mode of the open file description associated with  
 36193 the file descriptor is not O\_SEARCH, the function shall check whether directory searches are  
 36194 permitted using the current permissions of the directory underlying the file descriptor. If the  
 36195 access mode is O\_SEARCH, the function shall not perform the check.

36196 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 36197 in **<fcntl.h>**:

36198 AT\_SYMLINK\_NOFOLLOW

36199 If *path* names a symbolic link, the status of the symbolic link is returned.

36200 If *fstatat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 36201 directory shall be used and the behavior shall be identical to a call to *stat()* or *lstat()* respectively,  
 36202 depending on whether or not the AT\_SYMLINK\_NOFOLLOW bit is set in *flag*.

### 36203 RETURN VALUE

36204 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 36205 return -1 and set *errno* to indicate the error.

### 36206 ERRORS

36207 These functions shall fail if:

36208 [EACCES] Search permission is denied for a component of the path prefix.

36209 [EIO] An error occurred while reading from the file system.

36210 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 36211 argument.

36212 [ENAMETOOLONG]

36213 The length of a component of a pathname is longer than {NAME\_MAX}.

36214 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

36215 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
 36216 directory nor a symbolic link to a directory, or the *path* argument contains at  
 36217 least one non-*<slash>* character and ends with one or more trailing *<slash>*  
 36218 characters and the last pathname component names an existing file that is  
 36219 neither a directory nor a symbolic link to a directory.

36220 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
 36221 serial number cannot be represented correctly in the structure pointed to by  
 36222 *buf*.

36223 The *fstatat()* function shall fail if:

36224 [EACCES] The access mode of the open file description associated with *fd* is not  
 36225 O\_SEARCH and the permissions of the directory underlying *fd* do not permit  
 36226 directory searches.

36227 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
 36228 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.

36229 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated  
 36230 with a non-directory file.

36231 These functions may fail if:

36232 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 36233 resolution of the *path* argument.



- 36234 [ENAMETOOLONG]  
 36235           The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 36236           symbolic link produced an intermediate result with a length that exceeds  
 36237           {PATH\_MAX}.
- 36238 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.  
 36239 The *fstatat()* function may fail if:
- 36240 [EINVAL]       The value of the *flag* argument is not valid.

## 36241 EXAMPLES

### 36242 Obtaining File Status Information

36243 The following example shows how to obtain file status information for a file named  
 36244 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
36245 #include <sys/types.h>
36246 #include <sys/stat.h>
36247 #include <fcntl.h>
36248
36249 struct stat buffer;
36249 int      status;
36250 ...
36251 status = stat("/home/cnd/mod1", &buffer);
```

### 36252 Getting Directory Information

36253 The following example fragment gets status information for each entry in a directory. The call to  
 36254 the *stat()* function stores file information in the **stat** structure pointed to by *statbuf*. The lines  
 36255 that follow the *stat()* call format the fields in the **stat** structure for presentation to the user of the  
 36256 program.

```
36257 #include <sys/types.h>
36258 #include <sys/stat.h>
36259 #include <dirent.h>
36260 #include <pwd.h>
36261 #include <grp.h>
36262 #include <time.h>
36263 #include <locale.h>
36264 #include <langinfo.h>
36265 #include <stdio.h>
36266 #include <stdint.h>
36267
36268 struct dirent *dp;
36268 struct stat  statbuf;
36269 struct passwd *pwd;
36270 struct group  *grp;
36271 struct tm     *tm;
36272 char          datestring[256];
36273 ...
36274 /* Loop through directory entries. */
36275 while ((dp = readdir(dir)) != NULL) {
36276
36276     /* Get entry's information. */
36277     if (stat(dp->d_name, &statbuf) == -1)
```

```

36278         continue;
36279         /* Print out type, permissions, and number of links. */
36280         printf("%10.10s", mode_string(statbuf.st_mode));
36281         printf(" %4ju", (uintmax_t)statbuf.st_nlink);
36282         /* Print out owner's name if it is found using getpwuid(). */
36283         if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
36284             printf(" %-8.8s", pwd->pw_name);
36285         else
36286             printf(" %-8ju", (uintmax_t)statbuf.st_uid);
36287         /* Print out group name if it is found using getgrgid(). */
36288         if ((grp = getgrgid(statbuf.st_gid)) != NULL)
36289             printf(" %-8.8s", grp->gr_name);
36290         else
36291             printf(" %-8ju", (uintmax_t)statbuf.st_gid);
36292         /* Print size of file. */
36293         printf(" %9jd", (intmax_t)statbuf.st_size);
36294         tm = localtime(&statbuf.st_mtime);
36295         /* Get localized date string. */
36296         strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
36297         printf(" %s %s\n", datestring, dp->d_name);
36298     }

```

### 36299 Obtaining Symbolic Link Status Information

36300 The following example shows how to obtain status information for a symbolic link named  
36301 **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path*  
36302 argument specified the pathname for the file pointed to by the symbolic link (**/home/cnd/mod1**),  
36303 the results of calling the function would be the same as those returned by a call to the *stat()*  
36304 function.

```

36305 #include <sys/stat.h>
36306 struct stat buffer;
36307 int status;
36308 ...
36309 status = lstat("/modules/pass1", &buffer);

```

### 36310 APPLICATION USAGE

36311 None.

### 36312 RATIONALE

36313 The intent of the paragraph describing “additional or alternate file access control mechanisms”  
36314 is to allow a secure implementation where a process with a label that does not dominate the  
36315 file’s label cannot perform a *stat()* function. This is not related to read permission; a process with  
36316 a label that dominates the file’s label does not need read permission. An implementation that  
36317 supports write-up operations could fail *fstat()* function calls even though it has a valid file  
36318 descriptor open for writing.

36319 The purpose of the *fstatat()* function is to obtain the status of files in directories other than the  
36320 current working directory without exposure to race conditions. Any part of the path of a file  
36321 could be changed in parallel to a call to *stat()*, resulting in unspecified behavior. By opening a

36322 file descriptor for the target directory and using the *fstatat()* function it can be guaranteed that  
 36323 the file for which status is returned is located relative to the desired directory.

#### 36324 FUTURE DIRECTIONS

36325 None.

#### 36326 SEE ALSO

36327 *access()*, *chmod()*, *fdopendir()*, *fstat()*, *mknod()*, *readlink()*, *symlink()*

36328 XBD Section 4.12 (on page 98), [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)

#### 36329 CHANGE HISTORY

36330 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 36331 Issue 5

36332 Large File Summit extensions are added.

#### 36333 Issue 6

36334 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

36335 The following new requirements on POSIX implementations derive from alignment with the  
 36336 Single UNIX Specification:

- 36337 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 36338 required for conforming implementations of previous POSIX specifications, it was not  
 36339 required for UNIX applications.
- 36340 • The [EIO] mandatory error condition is added.
- 36341 • The [ELOOP] mandatory error condition is added.
- 36342 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 36343 files.
- 36344 • The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are  
 36345 added.

36346 The following changes were made to align with the IEEE P1003.1a draft standard:

- 36347 • Details are added regarding the treatment of symbolic links.
- 36348 • The [ELOOP] optional error condition is added.

36349 The normative text is updated to avoid use of the term “must” for application requirements.

36350 The **restrict** keyword is added to the *stat()* prototype for alignment with the ISO/IEC 9899:1999  
 36351 standard.

#### 36352 Issue 7

36353 Austin Group Interpretation 1003.1-2001 #143 is applied.

36354 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st\_nlink*  
 36355 applies.

36356 The *fstatat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 36357 Set Part 2.

36358 Changes are made related to support for finegrained timestamps.

36359 The *lstat()* function is now required to return meaningful data for symbolic links in all **stat**  
 36360 structure fields, except for the permission bits of *st\_mode*.

36361 Changes are made to allow a directory to be opened for searching.

36362 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
36363 pathname exists but is not a directory or a symbolic link to a directory.

36364 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0198 [461], XSH/TC1-2008/0199 [324],  
36365 XSH/TC1-2008/0200 [278], XSH/TC1-2008/0201 [278], and XSH/TC1-2008/0202 [291] are  
36366 applied.

36367 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0136 [591], XSH/TC2-2008/0137 [817],  
36368 XSH/TC2-2008/0138 [817], and XSH/TC2-2008/0139 [889] are applied.

36369 **Issue 8**

36370 Austin Group Defect 1380 is applied, changing text using the term “link” in line with its  
36371 updated definition.

36372 Austin Group Defect 1409 is applied, changing the EXAMPLES section.

36373 **NAME**

36374 fstatvfs, statvfs — get file system information

36375 **SYNOPSIS**

36376 #include &lt;sys/statvfs.h&gt;

36377 int fstatvfs(int *fildev*, struct statvfs \**buf*);36378 int statvfs(const char \*restrict *path*, struct statvfs \*restrict *buf*);36379 **DESCRIPTION**36380 The *fstatvfs()* function shall obtain information about the file system containing the file  
36381 referenced by *fildev*.36382 The *statvfs()* function shall obtain information about the file system containing the file named by  
36383 *path*.36384 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read,  
36385 write, or execute permission of the named file is not required.36386 The following flags can be returned in the *f\_flag* member:

36387 ST\_RDONLY Read-only file system.

36388 ST\_NOSUID Setuid/setgid bits ignored by *exec*.36389 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file  
36390 systems.36391 **RETURN VALUE**36392 Upon successful completion, *statvfs()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
36393 indicate the error.36394 **ERRORS**36395 The *fstatvfs()* and *statvfs()* functions shall fail if:

36396 [EIO] An I/O error occurred while reading the file system.

36397 [EINTR] A signal was caught during execution of the function.

36398 [EOVERFLOW] One of the values to be returned cannot be represented correctly in the  
36399 structure pointed to by *buf*.36400 The *fstatvfs()* function shall fail if:36401 [EBADF] The *fildev* argument is not an open file descriptor.36402 The *statvfs()* function shall fail if:

36403 [EACCES] Search permission is denied on a component of the path prefix.

36404 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
36405 argument.

36406 [ENAMETOOLONG]

36407 The length of a component of a pathname is longer than {NAME\_MAX}.

36408 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.36409 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
36410 directory nor a symbolic link to a directory, or the *path* argument contains at  
36411 least one non-*<slash>* character and ends with one or more trailing *<slash>*  
36412 characters and the last pathname component names an existing file that is  
36413 neither a directory nor a symbolic link to a directory.

36414 The *statvfs()* function may fail if:

36415 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
36416 resolution of the *path* argument.

36417 [ENAMETOOLONG]  
36418 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
36419 symbolic link produced an intermediate result with a length that exceeds  
36420 {PATH\_MAX}.

## 36421 EXAMPLES

### 36422 Obtaining File System Information Using *fstatvfs()*

36423 The following example shows how to obtain file system information for the file system upon  
36424 which the file named */home/cnd/mod1* resides, using the *fstatvfs()* function. The  
36425 */home/cnd/mod1* file is opened with read/write privileges and the open file descriptor is passed  
36426 to the *fstatvfs()* function.

```
36427 #include <sys/statvfs.h>
36428 #include <fcntl.h>

36429 struct statvfs buffer;
36430 int status;
36431 ...
36432 fildes = open("/home/cnd/mod1", O_RDWR);
36433 status = fstatvfs(fildes, &buffer);
```

### 36434 Obtaining File System Information Using *statvfs()*

36435 The following example shows how to obtain file system information for the file system upon  
36436 which the file named */home/cnd/mod1* resides, using the *statvfs()* function.

```
36437 #include <sys/statvfs.h>

36438 struct statvfs buffer;
36439 int status;
36440 ...
36441 status = statvfs("/home/cnd/mod1", &buffer);
```

## 36442 APPLICATION USAGE

36443 None.

## 36444 RATIONALE

36445 None.

## 36446 FUTURE DIRECTIONS

36447 None.

## 36448 SEE ALSO

36449 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *futimens()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*,  
36450 *time()*, *unlink()*, *write()*

36451 XBD [<sys/statvfs.h>](#)

36452 **CHANGE HISTORY**

36453 First released in Issue 4, Version 2.

36454 **Issue 5**

36455 Moved from X/OPEN UNIX extension to BASE.

36456 Large File Summit extensions are added.

36457 **Issue 6**

36458 The normative text is updated to avoid use of the term “must” for application requirements.

36459 The **restrict** keyword is added to the *statvfs()* prototype for alignment with the  
36460 ISO/IEC 9899:1999 standard.

36461 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
36462 [ELOOP] error condition is added.

36463 **Issue 7**

36464 Austin Group Interpretation 1003.1-2001 #143 is applied.

36465 SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.

36466 The *fstatvfs()* and *statvfs()* functions are moved from the XSI option to the Base.

36467 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
36468 pathname exists but is not a directory or a symbolic link to a directory.

36469 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0203 [324] is applied.

36470 **NAME**

36471 fsync — synchronize changes to a file

36472 **SYNOPSIS**

```
36473 #include <unistd.h>
36474 int fsync(int fildev);
```

36475 **DESCRIPTION**

36476 The *fsync()* function shall request that all data for the open file descriptor named by *fildev* is to be  
 36477 transferred to the storage device associated with the file described by *fildev*. The nature of the  
 36478 transfer is implementation-defined. The *fsync()* function shall not return until the system has  
 36479 completed that action or until an error is detected.

36480 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued  
 36481 I/O operations associated with the file indicated by file descriptor *fildev* to the synchronized I/O  
 36482 completion state. All I/O operations shall be completed as defined for synchronized I/O file  
 36483 integrity completion.

36484 **RETURN VALUE**

36485 Upon successful completion, *fsync()* shall return 0. Otherwise, `-1` shall be returned and *errno* set  
 36486 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed  
 36487 to have been completed.

36488 **ERRORS**36489 The *fsync()* function shall fail if:

- 36490 [EBADF] The *fildev* argument is not a valid descriptor.
- 36491 [EINTR] The *fsync()* function was interrupted by a signal.
- 36492 [EINVAL] The *fildev* argument does not refer to a file on which this operation is possible.
- 36493 [EIO] An I/O error occurred while reading from or writing to the file system.

36494 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions  
 36495 defined for *read()* and *write()*.

36496 **EXAMPLES**

36497 None.

36498 **APPLICATION USAGE**

36499 The *fsync()* function should be used by programs which require modifications to a file to be  
 36500 completed before continuing; for example, a program which contains a simple transaction  
 36501 facility might use it to ensure that all modifications to a file or files caused by a transaction are  
 36502 recorded.

36503 An application that modifies a directory, for example, by creating a file in the directory, can  
 36504 invoke *fsync()* on the directory to ensure that the directory's entries and file attributes are  
 36505 synchronized. For most applications, synchronizing the directory's entries should not be  
 36506 necessary (see XBD [Section 4.11](#), on page 98).

36507 **RATIONALE**

36508 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to  
 36509 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is  
 36510 recorded on the disk. Since the concepts of "buffer cache", "system crash", "physical write", and  
 36511 "non-volatile storage" are not defined here, the wording has to be more abstract.

36512 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance



36513 document to tell the user what can be expected from the system. It is explicitly intended that a  
36514 null implementation is permitted. This could be valid in the case where the system cannot assure  
36515 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the  
36516 functionality is not required. In the middle ground between these extremes, *fsync()* might or  
36517 might not actually cause data to be written where it is safe from a power failure. The  
36518 conformance document should identify at least that one configuration exists (and how to obtain  
36519 that configuration) where this can be assured for at least some files that the user can select to use  
36520 for critical data. It is not intended that an exhaustive list is required, but rather sufficient  
36521 information is provided so that if critical data needs to be saved, the user can determine how the  
36522 system is to be configured to allow the data to be written to non-volatile storage.

36523 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.  
36524 That does not make the function any less valuable, just more difficult to test. A formal  
36525 conformance test should probably force a system crash (power shutdown) during the test for  
36526 this condition, but it needs to be done in such a way that automated testing does not require this  
36527 to be done except when a formal record of the results is being made. It would also not be  
36528 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation  
36529 issue.

#### 36530 **FUTURE DIRECTIONS**

36531 None.

#### 36532 **SEE ALSO**

36533 [\*sync\(\)\*](#)

36534 XBD <[unistd.h](#)>

#### 36535 **CHANGE HISTORY**

36536 First released in Issue 3.

#### 36537 **Issue 5**

36538 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and  
36539 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate  
36540 that *fsync()* can return the error conditions defined for *read()* and *write()*.

#### 36541 **Issue 6**

36542 This function is marked as part of the File Synchronization option.

36543 The following new requirements on POSIX implementations derive from alignment with the  
36544 Single UNIX Specification:

- 36545 • The [EINVAL] and [EIO] mandatory error conditions are added.

36546 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/44 is applied, applying an editorial  
36547 rewording of the DESCRIPTION. No change in meaning is intended.

#### 36548 **Issue 8**

36549 Austin Group Defect 672 is applied, changing the APPLICATION USAGE section.

36550 **NAME**

36551 ftell, ftello — return a file offset in a stream

36552 **SYNOPSIS**

```
36553 #include <stdio.h>
36554 long ftell(FILE *stream);
36555 CX off_t ftello(FILE *stream);
```

36556 **DESCRIPTION**

36557 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 36558 conflict between the requirements described here and the ISO C standard is unintentional. This  
 36559 volume of POSIX.1-2024 defers to the ISO C standard.

36560 The *ftell()* function shall obtain the current value of the file-position indicator for the stream  
 36561 pointed to by *stream*.

36562 The *ftell()* function shall not change the setting of *errno* if successful.

36563 CX The *ftello()* function shall be equivalent to *ftell()*, except that the return value is of type **off\_t** and  
 36564 the *ftello()* function may change the setting of *errno* if successful.

36565 **RETURN VALUE**

36566 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position  
 36567 CX indicator for the stream measured in bytes from the beginning of the file, except in the case of  
 36568 streams opened with *open\_wmemstream()* for which the position shall be measured in wide  
 36569 characters.

36570 Otherwise, *ftell()* and *ftello()* shall return  $-1$ , and set *errno* to indicate the error.

36571 **ERRORS**

36572 CX The *ftell()* and *ftello()* functions shall fail if:

36573 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.

36574 CX [EOVERFLOW] For *ftell()*, the current file offset cannot be represented correctly in an object of  
 36575 type **long**.

36576 CX [EOVERFLOW] For *ftello()*, the current file offset cannot be represented correctly in an object  
 36577 of type **off\_t**.

36578 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

36579 **EXAMPLES**

36580 None.

36581 **APPLICATION USAGE**

36582 None.

36583 **RATIONALE**

36584 For all streams other than those opened by *open\_wmemstream()*, *ftell()* and *fseek()* operate on  
 36585 byte offsets. The behavior with *open\_wmemstream()* streams is intentionally different—*ftell()* and  
 36586 *fseek()* operate on wide character offsets. This is because those streams are unique in that the  
 36587 backing storage is not a multibyte representation but a wide character array, and it is useful to be  
 36588 able to use the output of *ftell()* to index into that array.

**36589 FUTURE DIRECTIONS**

36590 None.

**36591 SEE ALSO**

36592 [Section 2.5](#) (on page 521), [fgetpos\(\)](#), [fopen\(\)](#), [fseek\(\)](#), [lseek\(\)](#), [open\\_memstream\(\)](#)

36593 XBD [<stdio.h>](#)

**36594 CHANGE HISTORY**

36595 First released in Issue 1. Derived from Issue 1 of the SVID.

**36596 Issue 5**

36597 Large File Summit extensions are added.

**36598 Issue 6**

36599 Extensions beyond the ISO C standard are marked.

36600 The following new requirements on POSIX implementations derive from alignment with the  
36601 Single UNIX Specification:

- 36602 • The *ftello()* function is added.
- 36603 • The [Eoverflow] error conditions are added.

36604 An additional [ESPIPE] error condition is added for sockets.

**36605 Issue 7**

36606 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0204 [105], XSH/TC1-2008/0205 [421],  
36607 XSH/TC1-2008/0206 [122], XSH/TC1-2008/0207 [122], and XSH/TC1-2008/0208 [14] are  
36608 applied.

**36609 Issue 8**

36610 Austin Group Defect 1027 is applied, specifying that for streams opened with  
36611 *open\_wmemstream()* the position is measured in wide characters, not bytes.

36612 **NAME**

36613 ftok — generate an IPC key

36614 **SYNOPSIS**

```
36615 XSI #include <sys/ipc.h>
36616 key_t ftok(const char *path, int id);
```

36617 **DESCRIPTION**

36618 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to  
 36619 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the  
 36620 pathname of an existing file that the process is able to *stat()*, with the exception that if *stat()*  
 36621 would fail with [EOVERFLOW] due to file size, *ftok()* shall still succeed.

36622 The *ftok()* function shall return the same key value for all paths that name the same file, when  
 36623 called with the same *id* value, and should return different key values when called with different  
 36624 *id* values or with paths that name different files existing on the same file system at the same  
 36625 time. It is unspecified whether *ftok()* shall return the same key value when called again after the  
 36626 file named by *path* is removed and recreated with the same name.

36627 Only the low-order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits  
 36628 are 0.

36629 **RETURN VALUE**

36630 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return (**key\_t**)-1  
 36631 and set *errno* to indicate the error.

36632 **ERRORS**

36633 The *ftok()* function shall fail if:

- |       |                |   |
|-------|----------------|---|
| 36634 | [EACCES]       | Search permission is denied for a component of the path prefix.   |
| 36635 | [EIO]          | An error occurred while reading from the file system.   |
| 36636 | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i><br>36637 argument.   |
| 36638 | [ENAMETOOLONG] | 36639 The length of a component of a pathname is longer than {NAME_MAX}.  |
| 36640 | [ENOENT]       | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.  |
| 36641 | [ENOTDIR]      | A component of the path prefix names an existing file that is neither a<br>36642 directory nor a symbolic link to a directory, or the <i>path</i> argument contains at<br>36643 least one non- <i>&lt;slash&gt;</i> character and ends with one or more trailing <i>&lt;slash&gt;</i><br>36644 characters and the last pathname component names an existing file that is<br>36645 neither a directory nor a symbolic link to a directory. |

36646 The *ftok()* function may fail if:

- |       |                |   |
|-------|----------------|---|
| 36647 | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during<br>36648 resolution of the <i>path</i> argument.   |
| 36649 | [ENAMETOOLONG] | 36650 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a<br>36651 symbolic link produced an intermediate result with a length that exceeds<br>36652 {PATH_MAX}. |

36653 **EXAMPLES**36654 **Getting an IPC Key**

36655 The following example gets a key based on the pathname `/tmp` and the ID value `a`. It also  
 36656 assigns the value of the resulting key to the `semkey` variable so that it will be available to a later  
 36657 call to `semget()`, `msgget()`, or `shmget()`.

```
36658 #include <sys/ipc.h>
36659 ...
36660 key_t semkey;
36661 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
36662     perror("IPC error: ftok"); exit(1);
36663 }
```

36664 **APPLICATION USAGE**

36665 For maximum portability, `id` should be a single-byte character.

36666 Applications should not assume that the resulting key value is unique.

36667 **RATIONALE**

36668 None.

36669 **FUTURE DIRECTIONS**

36670 Future versions of this standard may add new interfaces to provide unique keys.

36671 **SEE ALSO**

36672 [\*msgget\(\)\*](#), [\*semget\(\)\*](#), [\*shmget\(\)\*](#)

36673 XBD [\*<sys/ipc.h>\*](#)

36674 **CHANGE HISTORY**

36675 First released in Issue 4, Version 2.

36676 **Issue 5**

36677 Moved from X/OPEN UNIX extension to BASE.

36678 **Issue 6**

36679 The normative text is updated to avoid use of the term “must” for application requirements.

36680 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 36681 [ELOOP] error condition is added.

36682 **Issue 7**

36683 Austin Group Interpretation 1003.1-2001 #143 is applied.

36684 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
 36685 pathname exists but is not a directory or a symbolic link to a directory.

36686 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0209 [343], XSH/TC1-2008/0210 [366],  
 36687 XSH/TC1-2008/0211 [343], XSH/TC1-2008/0212 [324], XSH/TC1-2008/0213 [366],  
 36688 XSH/TC1-2008/0214 [366], XSH/TC1-2008/0215 [366], and XSH/TC1-2008/0216 [366] are  
 36689 applied.

36690 **NAME**

36691 ftruncate — truncate a file to a specified length

36692 **SYNOPSIS**

36693 #include &lt;unistd.h&gt;

36694 int ftruncate(int *fildev*, off\_t *length*);36695 **DESCRIPTION**36696 If *fildev* is not a valid file descriptor open for writing, the *ftruncate()* function shall fail.

36697 If *fildev* refers to a regular file, the *ftruncate()* function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate()* shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate()*.

36703 Upon successful completion, if *fildev* refers to a regular file, *ftruncate()* shall mark for update the last data modification and last file status change timestamps of the file and the S\_ISUID and S\_ISGID bits of the file mode may be cleared. If the *ftruncate()* function is unsuccessful, the file is unaffected.

36707 If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate a SIGXFSZ signal for the thread.

36709 If *fildev* is a file descriptor open for writing and refers to a file that is neither a regular file nor a shared memory object, the result is unspecified.

36711 SHM If *fildev* refers to a shared memory object, *ftruncate()* shall set the size of the shared memory object to *length*.

36713 SHM If the effect of *ftruncate()* is to decrease the size of a memory mapped file or a shared memory object and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

36716 References to discarded pages shall result in the generation of a SIGBUS signal.

36717 If the effect of *ftruncate()* is to increase the size of a memory object, it is unspecified whether the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

36720 **RETURN VALUE**

36721 Upon successful completion, *ftruncate()* shall return 0; otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

36723 **ERRORS**

36724 The *ftruncate()* function shall fail if:

36725 [EBADF] or [EINVAL]

36726 The *fildev* argument is not a file descriptor open for writing.

36727 [EFBIG] or [EINVAL]

36728 The *length* argument is greater than the maximum file size.

36729 XSI [EFBIG] The *length* argument exceeds the file size limit of the process. A SIGXFSZ signal shall also be generated for the thread.

36731 [EFBIG] The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fildev*.

36733	[EINTR]	A signal was caught during execution.
36734	[EINVAL]	The <i>length</i> argument is less than 0 or the <i>files</i> argument refers to a file on which this operation is not possible (for example, a pipe, FIFO or socket).
36735		
36736	[EIO]	An I/O error occurred while reading from or writing to a file system.
36737	<b>EXAMPLES</b>	
36738	None.	
36739	<b>APPLICATION USAGE</b>	
36740	None.	
36741	<b>RATIONALE</b>	
36742	None.	
36743	<b>FUTURE DIRECTIONS</b>	
36744	None.	
36745	<b>SEE ALSO</b>	
36746	<i>open()</i> , <i>truncate()</i>	
36747	XBD < <b>unistd.h</b> >	
36748	<b>CHANGE HISTORY</b>	
36749	First released in Issue 4, Version 2.	
36750	<b>Issue 5</b>	
36751	Moved from X/OPEN UNIX extension to BASE and aligned with <i>ftruncate()</i> in the POSIX	
36752	Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is	
36753	added to the list of mandatory errors that can be returned by <i>ftruncate()</i> .	
36754	Large File Summit extensions are added.	
36755	<b>Issue 6</b>	
36756	The <i>truncate()</i> function is split out into a separate reference page.	
36757	The following new requirements on POSIX implementations derive from alignment with the	
36758	Single UNIX Specification:	
36759	• The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a	
36760	regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.	
36761	The following changes were made to align with the IEEE P1003.1a draft standard:	
36762	• The DESCRIPTION text is updated.	
36763	XSI-conformant systems are required to increase the size of the file if the file was previously	
36764	smaller than the size requested.	
36765	<b>Issue 7</b>	
36766	Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although	
36767	the [EINVAL] “may fail” error was subsequently removed during review of the XSI option).	
36768	Functionality relating to the Memory Protection and Memory Mapped Files options is moved to	
36769	the Base.	
36770	The DESCRIPTION is updated so that a call to <i>ftruncate()</i> when the file is smaller than the size	
36771	requested will increase the size of the file. Previously, non-XSI-conforming implementations	
36772	were allowed to increase the size of the file or fail.	
36773	Changes are made related to support for finegrained timestamps.	

36774 **Issue 8**

36775 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

36776 Austin Group Defect 1169 is applied, removing a redundant statement that *ftruncate()* fails if  
36777 *files* refers to a directory.

36778 Austin Group Defect 1381 is applied, adding a second condition to the [EINVAL] error and  
36779 rearranging the ERRORS section into alphabetical order.

36780 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
36781 relating to the file size limit for the process.



36782 **NAME**

36783 ftrylockfile — stdio locking functions

36784 **SYNOPSIS**

```
36785 CX #include <stdio.h>  
36786 int ftrylockfile(FILE *file);
```

36787 **DESCRIPTION**36788 Refer to *flockfile()*.

36789 **NAME**

36790 funlockfile — stdio locking functions

36791 **SYNOPSIS**

```
36792 CX #include <stdio.h>  
36793 void funlockfile(FILE *file);
```

36794 **DESCRIPTION**

36795 Refer to *flockfile()*.

36796 **NAME**

36797 futimens, utimensat, utimes — set file access and modification times

36798 **SYNOPSIS**

36799 #include &lt;sys/stat.h&gt;

36800 int futimens(int *fd*, const struct timespec *times*[2]);

36801 OH #include &lt;fcntl.h&gt;

36802 int utimensat(int *fd*, const char \**path*, const struct timespec *times*[2],  
36803 int *flag*);

36804 XSI #include &lt;sys/time.h&gt;

36805 int utimes(const char \**path*, const struct timeval *times*[2]);36806 **DESCRIPTION**

36807 The *futimens()* and *utimensat()* functions shall set the access and modification times of a file to  
 36808 the values of the *times* argument. The *futimens()* function changes the times of the file associated  
 36809 with the file descriptor *fd*. The *utimensat()* function changes the times of the file pointed to by  
 36810 the *path* argument, relative to the directory associated with the file descriptor *fd*. Both functions  
 36811 allow time specifications accurate to the nanosecond.

36812 For *futimens()* and *utimensat()*, the *times* argument is an array of two **timespec** structures. The  
 36813 first array member represents the date and time of last access, and the second member  
 36814 represents the date and time of last modification. The times in the **timespec** structure are  
 36815 measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp shall be set  
 36816 to the greatest value supported by the file system that is not greater than the specified time.

36817 If the *tv\_nsec* field of a **timespec** structure has the special value **UTIME\_NOW**, the file's relevant  
 36818 timestamp shall be set to the greatest value supported by the file system that is not greater than  
 36819 the current time. If the *tv\_nsec* field has the special value **UTIME\_OMIT**, the file's relevant  
 36820 timestamp shall not be changed. In either case, the *tv\_sec* field shall be ignored.

36821 If the *times* argument is a null pointer, both the access and modification timestamps shall be set  
 36822 to the greatest value supported by the file system that is not greater than the current time. If  
 36823 *utimensat()* is passed a relative path in the *path* argument, the file to be used shall be relative to  
 36824 the directory associated with the file descriptor *fd* instead of the current working directory. If the  
 36825 access mode of the open file description associated with the file descriptor is not **O\_SEARCH**,  
 36826 the function shall check whether directory searches are permitted using the current permissions  
 36827 of the directory underlying the file descriptor. If the access mode is **O\_SEARCH**, the function  
 36828 shall not perform the check.

36829 If *utimensat()* is passed the special value **AT\_FDCWD** in the *fd* parameter, the current working  
 36830 directory shall be used.

36831 Only a process with the effective user ID equal to the user ID of the file, or with write access to  
 36832 the file, or with appropriate privileges may use *futimens()* or *utimensat()* with a null pointer as  
 36833 the *times* argument or with both *tv\_nsec* fields set to the special value **UTIME\_NOW**. Only a  
 36834 process with the effective user ID equal to the user ID of the file or with appropriate privileges  
 36835 may use *futimens()* or *utimensat()* with a non-null *times* argument that does not have both  
 36836 *tv\_nsec* fields set to **UTIME\_NOW** and does not have both *tv\_nsec* fields set to **UTIME\_OMIT**. If  
 36837 both *tv\_nsec* fields are set to **UTIME\_OMIT**, no ownership or permissions check shall be  
 36838 performed for the file, but other error conditions may still be detected (including **[EACCES]**  
 36839 errors related to the path prefix).

36840 Values for the *flag* argument of *utimensat()* are constructed by a bitwise-inclusive OR of flags

36841 from the following list, defined in `<fcntl.h>`:

36842 `AT_SYMLINK_NOFOLLOW`

36843 If *path* names a symbolic link, then the access and modification times of the symbolic link  
36844 are changed.

36845 Upon successful completion, *futimens()* and *utimensat()* shall mark the last file status change  
36846 timestamp for update, with the exception that if both *tv\_nsec* fields are set to `UTIME_OMIT`, the  
36847 file status change timestamp need not be marked for update.

36848 The *utimes()* function shall be equivalent to the *utimensat()* function with the special value  
36849 `AT_FDCWD` as the *fd* argument and the *flag* argument set to zero, except that the *times* argument  
36850 is a **timeval** structure rather than a **timespec** structure, and accuracy is only to the microsecond,  
36851 not nanosecond, and rounding towards the nearest second may occur.

#### 36852 RETURN VALUE

36853 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
36854 return `-1` and set *errno* to indicate the error. If `-1` is returned, the file times shall not be affected.

#### 36855 ERRORS

36856 The *utimes()* function shall fail, the *futimens()* and *utimensat()* functions shall fail in the case that  
36857 the *times* argument does not have both *tv\_nsec* fields set to `UTIME_OMIT`, and the *futimens()* and  
36858 *utimensat()* functions may fail in the case that the *times* argument has both *tv\_nsec* fields set to  
36859 `UTIME_OMIT`, if:

36860 [EACCES] The *times* argument is a null pointer, or both *tv\_nsec* values are `UTIME_NOW`,  
36861 and the effective user ID of the process does not match the owner of the file  
36862 and write access is denied.

36863 [EINVAL] Either of the *times* argument structures specified a *tv\_nsec* value that was  
36864 neither `UTIME_NOW` nor `UTIME_OMIT`, and was a value less than zero or  
36865 greater than or equal to 1 000 million.

36866 [EINVAL] A new file timestamp would be a value whose *tv\_sec* component is not a value  
36867 supported by the file system.

36868 [EPERM] The *times* argument is not a null pointer, does not have both *tv\_nsec* fields set  
36869 to `UTIME_NOW`, does not have both *tv\_nsec* fields set to `UTIME_OMIT`, the  
36870 calling process' effective user ID does not match the owner of the file, and the  
36871 calling process does not have appropriate privileges.

36872 [EROFS] The file system containing the file is read-only.

36873 The *futimens()* function shall fail in the case that the *times* argument does not have both *tv\_nsec*  
36874 fields set to `UTIME_OMIT`, and may fail in the case that the *times* argument has both *tv\_nsec*  
36875 fields set to `UTIME_OMIT`, if:

36876 [EBADF] The *fd* argument is not a valid file descriptor.

36877 The *utimensat()* function shall fail in the case that the *times* argument does not have both *tv\_nsec*  
36878 fields set to `UTIME_OMIT`, and may fail in the case that the *times* argument has both *tv\_nsec*  
36879 fields set to `UTIME_OMIT`, if:

36880 [EACCES] The access mode of the open file description associated with *fd* is not  
36881 `O_SEARCH` and the permissions of the directory underlying *fd* do not permit  
36882 directory searches.

- 36883 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
36884 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.
- 36885 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated  
36886 with a non-directory file.
- 36887 The *utimes()* function shall fail, the *utimensat()* function shall fail in the case that the *times*  
36888 argument does not have both *tv\_nsec* fields set to *UTIME\_OMIT*, and the *utimensat()* function  
36889 may fail in the case that the *times* argument has both *tv\_nsec* fields set to *UTIME\_OMIT*, if:
- 36890 [EACCES] Search permission is denied by a component of the path prefix.
- 36891 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
36892 argument.
- 36893 [ENAMETOOLONG]  
36894 The length of a component of a pathname is longer than {*NAME\_MAX*}.
- 36895 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 36896 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
36897 directory nor a symbolic link to a directory, or the *path* argument contains at  
36898 least one non-*<slash>* character and ends with one or more trailing *<slash>*  
36899 characters and the last pathname component names an existing file that is  
36900 neither a directory nor a symbolic link to a directory.
- 36901 The *utimensat()* and *utimes()* functions may fail if:
- 36902 [ELOOP] More than {*SYMLOOP\_MAX*} symbolic links were encountered during  
36903 resolution of the *path* argument.
- 36904 [ENAMETOOLONG]  
36905 The length of a pathname exceeds {*PATH\_MAX*}, or pathname resolution of a  
36906 symbolic link produced an intermediate result with a length that exceeds  
36907 {*PATH\_MAX*}.
- 36908 The *utimensat()* function may fail if:
- 36909 [EINVAL] The value of the *flag* argument is not valid.

**36910 EXAMPLES**

36911 None.

**36912 APPLICATION USAGE**

36913 None.

**36914 RATIONALE**

36915 The purpose of the *utimensat()* function is to set the access and modification time of files in  
36916 directories other than the current working directory without exposure to race conditions. Any  
36917 part of the path of a file could be changed in parallel to a call to *utimes()*, resulting in unspecified  
36918 behavior. By opening a file descriptor for the target directory and using the *utimensat()* function  
36919 it can be guaranteed that the changed file is located relative to the desired directory.

36920 The standard developers considered including a special case for the permissions required by  
36921 *utimensat()* when one *tv\_nsec* field is *UTIME\_NOW* and the other is *UTIME\_OMIT*. One  
36922 possibility would be to include this case in with the cases where *times* is a null pointer or both  
36923 fields are *UTIME\_NOW*, where the call is allowed if the process has write permission for the file.  
36924 However, associating write permission with an update to just the last data access timestamp  
36925 (which is normally updated by *read()*) did not seem appropriate. The other possibility would be  
36926 to specify that this one case is allowed if the process has read permission, but this was felt to be

36927 too great a departure from the *utime()* and *utimes()* functions on which *utimensat()* is based. If  
36928 an application needs to set the last data access timestamp to the current time for a file on which  
36929 it has read permission but is not the owner, it can do so by opening the file, reading one or more  
36930 bytes (or reading a directory entry, if the file is a directory), and then closing it.

#### 36931 FUTURE DIRECTIONS

36932 None.

#### 36933 SEE ALSO

36934 *read()*

36935 XBD [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/time.h>](#)

#### 36936 CHANGE HISTORY

36937 First released in Issue 4, Version 2.

#### 36938 Issue 5

36939 Moved from X/OPEN UNIX extension to BASE.

#### 36940 Issue 6

36941 This function is marked LEGACY.

36942 The normative text is updated to avoid use of the term “must” for application requirements.

36943 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
36944 [ELOOP] error condition is added.

#### 36945 Issue 7

36946 Austin Group Interpretation 1003.1-2001 #143 is applied.

36947 The LEGACY marking is removed.

36948 The *utimensat()* function (renamed from *futimesat()*) is added from The Open Group Technical  
36949 Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by  
36950 adding a *flag* argument.

36951 The *futimens()* function is added.

36952 Changes are made related to support for finegrained timestamps.

36953 Changes are made to allow a directory to be opened for searching.

36954 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
36955 pathname exists but is not a directory or a symbolic link to a directory.

36956 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0220 [63,428], XSH/TC1-2008/0221  
36957 [278], XSH/TC1-2008/0222 [324], XSH/TC1-2008/0223 [306], and XSH/TC1-2008/0224 [278] are  
36958 applied.

36959 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0140 [591], XSH/TC2-2008/0141 [817],  
36960 XSH/TC2-2008/0142 [485], and XSH/TC2-2008/0143 [817] are applied.

#### 36961 Issue 8

36962 Austin Group Defect 1280 is applied, changing the ERRORS section.

36963 **NAME**

36964 fwide — set stream orientation

36965 **SYNOPSIS**

36966 #include &lt;stdio.h&gt;

36967 #include &lt;wchar.h&gt;

36968 int fwide(FILE \**stream*, int *mode*);36969 **DESCRIPTION**

36970 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
36971 conflict between the requirements described here and the ISO C standard is unintentional. This  
36972 volume of POSIX.1-2024 defers to the ISO C standard.

36973 The *fwide()* function shall determine the orientation of the stream pointed to by *stream*. If *mode* is  
36974 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less  
36975 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero  
36976 and the function does not alter the orientation of the stream.

36977 If the orientation of the stream has already been determined, *fwide()* shall not change it.

36978 CX The *fwide()* function shall not change the setting of *errno* if successful.

36979 Since no return value is reserved to indicate an error, an application wishing to check for error  
36980 situations should set *errno* to 0, then call *fwide()*, then check *errno*, and if it is non-zero, assume  
36981 an error has occurred.

36982 **RETURN VALUE**

36983 The *fwide()* function shall return a value greater than zero if, after the call, the stream has wide-  
36984 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no  
36985 orientation.

36986 **ERRORS**

36987 The *fwide()* function may fail if:

36988 CX [EBADF] The *stream* argument is not a valid stream.

36989 **EXAMPLES**

36990 None.

36991 **APPLICATION USAGE**

36992 A call to *fwide()* with *mode* set to zero can be used to determine the current orientation of a  
36993 stream.

36994 **RATIONALE**

36995 None.

36996 **FUTURE DIRECTIONS**

36997 None.

36998 **SEE ALSO**

36999 XBD <stdio.h>, <wchar.h>

37000 **CHANGE HISTORY**

37001 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
37002 (E).

37003 **Issue 6**

37004 Extensions beyond the ISO C standard are marked.

37005 **Issue 7**

37006 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0225 [272] is applied.



37007 **NAME**37008 `fwprintf`, `swprintf`, `wprintf` — print formatted wide-character output37009 **SYNOPSIS**

```

37010 #include <stdio.h>
37011 #include <wchar.h>
37012 int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
37013 int swprintf(wchar_t *restrict ws, size_t n,
37014             const wchar_t *restrict format, ...);
37015 int wprintf(const wchar_t *restrict format, ...);

```

37016 **DESCRIPTION**

37017 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 37018 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37019 volume of POSIX.1-2024 defers to the ISO C standard.

37020 The `fwprintf()` function shall place output on the named output *stream*. The `wprintf()` function  
 37021 shall place output on the standard output stream *stdout*. The `swprintf()` function shall place  
 37022 output followed by the null wide character in consecutive wide characters starting at *\*ws*; no  
 37023 more than *n* wide characters shall be written, including a terminating null wide character, which  
 37024 is always added (unless *n* is zero).

37025 Each of these functions shall convert, format, and print its arguments under control of the *format*  
 37026 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-characters*,  
 37027 which are simply copied to the output stream, and *conversion specifications*, each of which results  
 37028 in the fetching of zero or more arguments. The results are undefined if there are insufficient  
 37029 arguments for the *format*. If the *format* is exhausted while arguments remain, the excess  
 37030 arguments are evaluated but are otherwise ignored.

37031 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 37032 to the next unused argument. In this case, the conversion specifier wide character `%` (see below)  
 37033 is replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range  
 37034 `[1,{NL_ARGMAX}]`, giving the position of the argument in the argument list. This feature  
 37035 provides for the definition of *format* wide-character strings that select arguments in an order  
 37036 appropriate to specific languages (see the EXAMPLES section).

37037 The *format* can contain either numbered argument conversion specifications (that is, those  
 37038 introduced by `"%n$"` and optionally containing the `"*m$"` forms of field width and precision),  
 37039 or unnumbered argument conversion specifications (that is, those introduced by the `%` character  
 37040 and optionally containing the `*` form of field width and precision), but not both. The only  
 37041 exception to this is that `%%` can be mixed with the `"%n$"` form. The results of mixing numbered  
 37042 and unnumbered argument specifications in a *format* wide-character string are undefined. When  
 37043 numbered argument specifications are used, specifying the *N*th argument requires that all the  
 37044 leading arguments, from the first to the (*N*−1)th, are specified in the *format* wide-character string.

37045 In *format* wide-character strings containing the `"%n$"` form of conversion specification,  
 37046 numbered arguments in the argument list can be referenced from the *format* wide-character  
 37047 string as many times as required.

37048 In *format* wide-character strings containing the `%` form of conversion specification, each  
 37049 argument in the argument list shall be used exactly once. It is unspecified whether an encoding  
 37050 error occurs if the format string contains `wchar_t` values that do not correspond to members of  
 37051 the character set of the current locale and the specified semantics do not require that value to be  
 37052 processed by `wcrtomb()`.

37053 CX All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character  
 37054 in the output string, output as a wide-character value. The radix character is defined in the  
 37055 current locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the radix  
 37056 character is not defined, the radix character shall default to a <period> ('.').

37057 CX Each conversion specification is introduced by the '%' wide character or by the wide-character  
 37058 sequence "%n\$", after which the following appear in sequence:

- 37059 • Zero or more *flags* (in any order), which modify the meaning of the conversion  
 37060 specification.

- 37061 • An optional minimum *field width*. If the converted value has fewer wide characters than  
 37062 the field width, it shall be padded with <space> characters by default on the left; it shall be  
 37063 padded on the right, if the left-adjustment flag ('-'), described below, is given to the field  
 37064 CX width. The field width takes the form of an <asterisk> ('\*'), or in conversion  
 37065 specifications introduced by "%n\$" the "\*m\$" string, described below, or a decimal  
 37066 integer.

- 37067 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,  
 37068 x, and X conversion specifiers; the number of digits to appear after the radix character for  
 37069 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for  
 37070 the g and G conversion specifiers; or the maximum number of wide characters to be  
 37071 printed from a string in the s conversion specifiers. The precision takes the form of a  
 37072 CX <period> ('.') followed either by an <asterisk> ('\*'), or in conversion specifications  
 37073 introduced by "%n\$" the "\*m\$" string, described below, or an optional decimal digit  
 37074 string, where a null digit string is treated as 0. If a precision appears with any other  
 37075 conversion wide character, the behavior is undefined.

- 37076 • An optional length modifier that specifies the size of the argument.

- 37077 • A *conversion specifier* wide character that indicates the type of conversion to be applied.

37078 A field width, or precision, or both, may be indicated by an <asterisk> ('\*'). In this case an  
 37079 argument of type **int** supplies the field width or precision. Applications shall ensure that  
 37080 arguments specifying field width, or precision, or both appear in that order before the argument,  
 37081 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field  
 37082 CX width. A negative precision is taken as if the precision were omitted. In *format* wide-character  
 37083 strings containing conversion specifications introduced by "%n\$", in addition to being indicated  
 37084 by the decimal digit string, a field width may be indicated by the sequence "\*m\$" and precision  
 37085 by the sequence ".\*m\$", where *m* is a decimal integer in the range [1,{NL\_ARGMAX}] giving  
 37086 the position in the argument list (after the *format* argument) of an integer argument containing  
 37087 the field width or precision, for example:

```
37088 wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

37089 The flag wide characters and their meanings are:

- 37090 CX ' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d,  
 37091 %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping wide characters. For  
 37092 other conversions, the behavior is undefined. The numeric grouping wide character is  
 37093 used.

- 37094 - The result of the conversion shall be left-justified within the field. The conversion shall  
 37095 be right-justified if this flag is not specified.

- 37096 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The  
 37097 conversion shall begin with a sign only when a negative value is converted if this flag is  
 37098 not specified.

- 37099 <space> If the first wide character of a signed conversion is not a sign, or if a signed conversion  
 37100 results in no wide characters, a <space> shall be prefixed to the result. This means that  
 37101 if the <space> and '+' flags both appear, the <space> flag shall be ignored.
- 37102 # Specifies that the value is to be converted to an alternative form. For o conversion, it  
 37103 shall increase the precision, if and only if necessary, to force the first digit of the result  
 37104 to be zero (if the value and precision are both 0, a single 0 is printed). For x or X  
 37105 conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,  
 37106 E, f, F, g, and G conversion specifiers, the result shall always contain a radix character,  
 37107 even if no digits follow it. Without this flag, a radix character appears in the result of  
 37108 these conversions only if a digit follows it. For g and G conversion specifiers, trailing  
 37109 zeros shall *not* be removed from the result as they normally are. For other conversion  
 37110 specifiers, the behavior is undefined.
- 37111 0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros  
 37112 (following any indication of sign or base) are used to pad to the field width rather than  
 37113 performing space padding, except when converting an infinity or NaN. If the '0' and  
 37114 '-' flags both appear, the '0' flag shall be ignored. For d, i, o, u, x, and X conversion  
 37115 CX specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and  
 37116 <apostrophe> flags both appear, the grouping wide characters are inserted before zero  
 37117 padding. For other conversions, the behavior is undefined.
- 37118 The length modifiers and their meanings are:
- 37119 hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char**  
 37120 or **unsigned char** argument (the argument will have been promoted according to the  
 37121 integer promotions, but its value shall be converted to **signed char** or **unsigned char**  
 37122 before printing); or that a following n conversion specifier applies to a pointer to a  
 37123 **signed char** argument.
- 37124 h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or  
 37125 **unsigned short** argument (the argument will have been promoted according to the  
 37126 integer promotions, but its value shall be converted to **short** or **unsigned short** before  
 37127 printing); or that a following n conversion specifier applies to a pointer to a **short**  
 37128 argument.
- 37129 l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or  
 37130 **unsigned long** argument; that a following n conversion specifier applies to a pointer to  
 37131 a **long** argument; that a following c conversion specifier applies to a **wint\_t** argument;  
 37132 that a following s conversion specifier applies to a pointer to a **wchar\_t** argument; or  
 37133 has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.
- 37134 ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long**  
 37135 or **unsigned long long** argument; or that a following n conversion specifier applies to a  
 37136 pointer to a **long long** argument.
- 37138 j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax\_t**  
 37139 or **uintmax\_t** argument; or that a following n conversion specifier applies to a pointer  
 37140 to an **intmax\_t** argument.
- 37141 z Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **size\_t** or the  
 37142 corresponding signed integer type argument; or that a following n conversion specifier  
 37143 applies to a pointer to a signed integer type corresponding to a **size\_t** argument.

37144	t	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type argument; or that a following n conversion specifier applies to a pointer to a <b>ptrdiff_t</b> argument.
37145		
37146		
37147	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a <b>long double</b> argument.
37148		
37149		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
37150		
37151		The conversion specifiers and their meanings are:
37152	d, i	The <b>int</b> argument shall be converted to a signed decimal in the style "[−] dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
37153		
37154		
37155		
37156		
37157	o	The <b>unsigned</b> argument shall be converted to unsigned octal format in the style "dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
37158		
37159		
37160		
37161		
37162	u	The <b>unsigned</b> argument shall be converted to unsigned decimal format in the style "dddd". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
37163		
37164		
37165		
37166		
37167	x	The <b>unsigned</b> argument shall be converted to unsigned hexadecimal format in the style "dddd"; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.
37168		
37169		
37170		
37171		
37172	X	Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".
37173		
37174	f, F	The <b>double</b> argument shall be converted to decimal notation in the style "[−] ddd.ddd", where the number of digits after the radix character shall be equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit shall appear before it. The value shall be rounded in an implementation-defined manner to the appropriate number of digits.
37175		
37176		
37177		
37178		
37179		
37180		A <b>double</b> argument representing an infinity shall be converted in one of the styles "[−]inf" or "[−]infinity"; which style is implementation-defined. A <b>double</b> argument representing a NaN shall be converted in one of the styles "[−]nan" or "[−]nan (n-char-sequence)"; which style, and the meaning of any n-char-sequence, is implementation-defined. The F conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.
37181		
37182		
37183		
37184		
37185		
37186	e, E	The <b>double</b> argument shall be converted in the style "[−] d.ddde±dd", where there shall be one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it shall be equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no
37187		
37188		
37189		

37190		radix character shall appear. The value shall be rounded in an implementation-defined manner to the appropriate number of digits. The E conversion wide character shall produce a number with 'E' instead of 'e' introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.
37191		
37192		
37193		
37194		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
37195		
37196	g, G	The <b>double</b> argument representing a floating-point number shall be converted in the style <code>f</code> or <code>e</code> (or in the style <code>F</code> or <code>E</code> in the case of a <code>G</code> conversion specifier), depending on the value converted and the precision. Let <code>P</code> equal the precision if non-zero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style <code>E</code> would have an exponent of <code>X</code> :
37197		
37198		
37199		
37200		
37201		— If $P > X \geq -4$ , the conversion shall be with style <code>f</code> (or <code>F</code> ) and precision $P - (X + 1)$ .
37202		— Otherwise, the conversion shall be with style <code>e</code> (or <code>E</code> ) and precision $P - 1$ .
37203		Finally, unless the '#' flag is used, any trailing zeros shall be removed from the fractional portion of the result and the decimal-point character shall be removed if there is no fractional portion remaining.
37204		
37205		
37206		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
37207		
37208	a, A	A <b>double</b> argument representing a floating-point number shall be converted in the style " <code>[-]0xh.hhhhp±d</code> ", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and <code>FLT_RADIX</code> is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and <code>FLT_RADIX</code> is not a power of 2, then the precision shall be sufficient to distinguish values of type <b>double</b> , except that trailing zeros may be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point wide character shall appear. The letters "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.
37209		
37210		
37211		
37212		
37213		
37214		
37215		
37216		
37217		
37218		
37219		
37220		
37221		
37222		A <b>double</b> argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
37223		
37224	c	If no <code>l</code> (ell) qualifier is present, the <b>int</b> argument shall be converted to a wide character as if by calling the <code>btowc()</code> function and the resulting wide character shall be written. Otherwise, the <b>wint_t</b> argument shall be converted to <b>wchar_t</b> , and written.
37225		
37226		
37227	s	If no <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the <code>mbrtowc()</code> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.
37228		
37229		
37230		
37231		
37232		
37233		
37234		
37235		

37236			If an <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type <code>wchar_t</code> . Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.
37237			
37238			
37239			
37240			
37241			
37242		<code>p</code>	The application shall ensure that the argument is a pointer to <code>void</code> . The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.
37243			
37244			
37245		<code>n</code>	The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the <code>fwprintf()</code> functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
37246			
37247			
37248			
37249			
37250	XSI	<code>C</code>	Equivalent to <code>lc</code> .
37251	XSI	<code>S</code>	Equivalent to <code>ls</code> .
37252		<code>%</code>	Write a <code>'%'</code> wide character; no argument shall be converted. The application shall ensure that the complete conversion specification is <code>%%</code> .
37253			
37254			If a conversion specification does not match one of the above forms, the behavior is undefined.
37255			In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <code>fwprintf()</code> and <code>wprintf()</code> shall be printed as if <code>fputwc()</code> had been called.
37256			
37257			
37258			
37259			For <code>a</code> and <code>A</code> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.
37260			
37261			
37262			
37263			For <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, if the number of significant decimal digits is at most <code>DECIMAL_DIG</code> , then the result should be correctly rounded. If the number of significant decimal digits is more than <code>DECIMAL_DIG</code> but the source value is exactly representable with <code>DECIMAL_DIG</code> digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$ , both having <code>DECIMAL_DIG</code> significant digits; the value of the resultant decimal string $D$ should satisfy $L \leq D \leq U$ , with the extra stipulation that the error should have a correct sign for the current rounding direction.
37264			
37265			
37266			
37267			
37268			
37269			
37270			
37271	CX		The last data modification and last file status change timestamps of the file shall be marked for update between the call to a successful execution of <code>fwprintf()</code> or <code>wprintf()</code> and the next successful completion of a call to <code>fflush()</code> or <code>fclose()</code> on the same stream, or a call to <code>exit()</code> or <code>abort()</code> .
37272			
37273			
37274			
37275			<b>RETURN VALUE</b>
37276			Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide character in the case of <code>swprintf()</code> , or a negative value if an error was encountered, and set <code>errno</code> to indicate the error.
37277			
37278	CX		
37279			If <code>n</code> or more wide characters were requested to be written, <code>swprintf()</code> shall return a negative value, and set <code>errno</code> to indicate the error.
37280	CX		

**37281 ERRORS**

37282 For the conditions under which *fwprintf()* and *wprintf()* fail and may fail, refer to *fputwc()*.

37283 In addition, all forms of *fwprintf()* shall fail if:

37284 CX [EILSEQ] A wide-character code that does not correspond to a valid character has been  
37285 detected.

37286 CX [EOVERFLOW] The value to be returned is greater than {INT\_MAX}.

37287 In addition, *fwprintf()* and *wprintf()* may fail if:

37288 CX [ENOMEM] Insufficient storage space is available.

37289 The *swprintf()* shall fail if:

37290 CX [EOVERFLOW] The number of wide characters requested to be written was *n* or more.

**37291 EXAMPLES**

37292 To print the language-independent date and time format, the following statement could be used:

37293 `wprintf(format, weekday, month, day, hour, min);`

37294 For American usage, *format* could be a pointer to the wide-character string:

37295 `L"%s, %s %d, %d:%.2d\n"`

37296 producing the message:

37297 `Sunday, July 3, 10:02`

37298 whereas for German usage, *format* could be a pointer to the wide-character string:

37299 `L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"`

37300 producing the message:

37301 `Sonntag, 3. Juli, 10:02`

**37302 APPLICATION USAGE**

37303 None.

**37304 RATIONALE**

37305 If an implementation detects that there are insufficient arguments for the format, it is  
37306 recommended that the function should fail and report an [EINVAL] error.

**37307 FUTURE DIRECTIONS**

37308 None.

**37309 SEE ALSO**

37310 [Section 2.5](#) (on page 521), [btowc\(\)](#), [fputwc\(\)](#), [fwscanf\(\)](#), [mbrtowc\(\)](#), [setlocale\(\)](#)

37311 [XBD Chapter 7](#) (on page 127), [<inttypes.h>](#), [<stdio.h>](#), [<wchar.h>](#)

**37312 CHANGE HISTORY**

37313 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
37314 (E).

**37315 Issue 6**

37316 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing  
37317 the case if *n* or more wide characters are requested to be written using *swprintf()*.

37318 The normative text is updated to avoid use of the term “must” for application requirements.

37319 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 37320           • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 37321           • The DESCRIPTION is updated.
- 37322           • The hh, ll, j, t, and z length modifiers are added.
- 37323           • The a, A, and F conversion characters are added.
- 37324           • XSI shading is removed from the description of character string representations of infinity and NaN floating-point values.
- 37325
- 37326           The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.
- 37327
- 37328           ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.
- 37329   **Issue 7**
- 37330           Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0 flag.
- 37331
- 37332           Austin Group Interpretation 1003.1-2001 #170 is applied.
- 37333           ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied, revising the description of g and G.
- 37334
- 37335           Functionality relating to the "%n\$" form of conversion specification and the <apostrophe> flag is moved from the XSI option to the Base.
- 37336
- 37337           The [Eoverflow] error is added for *swprintf()*.
- 37338           Changes are made related to support for finegrained timestamps.
- 37339           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0226 [302] and XSH/TC1-2008/0227 [14] are applied.
- 37340
- 37341           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0144 [73], XSH/TC2-2008/0145 [894], XSH/TC2-2008/0146 [557], and XSH/TC2-2008/0147 [936] are applied.
- 37342
- 37343   **Issue 8**
- 37344           Austin Group Defect 1021 is applied, changing “output error” to “error” in the RETURN VALUE section and changing the requirements for [Eoverflow].
- 37345
- 37346           Austin Group Defect 1137 is applied, clarifying the use of "%n\$" and "%m\$" in conversion specifications.
- 37347
- 37348           Austin Group Defect 1205 is applied, changing the description of the % conversion specifier.
- 37349           Austin Group Defect 1219 is applied, changing the *swprintf()*-specific [Eoverflow] error.



37350 **NAME**

37351 fwrite — binary output

37352 **SYNOPSIS**

37353 #include &lt;stdio.h&gt;

```
37354     size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
37355                 FILE *restrict stream);
```

37356 **DESCRIPTION**

37357 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 37358 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37359 volume of POSIX.1-2024 defers to the ISO C standard.

37360 The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose  
 37361 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be  
 37362 made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly  
 37363 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by  
 37364 the number of bytes successfully written. If an error occurs, the resulting value of the file-  
 37365 position indicator for the stream is unspecified.

37366 CX The last data modification and last file status change timestamps of the file shall be marked for  
 37367 update between the successful execution of *fwrite()* and the next successful completion of a call  
 37368 to *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.

37369 **RETURN VALUE**

37370 The *fwrite()* function shall return the number of elements successfully written, which shall be  
 37371 less than *nitems* only if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0  
 37372 and the state of the stream remains unchanged. Otherwise, if a write error occurs, the error  
 37373 CX indicator for the stream shall be set, and *errno* shall be set to indicate the error.

37374 **ERRORS**37375 Refer to *fputc()*.37376 **EXAMPLES**

37377 None.

37378 **APPLICATION USAGE**

37379 Because of possible differences in element length and byte ordering, files written using *fwrite()*  
 37380 are application-dependent, and possibly cannot be read using *fread()* by a different application  
 37381 or by the same application on a different processor.

37382 **RATIONALE**

37383 None.

37384 **FUTURE DIRECTIONS**

37385 None.

37386 **SEE ALSO**37387 [Section 2.5](#) (on page 521), *ferror()*, *fopen()*, *fprintf()*, *putc()*, *puts()*, *write()*37388 XBD [<stdio.h>](#)37389 **CHANGE HISTORY**

37390 First released in Issue 1. Derived from Issue 1 of the SVID.

37391 **Issue 6**

37392 Extensions beyond the ISO C standard are marked.

37393 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

37394 • The *fwrite()* prototype is updated.37395 • The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.37396 **Issue 7**

37397 Changes are made related to support for finegrained timestamps.

37398 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0228 [14] is applied.

37399 **Issue 8**

37400 Austin Group Defect 1196 is applied, clarifying the RETURN VALUE section.

37401 **NAME**

37402 fwscanf, swscanf, wscanf — convert formatted wide-character input

37403 **SYNOPSIS**

37404 #include &lt;stdio.h&gt;

37405 #include &lt;wchar.h&gt;

37406 int fwscanf(FILE \*restrict stream, const wchar\_t \*restrict format, ...);

37407 int swscanf(const wchar\_t \*restrict ws,

37408 const wchar\_t \*restrict format, ...);

37409 int wscanf(const wchar\_t \*restrict format, ...);

37410 **DESCRIPTION**

37411 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 37412 conflict between the requirements described here and the ISO C standard is unintentional. This  
 37413 volume of POSIX.1-2024 defers to the ISO C standard.

37414 The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read  
 37415 from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character  
 37416 string *ws*. Each function reads wide characters, interprets them according to a format, and stores  
 37417 the results in its arguments. Each expects, as arguments, a control wide-character string *format*  
 37418 described below, and a set of *pointer* arguments indicating where the converted input should be  
 37419 stored. The result is undefined if there are insufficient arguments for the format. If the *format* is  
 37420 exhausted while arguments remain, the excess arguments are evaluated but are otherwise  
 37421 ignored.

37422 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 37423 to the next unused argument. In this case, the conversion specifier wide character % (see below)  
 37424 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range  
 37425 [1,{NL\_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that  
 37426 select arguments in an order appropriate to specific languages. In *format* wide-character strings  
 37427 containing the "%n\$" form of conversion specifications, it is unspecified whether numbered  
 37428 arguments in the argument list can be referenced from the *format* wide-character string more  
 37429 than once.

37430 The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the  
 37431 two forms cannot normally be mixed within a single *format* wide-character string. The only  
 37432 exception to this is that %% or %\* can be mixed with the "%n\$" form. When numbered argument  
 37433 specifications are used, specifying the *N*th argument requires that all the leading arguments,  
 37434 from the first to the (*N*−1)th, are pointers.

37435 The *fwscanf()* function in all its forms allows for detection of a language-dependent radix  
 37436 character in the input string, encoded as a wide-character value. The radix character is defined  
 37437 in the current locale (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the  
 37438 radix character is not defined, the radix character shall default to a <period> ('.').

37439 The *format* is a wide-character string composed of zero or more directives. Each directive is  
 37440 composed of one of the following: one or more white-space wide characters; an ordinary wide  
 37441 character (neither '%' nor a white-space wide character); or a conversion specification. It is  
 37442 unspecified whether an encoding error occurs if the format string contains **wchar\_t** values that  
 37443 do not correspond to members of the character set of the current locale and the specified  
 37444 semantics do not require that value to be processed by *wcrtomb()*.

37445 CX Each conversion specification is introduced by the '%' or by the character sequence "%n\$",  
 37446 after which the following appear in sequence:

- 37447 • An optional assignment-suppressing character ' \* '.
- 37448 • An optional non-zero decimal integer that specifies the maximum field width.
- 37449 CX • An optional assignment-allocation character ' m '.
- 37450 • An optional length modifier that specifies the size of the receiving object.
- 37451 • A conversion specifier wide character that specifies the type of conversion to be applied.
- 37452 The valid conversion specifiers are described below.

37453 The *fwscanf()* functions shall execute each directive of the format in turn. When all directives  
 37454 have been executed, or if a directive fails (as detailed below), the function shall return. Failures  
 37455 are described as input failures (due to the unavailability of input bytes) or matching failures  
 37456 (due to inappropriate input).

37457 A directive composed of one or more white-space wide characters shall be executed by reading  
 37458 input up to the first wide character that is not a white-space wide character, which shall remain  
 37459 unread, or until no more wide characters can be read. The directive shall never fail.

37460 A directive that is an ordinary wide character shall be executed as follows. The next wide  
 37461 character is read from the input and compared with the wide character that comprises the  
 37462 directive; if the comparison shows that they are not equivalent, the directive shall fail, and the  
 37463 differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding  
 37464 error, or a read error prevents a wide character from being read, the directive shall fail.

37465 A directive that is a conversion specification defines a set of matching input sequences, as  
 37466 described below for each conversion wide character. A conversion specification is executed in  
 37467 the following steps.

37468 Input white-space wide characters shall be skipped, unless the conversion specification includes  
 37469 a [ , c, or n conversion specifier.

37470 An item shall be read from the input, unless the conversion specification includes an n  
 37471 conversion specifier wide character. An input item is defined as the longest sequence of input  
 37472 wide characters, not exceeding any specified field width, which is an initial subsequence of a  
 37473 matching sequence. The first wide character, if any, after the input item shall remain unread. If  
 37474 the length of the input item is zero, the execution of the conversion specification shall fail; this  
 37475 condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented  
 37476 input from the stream, in which case it is an input failure.

37477 Except in the case of a % conversion specifier, the input item (or, in the case of a %n conversion  
 37478 specification, the count of input wide characters) shall be converted to a type appropriate to the  
 37479 conversion wide character. If the input item is not a matching sequence, the execution of the  
 37480 conversion specification shall fail; this condition is a matching failure. Unless assignment  
 37481 suppression was indicated by a ' \* ' , the result of the conversion shall be placed in the object  
 37482 pointed to by the first argument following the *format* argument that has not already received a  
 37483 CX conversion result if the conversion specification is introduced by %, or in the *n*th argument if  
 37484 introduced by the wide-character sequence "%n\$". If this object does not have an appropriate  
 37485 type, or if the result of the conversion cannot be represented in the space provided, the behavior  
 37486 is undefined.

37487 CX The c, s, and [ conversion specifiers shall accept an optional assignment-allocation character  
 37488 ' m ' , which shall cause a memory buffer to be allocated to hold the conversion results. If the  
 37489 conversion specifier is s or [ , the allocated buffer shall include space for a terminating null  
 37490 character (or wide character). In such a case, the argument corresponding to the conversion  
 37491 specifier should be a reference to a pointer value that will receive a pointer to the allocated  
 37492 buffer. The system shall allocate a buffer as if *malloc()* had been called. The application shall be

37493 responsible for freeing the memory after usage. If there is insufficient memory to allocate a  
 37494 buffer, the function shall set *errno* to [ENOMEM] and a conversion error shall result. If the  
 37495 function returns EOF, any memory successfully allocated for parameters using assignment-  
 37496 allocation character 'm' by this call shall be freed before the function returns.

37497 The length modifiers and their meanings are:

37498 hh Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
 37499 argument with type pointer to **signed char** or **unsigned char**.

37500 h Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
 37501 argument with type pointer to **short** or **unsigned short**.

37502 l (ell) Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
 37503 argument with type pointer to **long** or **unsigned long**; that a following a, A, e, E, f, F,  
 37504 g, or G conversion specifier applies to an argument with type pointer to **double**; or that  
 37505 a following c, s, or [ conversion specifier applies to an argument with type pointer to  
 37506 CX **wchar\_t**. If the 'm' assignment-allocation character is specified, the conversion  
 37507 applies to an argument with the type pointer to a pointer to **wchar\_t**.

37508 ll (ell-ell) Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
 37509 argument with type pointer to **long long** or **unsigned long long**.

37511 j Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
 37512 argument with type pointer to **intmax\_t** or **uintmax\_t**.

37513 z Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
 37514 argument with type pointer to **size\_t** or the corresponding signed integer type.

37515 t Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an  
 37516 argument with type pointer to **ptrdiff\_t** or the corresponding **unsigned** type.

37517 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an  
 37518 argument with type pointer to **long double**.

37519 If a length modifier appears with any conversion specifier other than as specified above, the  
 37520 behavior is undefined.

37521 The following conversion specifier wide characters are valid:

37522 d Matches an optionally signed decimal integer, whose format is the same as expected for  
 37523 the subject sequence of *wcstol()* with the value 10 for the *base* argument. In the absence  
 37524 of a size modifier, the application shall ensure that the corresponding argument is a  
 37525 pointer to **int**.

37526 i Matches an optionally signed integer, whose format is the same as expected for the  
 37527 subject sequence of *wcstol()* with 0 for the *base* argument. In the absence of a size  
 37528 modifier, the application shall ensure that the corresponding argument is a pointer to  
 37529 **int**.

37530 o Matches an optionally signed octal integer, whose format is the same as expected for  
 37531 the subject sequence of *wcstoul()* with the value 8 for the *base* argument. In the absence  
 37532 of a size modifier, the application shall ensure that the corresponding argument is a  
 37533 pointer to **unsigned**.

37534 u Matches an optionally signed decimal integer, whose format is the same as expected for  
 37535 the subject sequence of *wcstoul()* with the value 10 for the *base* argument. In the absence  
 37536 of a size modifier, the application shall ensure that the corresponding argument is a

37537		pointer to <b>unsigned</b> .
37538	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument.
37539		In the absence of a size modifier, the application shall ensure that the corresponding
37540		argument is a pointer to <b>unsigned</b> .
37541		
37542	a, e, f, g	
37543		Matches an optionally signed floating-point number, infinity, or NaN whose format is
37544		the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size
37545		modifier, the application shall ensure that the corresponding argument is a pointer to
37546		<b>float</b> .
37547		If the <i>fwprintf()</i> family of functions generates character string representations for
37548		infinity and NaN (a symbolic entity encoded in floating-point format) to support
37549		IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.
37550	s	Matches a sequence of non-white-space wide characters. If no l (ell) qualifier is present,
37551		characters from the input field shall be converted as if by repeated calls to the
37552		<i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object
37553		initialized to zero before the first wide character is converted. If the 'm' assignment-
37554		allocation character is not specified, the application shall ensure that the corresponding
37555		argument is a pointer to a character array large enough to accept the sequence and the
37556	CX	terminating null character, which shall be added automatically. Otherwise, the
37557		application shall ensure that the corresponding argument is a pointer to a pointer to a
37558		<b>char</b> .
37559		If the l (ell) qualifier is present and the 'm' assignment-allocation character is not
37560		specified, the application shall ensure that the corresponding argument is a pointer to
37561		an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide
37562	CX	character, which shall be added automatically. If the l (ell) qualifier is present and the
37563		'm' assignment-allocation character is present, the application shall ensure that the
37564		corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
37565	[	Matches a non-empty sequence of wide characters from a set of expected wide
37566		characters (the <i>scanset</i> ). If no l (ell) qualifier is present, wide characters from the input
37567		field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the
37568		conversion state described by an <b>mbstate_t</b> object initialized to zero before the first
37569		wide character is converted. If the 'm' assignment-allocation character is not specified,
37570		the application shall ensure that the corresponding argument is a pointer to a character
37571		array large enough to accept the sequence and the terminating null character, which
37572	CX	shall be added automatically. Otherwise, the application shall ensure that the
37573		corresponding argument is a pointer to a pointer to a <b>char</b> .
37574		If an l (ell) qualifier is present and the 'm' assignment-allocation character is not
37575		specified, the application shall ensure that the corresponding argument is a pointer to
37576		an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide
37577	CX	character. If an l (ell) qualifier is present and the 'm' assignment-allocation character
37578		is specified, the application shall ensure that the corresponding argument is a pointer to
37579		a pointer to a <b>wchar_t</b> .
37580		The conversion specification includes all subsequent wide characters in the <i>format</i>
37581		string up to and including the matching <right-square-bracket> (']'). The wide
37582		characters between the square brackets (the <i>scanlist</i> ) comprise the <i>scanset</i> , unless the
37583		wide character after the <left-square-bracket> is a <circumflex> ('^'), in which case
37584		the <i>scanset</i> contains all wide characters that do not appear in the <i>scanlist</i> between the

37585		<circumflex> and the <right-square-bracket>. If the conversion specification begins with "[ ]" or "[^]", the <right-square-bracket> is included in the scanlist and the next <right-square-bracket> is the matching <right-square-bracket> that ends the conversion specification; otherwise, the first <right-square-bracket> is the one that ends the conversion specification. If a '-' is in the scanlist and is not the first wide character, nor the second where the first wide character is a '^', nor the last wide character, the behavior is implementation-defined.
37586		
37587		
37588		
37589		
37590		
37591		
37592	c	Matches a sequence of wide characters of exactly the number specified by the field width (1 if no field width is present in the conversion specification).
37593		
37594		If no l (ell) length modifier is present, characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first wide character is converted. No null character is added. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial element of a character array large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .
37595		
37596		
37597		
37598		
37599	CX	
37600		
37601		
37602		No null wide character is added. If an l (ell) length modifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument shall be a pointer to the initial element of an array of <b>wchar_t</b> large enough to accept the sequence. If an l (ell) qualifier is present and the 'm' assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .
37603		
37604		
37605	CX	
37606		
37607		
37608	p	Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding <i>fwprintf()</i> functions. The application shall ensure that the corresponding argument is a pointer to a pointer to <b>void</b> . The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion is undefined.
37609		
37610		
37611		
37612		
37613		
37614		
37615	n	No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which is to be written the number of wide characters read from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing wide character or a field width, the behavior is undefined.
37616		
37617		
37618		
37619		
37620		
37621		
37622	XSI	<b>C</b> Equivalent to <b>lc</b> .
37623	XSI	<b>S</b> Equivalent to <b>ls</b> .
37624	%	Matches a single '%' wide character; no conversion or assignment shall occur. The complete conversion specification shall be %%.
37625		
37626		If a conversion specification is invalid, the behavior is undefined.
37627		The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively, a, e, f, g, and x.
37628		
37629		If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current conversion specification (except for %n) have been
37630		

37631 read (other than leading white-space wide characters, where permitted), execution of the current  
 37632 conversion specification shall terminate with an input failure. Otherwise, unless execution of  
 37633 the current conversion specification is terminated with a matching failure, execution of the  
 37634 following conversion specification (if any) shall be terminated with an input failure.

37635 Reaching the end of the string in *swscanf()* shall be equivalent to encountering end-of-file for  
 37636 *fwscanf()*.

37637 If conversion terminates on a conflicting input, the offending input shall be left unread in the  
 37638 input. Any trailing white-space wide characters (including <newline>) shall be left unread  
 37639 unless matched by a conversion specification. The success of literal matches and suppressed  
 37640 assignments is only directly determinable via the %n conversion specification.

37641 CX The *fwscanf()* and *wscanf()* functions may mark the last data access timestamp of the file  
 37642 associated with *stream* for update. The last data access timestamp shall be marked for update by  
 37643 the first successful execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *vfwscanf()*,  
 37644 *vscanf()*, or *wscanf()* using *stream* that returns data not supplied by a prior call to *ungetwc()*.

### 37645 RETURN VALUE

37646 Upon successful completion, these functions shall return the number of successfully matched  
 37647 and assigned input items; this number can be zero in the event of an early matching failure. If  
 37648 the input ends before the first conversion (if any) has completed, and without a matching failure  
 37649 having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has  
 37650 CX completed, and without a matching failure having occurred, EOF shall be returned and *errno*  
 37651 shall be set to indicate the error. If a read error occurs, the error indicator for the stream shall be  
 37652 set.

### 37653 ERRORS

37654 For the conditions under which the *fwscanf()* functions shall fail and may fail, refer to *fgetwc()*.

37655 In addition, the *fwscanf()* function shall fail if:

37656 CX [EILSEQ] Input byte sequence does not form a valid character.

37657 [ENOMEM] Insufficient storage space is available.

37658 In addition, the *fwscanf()* function may fail if:

37659 CX [EINVAL] There are insufficient arguments.

### 37660 EXAMPLES

37661 The call:

```
37662 int i, n; float x; char name[50];
37663 n = wscanf(L"%d%f%s", &i, &x, name);
```

37664 with the input line:

```
37665 25 54.32E-1 Hamster
```

37666 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string  
 37667 "Hamster".

37668 The call:

```
37669 int i; float x; char name[50];
37670 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

37671 with input:

```
37672 56789 0123 56a72
```



37673 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to  
 37674 *getchar()* shall return the character 'a'.

#### 37675 APPLICATION USAGE

37676 In format strings containing the '%' form of conversion specifications, each argument in the  
 37677 argument list is used exactly once.

37678 For functions that allocate memory as if by *malloc()*, the application should release such memory  
 37679 when it is no longer required by a call to *free()*. For *fwscanf()*, this is memory allocated via use of  
 37680 the 'm' assignment-allocation character.

#### 37681 RATIONALE

37682 None.

#### 37683 FUTURE DIRECTIONS

37684 None.

#### 37685 SEE ALSO

37686 [Section 2.5](#) (on page 521), *getwc()*, *fwprintf()*, *setlocale()*, *wcstod()*, *wcstol()*, *wcstoul()*, *wcrtomb()*

37687 [XBD Chapter 7](#) (on page 127), [<inttypes.h>](#), [<stdio.h>](#), [<wchar.h>](#)

#### 37688 CHANGE HISTORY

37689 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 37690 (E).

#### 37691 Issue 6

37692 The normative text is updated to avoid use of the term "must" for application requirements.

37693 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 37694 • The prototypes for *fwscanf()* and *swscanf()* are updated.
- 37695 • The DESCRIPTION is updated.
- 37696 • The hh, ll, j, t, and z length modifiers are added.
- 37697 • The a, A, and F conversion characters are added.

37698 The DESCRIPTION is updated to use the terms "conversion specifier" and "conversion  
 37699 specification" consistently.

#### 37700 Issue 7

37701 Austin Group Interpretation 1003.1-2001 #170 is applied.

37702 SD5-XSH-ERN-132 is applied, adding the assignment-allocation character 'm'.

37703 Functionality relating to the "%n\$" form of conversion specification is moved from the XSI  
 37704 option to the Base.

37705 Changes are made related to support for finegrained timestamps.

37706 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
 37707 *malloc()*.

37708 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0229 [302] and XSH/TC1-2008/0230  
 37709 [14] are applied.

37710 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0148 [73], XSH/TC2-2008/0149 [823],  
 37711 and XSH/TC2-2008/0150 [936] are applied.

37712 **Issue 8**

37713 Austin Group Defect 1163 is applied, clarifying the handling of white space in the format string.

37714 Austin Group Defect 1173 is applied, clarifying the description of the assignment-allocation  
37715 character 'm'.

37716 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
37717 standard.

37718 Austin Group Defect 1375 is applied, changing “terminating null wide character” to  
37719 “terminating null character (or wide character)” and changing the first occurrence of **wchar\_t** in  
37720 the descriptions of the `s` and `[` conversion specifiers to **char**.

37721 **NAME**

37722        gai\_strerror — address and name information error description

37723 **SYNOPSIS**

37724        #include &lt;netdb.h&gt;

37725        const char \*gai\_strerror(int *ecode*);37726 **DESCRIPTION**37727        The *gai\_strerror()* function shall return a text string describing an error value for the *getaddrinfo()*  
37728        and *getnameinfo()* functions listed in the <netdb.h> header.37729        When the *ecode* argument is one of the following values listed in the <netdb.h> header:

37730        [EAI_AGAIN]	[EAI_NONAME]
37731        [EAI_BADFLAGS]	[EAI_OVERFLOW]
37732        [EAI_FAIL]	[EAI_SERVICE]
37733        [EAI_FAMILY]	[EAI_SOCKTYPE]
37734        [EAI_MEMORY]	[EAI_SYSTEM]

37735        the function return value shall point to a string describing the error. If the argument is not one  
37736        of those values, the function shall return a pointer to a string whose contents indicate an  
37737        unknown error.37738 **RETURN VALUE**37739        Upon successful completion, *gai\_strerror()* shall return a pointer to an implementation-defined  
37740        string.37741 **ERRORS**

37742        No errors are defined.

37743 **EXAMPLES**

37744        None.

37745 **APPLICATION USAGE**

37746        None.

37747 **RATIONALE**

37748        None.

37749 **FUTURE DIRECTIONS**

37750        None.

37751 **SEE ALSO**37752        [\*freeaddrinfo\(\)\*](#)

37753        XBD &lt;netdb.h&gt;

37754 **CHANGE HISTORY**

37755        First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37756        The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from  
37757        **char \*** to **const char \***. This is for coordination with the IPnG Working Group.37758        IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the  
37759        [EAI\_OVERFLOW] error code.

37760 **NAME**

37761           getaddrinfo — get address information

37762 **SYNOPSIS**

37763           #include <sys/socket.h>

37764           #include <netdb.h>

37765           int getaddrinfo(const char \*restrict *nodename*,

37766                           const char \*restrict *servname*,

37767                           const struct addrinfo \*restrict *hints*,

37768                           struct addrinfo \*\*restrict *res*);

37769 **DESCRIPTION**

37770           Refer to [freeaddrinfo\(\)](#).

37771 **NAME**

37772        getc — get a byte from a stream

37773 **SYNOPSIS**

37774        #include &lt;stdio.h&gt;

37775        int getc(FILE \*stream);

37776 **DESCRIPTION**

37777 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
37778        conflict between the requirements described here and the ISO C standard is unintentional. This  
37779        volume of POSIX.1-2024 defers to the ISO C standard.

37780        The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it  
37781        may evaluate *stream* more than once, so the argument should never be an expression with side-  
37782        effects.

37783 **RETURN VALUE**37784        Refer to *fgetc()*.37785 **ERRORS**37786        Refer to *fgetc()*.37787 **EXAMPLES**

37788        None.

37789 **APPLICATION USAGE**

37790        If the integer value returned by *getc()* is stored into a variable of type **char** and then compared  
37791        against the integer constant EOF, the comparison may never succeed, because sign-extension of  
37792        a variable of type **char** on widening to integer is implementation-defined.

37793        Since it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with  
37794        side-effects. In particular, *getc(\*f++)* does not necessarily work as expected. Therefore, use of this  
37795        function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.

37796 **RATIONALE**

37797        None.

37798 **FUTURE DIRECTIONS**

37799        None.

37800 **SEE ALSO**37801        [Section 2.5](#) (on page 521), *fgetc()*

37802        XBD &lt;stdio.h&gt;

37803 **CHANGE HISTORY**

37804        First released in Issue 1. Derived from Issue 1 of the SVID.

37805 **Issue 7**

37806        POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0231 [14] is applied.

37807 **NAME**

37808        getc\_unlocked, getchar\_unlocked, putc\_unlocked, putchar\_unlocked — stdio with explicit client  
37809        locking

37810 **SYNOPSIS**

```
37811 CX       #include <stdio.h>
37812       int getc_unlocked(FILE *stream);
37813       int getchar_unlocked(void);
37814       int putc_unlocked(int c, FILE *stream);
37815       int putchar_unlocked(int c);
```

37816 **DESCRIPTION**

37817        Versions of the functions *getc()*, *getchar()*, *putc()*, and *putchar()* respectively named  
37818        *getc\_unlocked()*, *getchar\_unlocked()*, *putc\_unlocked()*, and *putchar\_unlocked()* shall be provided  
37819        which are functionally equivalent to the original versions, with the exception that they are not  
37820        required to be implemented in a fully thread-safe manner. They shall be thread-safe when used  
37821        within a scope protected by *flockfile()* (or *ftrylockfile()*) and *funlockfile()*. These functions can  
37822        safely be used in a multi-threaded program if and only if they are called while the invoking  
37823        thread owns the (**FILE \***) object, as is the case after a successful call to the *flockfile()* or  
37824        *ftrylockfile()* functions.

37825        If *getc\_unlocked()* or *putc\_unlocked()* are implemented as macros they may evaluate *stream* more  
37826        than once, so the *stream* argument should never be an expression with side-effects.

37827 **RETURN VALUE**

37828        See *getc()*, *getchar()*, *putc()*, and *putchar()*.

37829 **ERRORS**

37830        See *getc()*, *getchar()*, *putc()*, and *putchar()*.

37831 **EXAMPLES**

37832        None.

37833 **APPLICATION USAGE**

37834        Since they may be implemented as macros, *getc\_unlocked()* and *putc\_unlocked()* may treat  
37835        incorrectly a *stream* argument with side-effects. In particular, *getc\_unlocked(\*f++)* and  
37836        *putc\_unlocked(c,\*f++)* do not necessarily work as expected. Therefore, use of these functions in  
37837        such situations should be preceded by the following statement as appropriate:

```
37838        #undef getc_unlocked
37839        #undef putc_unlocked
```

37840 **RATIONALE**

37841        Some I/O functions are typically implemented as macros for performance reasons (for example,  
37842        *putc()* and *getc()*). For safety, they need to be synchronized, but it is often too expensive to  
37843        synchronize on every character. Nevertheless, it was felt that the safety concerns were more  
37844        important; consequently, the *getc()*, *getchar()*, *putc()*, and *putchar()* functions are required to be  
37845        thread-safe. However, unlocked versions are also provided with names that clearly indicate the  
37846        unsafe nature of their operation but can be used to exploit their higher performance. These  
37847        unlocked versions can be safely used only within explicitly locked program regions, using  
37848        exported locking primitives. In particular, a sequence such as:

```
37849        flockfile(fileptr);
37850        putc_unlocked('1', fileptr);
37851        putc_unlocked('\n', fileptr);
37852        fprintf(fileptr, "Line 2\n");
```

37853 `funlockfile(fileptr);`

37854 is permissible, and results in the text sequence:

37855 1

37856 Line 2

37857 being printed without being interspersed with output from other threads.

37858 It would be wrong to have the standard names such as `getc()`, `putc()`, and so on, map to the  
 37859 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still  
 37860 want to inspect all uses of `getc()`, `putc()`, and so on, by hand when converting existing code.  
 37861 Choosing the safe bindings as the default, at least, results in correct code and maintains the  
 37862 “atomicity at the function” invariant. To do otherwise would introduce gratuitous  
 37863 synchronization errors into converted code. Other routines that modify the `stdio (FILE *)`  
 37864 structures or buffers are also safely synchronized.

37865 Note that there is no need for functions of the form `getc_locked()`, `putc_locked()`, and so on, since  
 37866 this is the functionality of `getc()`, `putc()`, *et al.* It would be inappropriate to use a feature test  
 37867 macro to switch a macro definition of `getc()` between `getc_locked()` and `getc_unlocked()`, since the  
 37868 ISO C standard requires an actual function to exist, a function whose behavior could not be  
 37869 changed by the feature test macro. Also, providing both the `xxx_locked()` and `xxx_unlocked()`  
 37870 forms leads to the confusion of whether the suffix describes the behavior of the function or the  
 37871 circumstances under which it should be used.

37872 Three additional routines, `flockfile()`, `ftrylockfile()`, and `funlockfile()` (which may be macros), are  
 37873 provided to allow the user to delineate a sequence of I/O statements that are executed  
 37874 synchronously.

37875 The `ungetc()` function is infrequently called relative to the other functions/macros so no  
 37876 unlocked variation is needed.

#### 37877 FUTURE DIRECTIONS

37878 None.

#### 37879 SEE ALSO

37880 [Section 2.5](#) (on page 521), `flockfile()`, `getc()`, `getchar()`, `putc()`, `putchar()`

37881 XBD [<stdio.h>](#)

#### 37882 CHANGE HISTORY

37883 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 37884 Issue 6

37885 These functions are marked as part of the Thread-Safe Functions option.

37886 The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing  
 37887 how applications should be written to avoid the case when the functions are implemented as  
 37888 macros.

#### 37889 Issue 7

37890 The `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`, and `putchar_unlocked()` functions are  
 37891 moved from the Thread-Safe Functions option to the Base.

37892 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0232 [395], XSH/TC1-2008/0233 [395],  
 37893 XSH/TC1-2008/0234 [395], and XSH/TC1-2008/0235 [14] are applied.

37894 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0151 [826] is applied.

37895 **NAME**37896            getchar — get a byte from a *stdin* stream37897 **SYNOPSIS**

37898            #include &lt;stdio.h&gt;

37899            int getchar(void);

37900 **DESCRIPTION**

37901 **CX**        The functionality described on this reference page is aligned with the ISO C standard. Any  
37902 conflict between the requirements described here and the ISO C standard is unintentional. This  
37903 volume of POSIX.1-2024 defers to the ISO C standard.

37904            The *getchar()* function shall be equivalent to *getc(stdin)*.37905 **RETURN VALUE**37906            Refer to *fgetc()*.37907 **ERRORS**37908            Refer to *fgetc()*.37909 **EXAMPLES**

37910            None.

37911 **APPLICATION USAGE**

37912            If the integer value returned by *getchar()* is stored into a variable of type **char** and then  
37913 compared against the integer constant EOF, the comparison may never succeed, because sign-  
37914 extension of a variable of type **char** on widening to integer is implementation-defined.

37915 **RATIONALE**

37916            None.

37917 **FUTURE DIRECTIONS**

37918            None.

37919 **SEE ALSO**37920            [Section 2.5](#) (on page 521), *getc()*

37921            XBD &lt;stdio.h&gt;

37922 **CHANGE HISTORY**

37923            First released in Issue 1. Derived from Issue 1 of the SVID.

37924 **Issue 7**

37925            POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0236 [14] is applied.



37926 **NAME**

37927        getchar\_unlocked — stdio with explicit client locking

37928 **SYNOPSIS**

37929 CX        #include &lt;stdio.h&gt;

37930        int getchar\_unlocked(void);

37931 **DESCRIPTION**37932        Refer to *getc\_unlocked()*.

37933 **NAME**

37934           getcwd — get the pathname of the current working directory

37935 **SYNOPSIS**

37936           #include &lt;unistd.h&gt;

37937           char \*getcwd(char \*buf, size\_t size);

37938 **DESCRIPTION**

37939           The *getcwd()* function shall place an absolute pathname of the current working directory in the  
 37940           array pointed to by *buf*, and return *buf*. The pathname shall contain no components that are dot  
 37941           or dot-dot, or are symbolic links.

37942           If there are multiple pathnames that *getcwd()* could place in the array pointed to by *buf*, one  
 37943           beginning with a single <slash> character and one or more beginning with two <slash>  
 37944           characters, then *getcwd()* shall place the pathname beginning with a single <slash> character in  
 37945           the array. The pathname shall not contain any unnecessary <slash> characters after the leading  
 37946           one or two <slash> characters.

37947           The *size* argument is the size in bytes of the character array pointed to by the *buf* argument. If *buf*  
 37948           is a null pointer, the behavior of *getcwd()* is unspecified.

37949 **RETURN VALUE**

37950           Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall  
 37951           return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by  
 37952           *buf* are then undefined.

37953 **ERRORS**37954           The *getcwd()* function shall fail if:37955           [EINVAL]           The *size* argument is 0.37956           [ERANGE]           The *size* argument is greater than 0, but is smaller than the length of the string  
37957           +1.37958           The *getcwd()* function may fail if:37959           [EACCES]           Search permission was denied for the current directory, or read or search  
37960           permission was denied for a directory above the current directory in the file  
37961           hierarchy.

37962           [ENOMEM]           Insufficient storage space is available.

37963 **EXAMPLES**

37964           The following example uses {PATH\_MAX} as the initial buffer size (unless it is indeterminate or  
 37965           very large), and calls *getcwd()* with progressively larger buffers until it does not give an  
 37966           [ERANGE] error.

37967           #include &lt;stdlib.h&gt;

37968           #include &lt;errno.h&gt;

37969           #include &lt;unistd.h&gt;

37970           ...

37971           long path\_max;

37972           size\_t size;

37973           char \*buf;

37974           char \*ptr;

37975           path\_max = pathconf(".", \_PC\_PATH\_MAX);

37976           if (path\_max == -1)

```

37977         size = 1024;
37978     else if (path_max > 10240)
37979         size = 10240;
37980     else
37981         size = path_max;
37982     for (buf = ptr = NULL; ptr == NULL; size *= 2)
37983     {
37984         if ((buf = realloc(buf, size)) == NULL)
37985         {
37986             ... handle error ...
37987         }
37988         ptr = getcwd(buf, size);
37989         if (ptr == NULL && errno != ERANGE)
37990         {
37991             ... handle error ...
37992         }
37993     }
37994     ...
37995     free (buf);

```

#### 37996 APPLICATION USAGE

37997 If the pathname obtained from *getcwd()* is longer than {PATH\_MAX} bytes, it could produce an  
37998 [ENAMETOOLONG] error if passed to *chdir()*. Therefore, in order to return to that directory it  
37999 may be necessary to break the pathname into sections shorter than {PATH\_MAX} bytes and call  
38000 *chdir()* on each section in turn (the first section being an absolute pathname and subsequent  
38001 sections being relative pathnames). A simpler way to handle saving and restoring the working  
38002 directory when it may be deeper than {PATH\_MAX} bytes in the file hierarchy is to use a file  
38003 descriptor and *fchdir()*, rather than *getcwd()* and *chdir()*. However, the two methods do have  
38004 some differences. The *fchdir()* approach causes the program to restore a working directory even  
38005 if it has been renamed in the meantime, whereas the *chdir()* approach restores to a directory with  
38006 the same name as the original, even if the directories were renamed in the meantime. Since the  
38007 *fchdir()* approach does not access parent directories, it can succeed when *getcwd()* would fail  
38008 due to permissions problems. In applications conforming to earlier versions of this standard, it  
38009 was not possible to use the *fchdir()* approach when the working directory is searchable but not  
38010 readable, as the only way to open a directory was with O\_RDONLY, whereas the *getcwd()*  
38011 approach can succeed in this case.

#### 38012 RATIONALE

38013 Having *getcwd()* take no arguments and instead use the *malloc()* function to produce space for  
38014 the returned argument was considered. The advantage is that *getcwd()* knows how big the  
38015 working directory pathname is and can allocate an appropriate amount of space. But the  
38016 programmer would have to use the *free()* function to free the resulting object, or each use of  
38017 *getcwd()* would further reduce the available memory. Finally, *getcwd()* is taken from the SVID  
38018 where it has the two arguments used in this volume of POSIX.1-2024.

38019 The older function *getwd()* was rejected for use in this context because it had only a buffer  
38020 argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except  
38021 to depend on the programmer to provide a large enough buffer.

38022 On some implementations, if *buf* is a null pointer, *getcwd()* may obtain *size* bytes of memory  
38023 using *malloc()*. In this case, the pointer returned by *getcwd()* may be used as the argument in a  
38024 subsequent call to *free()*. Invoking *getcwd()* with *buf* as a null pointer is not recommended in  
38025 conforming applications.

38026 Earlier implementations of `getcwd()` sometimes generated pathnames like  
38027 `"../../../../subdirname"` internally, using them to explore the path of ancestor directories  
38028 back to the root. If one of these internal pathnames exceeded `{PATH_MAX}` in length, the  
38029 implementation could fail with *errno* set to `[ENAMETOOLONG]`. This is no longer allowed.

38030 If a program is operating in a directory where some (grand)parent directory does not permit  
38031 reading, `getcwd()` may fail, as in most implementations it must read the directory to determine  
38032 the name of the file. This can occur if search, but not read, permission is granted in an  
38033 intermediate directory, or if the program is placed in that directory by some more privileged  
38034 process (for example, login). Including the `[EACCES]` error condition makes the reporting of the  
38035 error consistent and warns the application developer that `getcwd()` can fail for reasons beyond  
38036 the control of the application developer or user. Some implementations can avoid this  
38037 occurrence (for example, by implementing `getcwd()` using `pwd`, where `pwd` is a set-user-root  
38038 process), thus the error was made optional. Since this volume of POSIX.1-2024 permits the  
38039 addition of other errors, this would be a common addition and yet one that applications could  
38040 not be expected to deal with without this addition.

#### 38041 FUTURE DIRECTIONS

38042 None.

#### 38043 SEE ALSO

38044 [\*malloc\(\)\*](#)

38045 XBD [`<unistd.h>`](#)

#### 38046 CHANGE HISTORY

38047 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 38048 Issue 6

38049 The following new requirements on POSIX implementations derive from alignment with the  
38050 Single UNIX Specification:

- 38051 • The `[ENOMEM]` optional error condition is added.

#### 38052 Issue 7

38053 Austin Group Interpretation 1003.1-2001 #140 is applied, changing the text for consistency with  
38054 the `pwd` utility, adding text to address the case where the current directory is deeper in the file  
38055 hierarchy than `{PATH_MAX}` bytes, and adding the requirements relating to pathnames  
38056 beginning with two `<slash>` characters.

38057 **NAME**

38058 getdate — convert user format date and time

38059 **SYNOPSIS**

```
38060 XSI      #include <time.h>
38061      struct tm *getdate(const char *string);
```

38062 **DESCRIPTION**

38063 The *getdate()* function shall convert a string representation of a date or time into a broken-down  
38064 time.

38065 The external variable or macro *getdate\_err*, which has type **int**, is used by *getdate()* to return error  
38066 values. It is unspecified whether *getdate\_err* is a macro or an identifier declared with external  
38067 linkage, and whether or not it is a modifiable lvalue. If a macro definition is suppressed in order  
38068 to access an actual object, or a program defines an identifier with the name *getdate\_err*, the  
38069 behavior is undefined.

38070 Templates are used to parse and interpret the input string. The templates are contained in a text  
38071 file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to  
38072 indicate the full pathname of the file that contains the templates. The first line in the template  
38073 that matches the input specification is used for interpretation and conversion into the internal  
38074 time format.

38075 The following conversion specifications shall be supported:

38076	%%	Equivalent to %.
38077	%a	Abbreviated weekday name.
38078	%A	Full weekday name.
38079	%b	Abbreviated month name.
38080	%B	Full month name.
38081	%c	Locale's appropriate date and time representation.
38082	%C	Century number [00,99]; leading zeros are permitted but not required.
38083	%d	Day of month [01,31]; the leading 0 is optional.
38084	%D	Date as %m/%d/%y.
38085	%e	Equivalent to %d.
38086	%h	Abbreviated month name.
38087	%H	Hour [00,23].
38088	%I	Hour [01,12].
38089	%m	Month number [01,12].
38090	%M	Minute [00,59].
38091	%n	Equivalent to <newline>.
38092	%p	Locale's equivalent of either AM or PM.
38093	%r	The locale's appropriate representation of time in 12-hour clock notation, if the 12-hour 38094 format is supported in the locale (see XBD <a href="#">Section 7.3.5</a> , on page 152). In the POSIX 38095 locale, this shall be equivalent to %I:%M:%S %p.

38096	%R	Time as %H:%M.
38097	%S	Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap second data has to come from some external source.
38098		
38099		
38100	%t	Equivalent to <tab>.
38101	%T	Time as %H:%M:%S.
38102	%w	Weekday number (Sunday = [0,6]).
38103	%x	Locale's appropriate date representation.
38104	%X	Locale's appropriate time representation.
38105	%y	Year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.
38106		
38107		
38108		<b>Note:</b> It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
38109		
38110		
38111	%Y	Year as "ccyy" (for example, 2001).
38112	%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result. The <i>getdate()</i> function calculates an expected timezone based on information supplied to the function (such as the hour, day, and month).
38113		
38114		
38115		
38116		The match between the template and input specification performed by <i>getdate()</i> shall be case-insensitive.
38117		
38118		The month and weekday names can consist of any combination of upper and lowercase letters. The process can request that the input date or time specification be in a specific language by setting the <i>LC_TIME</i> category (see <i>setlocale()</i> ).
38119		
38120		
38121		Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most two digits are allowed for those descriptors, including leading zeros. Extra white space in either the template file or in <i>string</i> shall be ignored.
38122		
38123		
38124		The results are undefined if the conversion specifications %c, %x, and %X include unsupported conversion specifications.
38125		
38126		The following rules apply for converting the input specification into the internal format:
38127		• If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on the current local time as if <i>localtime()</i> had been called.
38128		
38129		
38130		• If only the weekday is given, the day chosen shall be the day, starting with today and moving into the future, which first matches the named day.
38131		
38132		• If only the month (and no year) is given, the month chosen shall be the month, starting with the current month and moving into the future, which first matches the named month. The first day of the month shall be assumed if no day is given.
38133		
38134		
38135		• If no hour, minute, and second are given, the current hour, minute, and second shall be assumed.
38136		

- 38137           • If no date is given, the hour chosen shall be the hour, starting with the current hour and  
38138           moving into the future, which first matches the named hour.

38139           If a conversion specification in the DATEMSK file does not correspond to one of the conversion  
38140           specifications above, the behavior is unspecified.

38141           The *getdate()* function need not be thread-safe.

#### 38142 RETURN VALUE

38143           Upon successful completion, *getdate()* shall return a pointer to a **struct tm**. Otherwise, it shall  
38144           return a null pointer and set *getdate\_err* to indicate the error.

#### 38145 ERRORS

38146           The *getdate()* function shall fail in the following cases, setting *getdate\_err* to the value shown in  
38147           the list below. Any changes to *errno* are unspecified.

- 38148           1. The *DATEMSK* environment variable is null or undefined.
- 38149           2. The template file cannot be opened for reading.
- 38150           3. Failed to get file status information.
- 38151           4. The template file is not a regular file.
- 38152           5. An I/O error is encountered while reading the template file.
- 38153           6. Memory allocation failed (not enough memory available).
- 38154           7. There is no line in the template that matches the input.
- 38155           8. Invalid input specification. For example, February 31; or a time is specified that cannot be  
38156           represented in a **time\_t** (representing the time in seconds since the Epoch).

#### 38157 EXAMPLES

- 38158           1. The following example shows the possible contents of a template:

```
38159           %m
38160           %A %B %d, %Y, %H:%M:%S
38161           %A
38162           %B
38163           %m/%d/%y %I %p
38164           %d, %m, %Y %H:%M
38165           at %A the %dst of %B in %Y
38166           run job at %I %p, %B %dnd
38167           %A den %d. %B %Y %H.%M Uhr
```

- 38168           2. The following are examples of valid input specifications for the template in Example 1:

```
38169           getdate("10/1/87 4 PM");
38170           getdate("Friday");
38171           getdate("Friday September 18, 1987, 10:30:30");
38172           getdate("24,9,1986 10:30");
38173           getdate("at monday the 1st of december in 1986");
38174           getdate("run job at 3 PM, december 2nd");
```

38175           If the *LC\_TIME* category is set to a German locale that includes *freitag* as a weekday name  
38176           and *oktober* as a month name, the following would be valid:

```
38177           getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

38178 3. The following example shows how local date and time specification can be defined in the  
38179 template:

Invocation	Line in Template
getdate("11/27/86")	%m/%d/%y
getdate("27.11.86")	%d.%m.%y
getdate("86-11-27")	%y-%m-%d
getdate("Friday 12:00:00")	%A %H:%M:%S

38185 4. The following examples help to illustrate the above rules assuming that the current date  
38186 is Mon Sep 22 12:19:47 EDT 1986 and the *LC\_TIME* category is set to the default C or  
38187 POSIX locale:

Input	Line in Template	Date
Mon	%a	Mon Sep 22 12:19:47 EDT 1986
Sun	%a	Sun Sep 28 12:19:47 EDT 1986
Fri	%a	Fri Sep 26 12:19:47 EDT 1986
September	%B	Mon Sep 1 12:19:47 EDT 1986
January	%B	Thu Jan 1 12:19:47 EST 1987
December	%B	Mon Dec 1 12:19:47 EST 1986
Sep Mon	%b %a	Mon Sep 1 12:19:47 EDT 1986
Jan Fri	%b %a	Fri Jan 2 12:19:47 EST 1987
Dec Mon	%b %a	Mon Dec 1 12:19:47 EST 1986
Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:47 EST 1989
Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

### 38203 APPLICATION USAGE

38204 Although historical versions of *getdate()* did not require that **<time.h>** declare the external  
38205 variable *getdate\_err*, this volume of POSIX.1-2024 does require it. The standard developers  
38206 encourage applications to remove declarations of *getdate\_err* and instead incorporate the  
38207 declaration by including **<time.h>**.

38208 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

### 38209 RATIONALE

38210 In standard locales, the conversion specifications %c, %x, and %X do not include unsupported  
38211 conversion specifiers and so the text regarding results being undefined is not a problem in that  
38212 case.

### 38213 FUTURE DIRECTIONS

38214 None.

### 38215 SEE ALSO

38216 *ctime()*, *localtime()*, *setlocale()*, *strftime()*, *times()*

38217 XBD **<time.h>**

### 38218 CHANGE HISTORY

38219 First released in Issue 4, Version 2.



**Issue 5**

- 38220 Moved from X/OPEN UNIX extension to BASE.
- 38221
- 38222 The last paragraph of the DESCRIPTION is added.
- 38223 The %C conversion specification is added, and the exact meaning of the %y conversion  
38224 specification is clarified in the DESCRIPTION.
- 38225 A note indicating that this function need not be reentrant is added to the DESCRIPTION.
- 38226 The %R conversion specification is changed to follow historical practice.

**Issue 6**

- 38227 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since  
38228 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*  
38229 functions.
- 38230
- 38231 The description of %S is updated so that the valid range is [00,60] rather than [00,61].
- 38232 The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors  
38233 for consistency with other functions.

**Issue 7**

- 38234 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 38235
- 38236 The description of the *getdate\_err* value is expanded.
- 38237 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0152 [796] is applied.

**Issue 8**

- 38238 Austin Group Defect 1307 is applied, changing the %r conversion in relation to locales that do  
38239 not support the 12-hour clock format.
- 38240

38241 **NAME**38242 getdelim, getline — read a delimited record from *stream*38243 **SYNOPSIS**

```
38244 CX #include <stdio.h>
38245     ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
38246                   int delimiter, FILE *restrict stream);
38247     ssize_t getline(char **restrict lineptr, size_t *restrict n,
38248                   FILE *restrict stream);
```

38249 **DESCRIPTION**

38250 The *getdelim()* function shall read from *stream* until it encounters a character matching the  
 38251 *delimiter* character. The *delimiter* argument is an **int**, the value of which the application shall  
 38252 ensure is a character representable as an **unsigned char** of equal value that terminates the read  
 38253 process. If the *delimiter* argument has any other value, the behavior is undefined.

38254 The application shall ensure that *\*lineptr* is a valid argument that could be passed to the *free()*  
 38255 function. If *\*n* is non-zero, the application shall ensure that *\*lineptr* either points to an object of  
 38256 size at least *\*n* bytes, or is a null pointer.

38257 If *\*lineptr* is a null pointer or if the object pointed to by *\*lineptr* is of insufficient size, an object  
 38258 shall be allocated as if by *malloc()* or the object shall be reallocated as if by *realloc()*, respectively,  
 38259 such that the object is large enough to hold the characters to be written to it, including the  
 38260 terminating NUL, and *\*n* shall be set to the new size. If the object was allocated, or if the  
 38261 reallocation operation moved the object, *\*lineptr* shall be updated to point to the new object or  
 38262 new location. The characters read, including any delimiter, shall be stored in the object, and a  
 38263 terminating NUL added when the delimiter or end-of-file is encountered.

38264 The *getline()* function shall be equivalent to the *getdelim()* function with the *delimiter* character  
 38265 equal to the <newline> character.

38266 The *getdelim()* and *getline()* functions may mark the last data access timestamp of the file  
 38267 associated with *stream* for update. The last data access timestamp shall be marked for update by  
 38268 the first successful execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*,  
 38269 *getline()*, or *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

38270 **RETURN VALUE**

38271 Upon successful completion, the *getline()* and *getdelim()* functions shall return the number of  
 38272 bytes written into the buffer, including the delimiter character if one was encountered before  
 38273 EOF, but excluding the terminating NUL character. If the end-of-file indicator for the stream is  
 38274 set, or if no characters were read and the stream is at end-of-file, the end-of-file indicator for the  
 38275 stream shall be set and the function shall return  $-1$ . If an error occurs, the error indicator for the  
 38276 stream shall be set, and the function shall return  $-1$  and set *errno* to indicate the error.

38277 **ERRORS**

38278 For the conditions under which the *getdelim()* and *getline()* functions shall fail and may fail, refer  
 38279 to *fgetc()*.

38280 In addition, these functions shall fail if:

38281 [EINVAL] *lineptr* or *n* is a null pointer.

38282 [ENOMEM] Insufficient memory is available.

38283 These functions may fail if:

38284 [EOVERFLOW] The number of bytes to be written into the buffer, including the delimiter  
38285 character (if encountered), would exceed {SSIZE\_MAX}.

### 38286 EXAMPLES

```
38287 #include <stdio.h>
38288 #include <stdlib.h>
38289 int main(void)
38290 {
38291     FILE *fp;
38292     char *line = NULL;
38293     size_t len = 0;
38294     ssize_t read;
38295     fp = fopen("/etc/motd", "r");
38296     if (fp == NULL)
38297         exit(1);
38298     while ((read = getline(&line, &len, fp)) != -1) {
38299         printf("Retrieved line of length %zu :\n", read);
38300         printf("%s", line);
38301     }
38302     if (ferror(fp)) {
38303         /* handle error */
38304     }
38305     free(line);
38306     fclose(fp);
38307     return 0;
38308 }
```

### 38309 APPLICATION USAGE

38310 Setting *\*lineptr* to a null pointer and *\*n* to zero are allowed and a recommended way to start  
38311 parsing a file.

38312 The *ferror()* or *feof()* functions should be used to distinguish between an error condition and an  
38313 end-of-file condition.

38314 Although a null terminator is always supplied after the line, note that *strlen(\*lineptr)* will be  
38315 smaller than the return value if the line contains embedded NUL characters.

### 38316 RATIONALE

38317 These functions are widely used to solve the problem that the *fgets()* function has with long  
38318 lines. The functions automatically enlarge the target buffers if needed. These are especially  
38319 useful since they reduce code needed for applications.

### 38320 FUTURE DIRECTIONS

38321 None.

### 38322 SEE ALSO

38323 [Section 2.5](#) (on page 521), *fgetc()*, *fgets()*, *free()*, *malloc()*, *realloc()*

38324 XBD [<stdio.h>](#)

### 38325 CHANGE HISTORY

38326 First released in Issue 7.

38327 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0237 [14] is applied.

38328 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0153 [569], XSH/TC2-2008/0154 [571],  
38329 and XSH/TC2-2008/0155 [570] are applied.

38330 **Issue 8**

38331 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

38332 Austin Group Defect 1621 is applied, changing ``NUL terminator'' to ``null terminator''.

**38333 NAME**

38334 getegid — get the effective group ID

**38335 SYNOPSIS**

38336 #include <unistd.h>  
38337 gid\_t getegid(void);

**38338 DESCRIPTION**

38339 The *getegid()* function shall return the effective group ID of the calling process. The *getegid()*  
38340 function shall not modify *errno*.

**38341 RETURN VALUE**

38342 The *getegid()* function shall always be successful and no return value is reserved to indicate an  
38343 error.

**38344 ERRORS**

38345 No errors are defined.

**38346 EXAMPLES**

38347 None.

**38348 APPLICATION USAGE**

38349 None.

**38350 RATIONALE**

38351 In a conforming environment, *getegid()* will always succeed. It is possible for implementations to  
38352 provide an extension where a process in a non-conforming environment will not be associated  
38353 with a user or group ID. It is recommended that such implementations return (**gid\_t**)-1 and set  
38354 *errno* to indicate such an environment; doing so does not violate this standard, since such an  
38355 environment is already an extension.

**38356 FUTURE DIRECTIONS**

38357 None.

**38358 SEE ALSO**

38359 *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*,  
38360 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

38361 XBD <[sys/types.h](#)>, <[unistd.h](#)>

**38362 CHANGE HISTORY**

38363 First released in Issue 1. Derived from Issue 1 of the SVID.

**38364 Issue 6**

38365 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.

38366 The following new requirements on POSIX implementations derive from alignment with the  
38367 Single UNIX Specification:

- 38368 • The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was  
38369 required for conforming implementations of previous POSIX specifications, it was not  
38370 required for UNIX applications.

**38371 Issue 7**

38372 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0156 [511] is applied.

38373 **Issue 8**

38374

38375

Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to SEE ALSO.

38376 **NAME**38377 `getentropy` — fill a buffer with random bytes38378 **SYNOPSIS**38379 `#include <unistd.h>`38380 `int getentropy(void *buffer, size_t length);`38381 **DESCRIPTION**

38382 The `getentropy()` function shall write *length* bytes of data starting at the location pointed to by  
38383 *buffer*. The output shall be unpredictable high quality random data, generated by a  
38384 cryptographically secure pseudo-random number generator. The maximum permitted value for  
38385 the *length* argument is given by the {GETENTROPY\_MAX} symbolic constant defined in  
38386 `<limits.h>`.

38387 A successful call to `getentropy()` shall always provide the requested number of bytes of entropy.

38388 **RETURN VALUE**

38389 Upon successful completion, `getentropy()` shall return 0; otherwise, `-1` shall be returned and  
38390 *errno* set to indicate the error.

38391 **ERRORS**

38392 The `getentropy()` function shall fail if:

38393 [EINVAL] The value of *length* is greater than {GETENTROPY\_MAX}.

38394 The `getentropy()` function may fail if:

38395 [ENOSYS] The system does not provide the necessary source of entropy.

38396 **EXAMPLES**

38397 None.

38398 **APPLICATION USAGE**

38399 The intended use of this function is to create a seed for other pseudo-random number  
38400 generators.

38401 **RATIONALE**

38402 The `getentropy()` function is not a cancellation point. (See [Section 2.9.5.2](#) (on page 543).)

38403 **FUTURE DIRECTIONS**

38404 None.

38405 **SEE ALSO**

38406 `drand48()`, `initstate()`, `rand()`

38407 XBD `<limits.h>`, `<unistd.h>`

38408 **CHANGE HISTORY**

38409 First released in Issue 8.

38410 **NAME**

38411            getenv, secure\_getenv — get value of an environment variable

38412 **SYNOPSIS**

38413            #include &lt;stdlib.h&gt;

38414            char \*getenv(const char \*name);

38415 CX         char \*secure\_getenv(const char \*name);

38416 **DESCRIPTION**38417 CX         The functionality described on this reference page is aligned with the ISO C standard. Any  
38418 conflict between the requirements described here and the ISO C standard is unintentional. This  
38419 volume of POSIX.1-2024 defers to the ISO C standard.38420         The *getenv()* function shall search the environment of the calling process (see XBD [Chapter 8](#), on  
38421 page 167) for the environment variable *name* if it exists and return a pointer to the value of the  
38422 environment variable. If the specified environment variable cannot be found, a null pointer shall  
38423 be returned. The application shall ensure that it does not modify the string pointed to by the  
38424 CX         *getenv()* function, unless it is part of a modifiable object previously placed in the environment  
38425 by assigning a new value to *environ*  
38426 XSI         or by using *putenv()*.38427 CX         The pointer returned by *getenv()* shall point to a string within the environment data pointed to  
38428 by *environ*.38429         **Note:**         This requirement is an extension to the ISO C standard, which allows *getenv()* to copy the data  
38430 to an internal buffer.38431         The *secure\_getenv()* function shall be equivalent to *getenv()*, except that it shall return a null  
38432 pointer if the calling process does not meet all of the following security criteria:

- 38433            1. The effective user ID and real user ID of the calling process were equal during program
- 
- 38434 startup.
- 
- 38435            2. The effective group ID and real group ID of the calling process were equal during
- 
- 38436 program startup.
- 
- 38437            3. Additional implementation-defined security criteria.

38438 **RETURN VALUE**38439         Upon successful completion, *getenv()* shall return a pointer to a string containing the value for  
38440 the specified *name*. If the specified *name* cannot be found in the environment of the calling  
38441 process, a null pointer shall be returned.38442 CX         Upon successful completion, *secure\_getenv()* shall return a pointer to a string containing the  
38443 value for the specified *name*. If the specified *name* cannot be found in the environment of the  
38444 calling process, or the calling process does not meet the security criteria listed in DESCRIPTION,  
38445 a null pointer shall be returned.38446 **ERRORS**

38447         No errors are defined.



38448 **EXAMPLES**38449 **Getting the Value of an Environment Variable**

38450 The following example gets the value of the *HOME* environment variable.

```
38451 #include <stdlib.h>
38452 ...
38453 const char *name = "HOME";
38454 char *value;
38455 value = getenv(name);
```

38456 **APPLICATION USAGE**

38457 None.

38458 **RATIONALE**

38459 The *clearenv()* function was considered but rejected. The *putenv()* function has now been  
38460 included for alignment with the Single UNIX Specification.

38461 Some earlier versions of this standard did not require *getenv()* to be thread-safe because it was  
38462 allowed to return a value pointing to an internal buffer. However, this behavior allowed by the  
38463 ISO C standard is no longer allowed by POSIX.1. POSIX.1 requires the environment data to be  
38464 available through *environ[]*, so there is no reason why *getenv()* can't return a pointer to the actual  
38465 data instead of a copy. Therefore *getenv()* is now required to be thread-safe (except when  
38466 another thread modifies the environment).

38467 Conforming applications are required not to directly modify the pointers to which *environ*  
38468 points, but to use only the *setenv()*, *unsetenv()*, and *putenv()* functions, or assignment to *environ*  
38469 itself, to manipulate the process environment. This constraint allows the implementation to  
38470 properly manage the memory it allocates. This enables the implementation to free any space it  
38471 has allocated to strings (and perhaps the pointers to them) stored in *environ* when *unsetenv()* is  
38472 called. A C runtime start-up procedure (that which invokes *main()* and perhaps initializes  
38473 *environ*) can also initialize a flag indicating that none of the environment has yet been copied to  
38474 allocated storage, or that the separate table has not yet been initialized. If the application  
38475 switches to a complete new environment by assigning a new value to *environ*, this can be  
38476 detected by *getenv()*, *setenv()*, *unsetenv()*, or *putenv()* and the implementation can at that point  
38477 reinitialize based on the new environment. (This may include copying the environment strings  
38478 into a new array and assigning *environ* to point to it.)

38479 In fact, for higher performance of *getenv()*, implementations that do not provide *putenv()* could  
38480 also maintain a separate copy of the environment in a data structure that could be searched  
38481 much more quickly (such as an indexed hash table, or a binary tree), and update both it and the  
38482 linear list at *environ* when *setenv()* or *unsetenv()* is invoked. On implementations that do provide  
38483 *putenv()*, such a copy might still be worthwhile but would need to allow for the fact that  
38484 applications can directly modify the content of environment strings added with *putenv()*. For  
38485 example, if an environment string found by searching the copy is one that was added using  
38486 *putenv()*, the implementation would need to check that the string in *environ* still has the same  
38487 name (and value, if the copy includes values), and whenever searching the copy produces no  
38488 match the implementation would then need to search each environment string in *environ* that  
38489 was added using *putenv()* in case any of them have changed their names and now match. Thus,  
38490 each use of *putenv()* to add to the environment would reduce the speed advantage of having the  
38491 copy.

38492 Performance of *getenv()* can be important for applications which have large numbers of  
38493 environment variables. Typically, applications like this use the environment as a resource

38494 database of user-configurable parameters. The fact that these variables are in the user's shell  
38495 environment usually means that any other program that uses environment variables (such as *ls*,  
38496 which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC\_ALL*, and so on) is  
38497 similarly slowed down by the linear search through the variables.

38498 An implementation that maintains separate data structures, or even one that manages the  
38499 memory it consumes, is not currently required as it was thought it would reduce consensus  
38500 among implementors who do not want to change their historical implementations.

#### 38501 **FUTURE DIRECTIONS**

38502 None.

#### 38503 **SEE ALSO**

38504 *exec*, *putenv()*, *setenv()*, *unsetenv()*

38505 XBD Chapter 8 (on page 167), `<stdlib.h>`

#### 38506 **CHANGE HISTORY**

38507 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 38508 **Issue 5**

38509 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
38510 VALUE section.

38511 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

#### 38512 **Issue 6**

38513 The following changes were made to align with the IEEE P1003.1a draft standard:

- 38514 • References added to the new *setenv()* and *unsetenv()* functions.

38515 The normative text is updated to avoid use of the term "must" for application requirements.

#### 38516 **Issue 7**

38517 Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to *putenv()* may  
38518 also cause the string to be overwritten.

38519 Austin Group Interpretation 1003.1-2001 #148 is applied, adding the FUTURE DIRECTIONS.

38520 Austin Group Interpretation 1003.1-2001 #156 is applied.

38521 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0238 [75,428], XSH/TC1-2008/0239  
38522 [167], and XSH/TC1-2008/0240 [167] are applied.

38523 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0157 [656] is applied.

#### 38524 **Issue 8**

38525 Austin Group Defects 188 and 1394 are applied, changing *getenv()* to be thread-safe.

38526 Austin Group Defect 922 is applied, adding the *secure\_getenv()* function.

38527 **NAME**

38528           geteuid — get the effective user ID

38529 **SYNOPSIS**

38530           #include <unistd.h>

38531           uid\_t geteuid(void);

38532 **DESCRIPTION**

38533           The *geteuid()* function shall return the effective user ID of the calling process. The *geteuid()*  
38534           function shall not modify *errno*.

38535 **RETURN VALUE**

38536           The *geteuid()* function shall always be successful and no return value is reserved to indicate an  
38537           error.

38538 **ERRORS**

38539           No errors are defined.

38540 **EXAMPLES**

38541           None.

38542 **APPLICATION USAGE**

38543           None.

38544 **RATIONALE**

38545           In a conforming environment, *geteuid()* will always succeed. It is possible for implementations  
38546           to provide an extension where a process in a non-conforming environment will not be associated  
38547           with a user or group ID. It is recommended that such implementations return **(uid\_t)-1** and set  
38548           *errno* to indicate such an environment; doing so does not violate this standard, since such an  
38549           environment is already an extension.

38550 **FUTURE DIRECTIONS**

38551           None.

38552 **SEE ALSO**

38553           *getegid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*,  
38554           *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

38555           XBD <[sys/types.h](#)>, <[unistd.h](#)>

38556 **CHANGE HISTORY**

38557           First released in Issue 1. Derived from Issue 1 of the SVID.

38558 **Issue 6**

38559           In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.

38560           The following new requirements on POSIX implementations derive from alignment with the  
38561           Single UNIX Specification:

- 38562           • The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was  
38563           required for conforming implementations of previous POSIX specifications, it was not  
38564           required for UNIX applications.

38565 **Issue 7**

38566           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0158 [511] is applied.

38567 **Issue 8**

38568

38569

Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to SEE ALSO.

38570 **NAME**

38571 getgid — get the real group ID

38572 **SYNOPSIS**

38573 #include &lt;unistd.h&gt;

38574 gid\_t getgid(void);

38575 **DESCRIPTION**38576 The *getgid()* function shall return the real group ID of the calling process. The *getgid()* function  
38577 shall not modify *errno*.38578 **RETURN VALUE**38579 The *getgid()* function shall always be successful and no return value is reserved to indicate an  
38580 error.38581 **ERRORS**

38582 No errors are defined.

38583 **EXAMPLES**

38584 None.

38585 **APPLICATION USAGE**

38586 None.

38587 **RATIONALE**38588 In a conforming environment, *getgid()* will always succeed. It is possible for implementations to  
38589 provide an extension where a process in a non-conforming environment will not be associated  
38590 with a user or group ID. It is recommended that such implementations return (**gid\_t**)-1 and set  
38591 *errno* to indicate such an environment; doing so does not violate this standard, since such an  
38592 environment is already an extension.38593 **FUTURE DIRECTIONS**

38594 None.

38595 **SEE ALSO**38596 *getegid()*, *geteuid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*,  
38597 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*38598 XBD <[sys/types.h](#)>, <[unistd.h](#)>38599 **CHANGE HISTORY**

38600 First released in Issue 1. Derived from Issue 1 of the SVID.

38601 **Issue 6**38602 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.38603 The following new requirements on POSIX implementations derive from alignment with the  
38604 Single UNIX Specification:

- 38605
- The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was  
38606 required for conforming implementations of previous POSIX specifications, it was not  
38607 required for UNIX applications.

38608 **Issue 7**

38609 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0159 [511] is applied.

38610 **Issue 8**

38611

38612

Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to SEE ALSO.

38613 **NAME**

38614       getgrent — get the group database entry

38615 **SYNOPSIS**

```
38616 XSI       #include <grp.h>  
38617       struct group *getgrent(void);
```

38618 **DESCRIPTION**38619       Refer to *endgrent()*.

38620 **NAME**

38621 getgrgid, getgrgid\_r — get group database entry for a group ID

38622 **SYNOPSIS**

```
38623 #include <grp.h>
38624 struct group *getgrgid(gid_t gid);
38625 int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
38626               size_t bufsize, struct group **result);
```

38627 **DESCRIPTION**38628 The *getgrgid()* function shall search the group database for an entry with a matching *gid*.38629 The *getgrgid()* function need not be thread-safe.38630 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*.  
38631 If *getgrgid()* returns a null pointer and *errno* is set to non-zero, an error occurred.

38632 The *getgrgid\_r()* function shall update the **group** structure pointed to by *grp* and store a pointer  
38633 to that structure at the location pointed to by *result*. The structure shall contain an entry from  
38634 the group database with a matching *gid*. Storage referenced by the group structure is allocated  
38635 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to  
38636 *sysconf(\_SC\_GETGR\_R\_SIZE\_MAX)* returns either  $-1$  without changing *errno* or an initial value  
38637 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to  
38638 by *result* on error or if the requested entry is not found.

38639 **RETURN VALUE**

38640 Upon successful completion, *getgrgid()* shall return a pointer to a **struct group** with the structure  
38641 defined in **<grp.h>** with a matching entry if one is found. The *getgrgid()* function shall return a  
38642 null pointer if either the requested entry was not found, or an error occurred. If the requested  
38643 entry was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

38644 The application shall not modify the structure to which the return value points, nor any storage  
38645 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
38646 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
38647 subsequent call to *getgrent()*, *getgrgid()*, or *getgrnam()*. The returned pointer, and pointers  
38648 within the structure, might also be invalidated if the calling thread is terminated.

38649 If successful, the *getgrgid\_r()* function shall return zero; otherwise, an error number shall be  
38650 returned to indicate the error.38651 **ERRORS**38652 The *getgrgid()* and *getgrgid\_r()* functions may fail if:

- |       |          |  |
|-------|----------|--|
| 38653 | [EIO]    | An I/O error has occurred.   |
| 38654 | [EINTR]  | A signal was caught during <i>getgrgid()</i> .                         |
| 38655 | [EMFILE] | All file descriptors available to the process are currently open.      |
| 38656 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

38657 The *getgrgid\_r()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 38658 | [ERANGE] | Insufficient storage was supplied via <i>buffer</i> and <i>bufsize</i> to contain the data to be<br>38659 referenced by the resulting <b>group</b> structure. |
|-------|----------|---|



38660 **EXAMPLES**

38661 Note that `sysconf(_SC_GETGR_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size  
 38662 of the buffer needed to store all the groups returned. This example shows how an application  
 38663 can allocate a buffer of sufficient size to work with `getgr_r()`.

```

38664 long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
38665 size_t len;
38666 if (initlen == -1)
38667     /* Default initial length. */
38668     len = 1024;
38669 else
38670     len = (size_t) initlen;
38671 struct group result;
38672 struct group *resultp;
38673 char *buffer = malloc(len);
38674 if (buffer == NULL)
38675     ...handle error...
38676 int e;
38677 while ((e = getgrgid_r(42, &result, buffer, len, &resultp)) == ERANGE)
38678     {
38679     size_t newlen = 2 * len;
38680     if (newlen < len)
38681         ...handle error...
38682     len = newlen;
38683     char *newbuffer = realloc(buffer, len);
38684     if (newbuffer == NULL)
38685         ...handle error...
38686     buffer = newbuffer;
38687     }
38688 if (e != 0)
38689     ...handle error...
38690 free (buffer);

```

38691 **Finding an Entry in the Group Database**

38692 The following example uses `getgrgid()` to search the group database for a group ID that was  
 38693 previously stored in a `stat` structure, then prints out the group name if it is found. If the group is  
 38694 not found, the program prints the numeric value of the group for the entry.

```

38695 #include <sys/types.h>
38696 #include <grp.h>
38697 #include <stdio.h>
38698 ...
38699 struct stat statbuf;
38700 struct group *grp;
38701 ...
38702 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
38703     printf(" %-8.8s", grp->gr_name);
38704 else
38705     printf(" %-8d", statbuf.st_gid);
38706 ...

```

38707 **APPLICATION USAGE**

38708 The `getgrgid_r()` function is thread-safe and shall return values in a user-supplied buffer instead  
 38709 of possibly using a static data area that may be overwritten by each call.

38710 Portable applications should take into account that it is usual for an implementation to return `-1`  
 38711 from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

38712 **RATIONALE**

38713 None.

38714 **FUTURE DIRECTIONS**

38715 None.

38716 **SEE ALSO**

38717 `endgrent()`, `getgrnam()`, `sysconf()`

38718 XBD `<grp.h>`, `<sys/types.h>`

38719 **CHANGE HISTORY**

38720 First released in Issue 1. Derived from System V Release 2.0.

38721 **Issue 5**

38722 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 38723 VALUE section.

38724 The `getgrgid_r()` function is included for alignment with the POSIX Threads Extension.

38725 A note indicating that the `getgrgid()` function need not be reentrant is added to the  
 38726 DESCRIPTION.

38727 **Issue 6**

38728 The `getgrgid_r()` function is marked as part of the Thread-Safe Functions option.

38729 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 38730 describing matching the `gid`.

38731 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

38732 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

38733 The following new requirements on POSIX implementations derive from alignment with the  
 38734 Single UNIX Specification:

38735 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 38736 required for conforming implementations of previous POSIX specifications, it was not  
 38737 required for UNIX applications.

38738 • In the RETURN VALUE section, the requirement to set `errno` on error is added.

38739 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

38740 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 38741 its avoidance of possibly using a static data area.

38742 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
 38743 buffer from `bufsize` characters to bytes.

38744 **Issue 7**

38745 Austin Group Interpretation 1003.1-2001 #156 is applied.

38746 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

38747 SD5-XSH-ERN-166 is applied.

- 38748 The `getgrgid_r()` function is moved from the Thread-Safe Functions option to the Base.
- 38749 A minor addition is made to the EXAMPLES section, reminding the application developer to  
38750 free memory allocated as if by `malloc()`.
- 38751 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0241 [75] is applied.
- 38752 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0160 [808], XSH/TC2-2008/0161 [808],  
38753 XSH/TC2-2008/0162 [656], and XSH/TC2-2008/0163 [808] are applied.
- 38754 **Issue 8**
- 38755 Austin Group Defect 398 is applied, changing the [ERANGE] error from “may fail” to “shall  
38756 fail”.
- 38757 Austin Group Defect 1570 is applied, removing extra spacing in “==”.

38758 **NAME**

38759 getgrnam, getgrnam\_r — search group database for a name

38760 **SYNOPSIS**

```
38761 #include <grp.h>
38762 struct group *getgrnam(const char *name);
38763 int getgrnam_r(const char *name, struct group *grp, char *buffer,
38764 size_t bufsize, struct group **result);
```

38765 **DESCRIPTION**38766 The *getgrnam()* function shall search the group database for an entry with a matching *name*.38767 The *getgrnam()* function need not be thread-safe.38768 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam()*.  
38769 If *getgrnam()* returns a null pointer and *errno* is set to non-zero, an error occurred.

38770 The *getgrnam\_r()* function shall update the **group** structure pointed to by *grp* and store a pointer  
38771 to that structure at the location pointed to by *result*. The structure shall contain an entry from  
38772 the group database with a matching *name*. Storage referenced by the **group** structure is allocated  
38773 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to  
38774 *sysconf(\_SC\_GETGR\_R\_SIZE\_MAX)* returns either  $-1$  without changing *errno* or an initial value  
38775 suggested for the size of this buffer. A null pointer is returned at the location pointed to by *result*  
38776 on error or if the requested entry is not found.

38777 **RETURN VALUE**

38778 The *getgrnam()* function shall return a pointer to a **struct group** with the structure defined in  
38779 **<grp.h>** with a matching entry if one is found. The *getgrnam\_r()* function shall return a null  
38780 pointer if either the requested entry was not found, or an error occurred. If the requested entry  
38781 was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

38782 The application shall not modify the structure to which the return value points, nor any storage  
38783 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
38784 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
38785 subsequent call to *getgrent()*, *getgrgid()*, or *getgrnam()*. The returned pointer, and pointers  
38786 within the structure, might also be invalidated if the calling thread is terminated.

38787 The *getgrnam\_r()* function shall return zero on success or if the requested entry was not found  
38788 and no error has occurred. If any error has occurred, an error number shall be returned to  
38789 indicate the error.

38790 **ERRORS**38791 The *getgrnam()* and *getgrnam\_r()* functions may fail if:

- |       |          |  |
|-------|----------|--|
| 38792 | [EIO]    | An I/O error has occurred.   |
| 38793 | [EINTR]  | A signal was caught during <i>getgrnam()</i> .                         |
| 38794 | [EMFILE] | All file descriptors available to the process are currently open.      |
| 38795 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

38796 The *getgrnam\_r()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 38797 | [ERANGE] | Insufficient storage was supplied via <i>buffer</i> and <i>bufsize</i> to contain the data to be<br>38798 referenced by the resulting <b>group</b> structure. |
|-------|----------|---|

**EXAMPLES**

38799 Note that `sysconf(_SC_GETGR_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size  
 38800 of the buffer needed to store all the groups returned. This example shows how an application  
 38801 can allocate a buffer of sufficient size to work with `getgrnam_r()`.  
 38802

```

38803 long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
38804 size_t len;
38805 if (initlen == -1)
38806     /* Default initial length. */
38807     len = 1024;
38808 else
38809     len = (size_t) initlen;
38810 struct group result;
38811 struct group *resultp;
38812 char *buffer = malloc(len);
38813 if (buffer == NULL)
38814     ...handle error...
38815 int e;
38816 while ((e = getgrnam_r("somegroup", &result, buffer, len, &resultp))
38817     == ERANGE)
38818     {
38819         size_t newlen = 2 * len;
38820         if (newlen < len)
38821             ...handle error...
38822         len = newlen;
38823         char *newbuffer = realloc(buffer, len);
38824         if (newbuffer == NULL)
38825             ...handle error...
38826         buffer = newbuffer;
38827     }
38828 if (e != 0)
38829     ...handle error...
38830 free (buffer);

```

**APPLICATION USAGE**

38831 The `getgrnam_r()` function is thread-safe and shall return values in a user-supplied buffer instead  
 38832 of possibly using a static data area that may be overwritten by each call.  
 38833

38834 Portable applications should take into account that it is usual for an implementation to return `-1`  
 38835 from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

**RATIONALE**

38836 None.  
 38837

**FUTURE DIRECTIONS**

38838 None.  
 38839

**SEE ALSO**

38841 [endgrent\(\)](#), [getgrgid\(\)](#), [sysconf\(\)](#)

38842 XBD [<grp.h>](#), [<sys/types.h>](#)

38843 **CHANGE HISTORY**

38844 First released in Issue 1. Derived from System V Release 2.0.

38845 **Issue 5**38846 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
38847 VALUE section.38848 The `getgrnam_r()` function is included for alignment with the POSIX Threads Extension.38849 A note indicating that the `getgrnam()` function need not be reentrant is added to the  
38850 DESCRIPTION.38851 **Issue 6**38852 The `getgrnam_r()` function is marked as part of the Thread-Safe Functions option.

38853 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

38854 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.38855 The following new requirements on POSIX implementations derive from alignment with the  
38856 Single UNIX Specification:38857 

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
38858 required for conforming implementations of previous POSIX specifications, it was not  
38859 required for UNIX applications.

38860 

- In the RETURN VALUE section, the requirement to set `errno` on error is added.

38861 

- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

38862 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
38863 its avoidance of possibly using a static data area.38864 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
38865 buffer from `bufsize` characters to bytes.38866 **Issue 7**

38867 Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section.

38868 Austin Group Interpretation 1003.1-2001 #156 is applied.

38869 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

38870 SD5-XSH-ERN-166 is applied.

38871 The `getgrnam_r()` function is moved from the Thread-Safe Functions option to the Base.38872 A minor addition is made to the EXAMPLES section, reminding the application developer to  
38873 free memory allocated as if by `malloc()`.

38874 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0242 [75] is applied.

38875 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0164 [808], XSH/TC2-2008/0165 [808],  
38876 XSH/TC2-2008/0166 [656], and XSH/TC2-2008/0167 [808] are applied.38877 **Issue 8**38878 Austin Group Defect 398 is applied, changing the [ERANGE] error from “may fail” to “shall  
38879 fail”.

38880 Austin Group Defect 1570 is applied, removing extra spacing in “==”.

38881 **NAME**

38882 getgroups — get supplementary group IDs

38883 **SYNOPSIS**

38884 #include &lt;unistd.h&gt;

38885 int getgroups(int *gidsetsize*, gid\_t *grouplist*[]);38886 **DESCRIPTION**

38887 The *getgroups()* function shall fill in the array *grouplist* with the current supplementary group  
 38888 IDs of the calling process. It is implementation-defined whether *getgroups()* also returns the  
 38889 effective group ID in the *grouplist* array.

38890 The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual  
 38891 number of group IDs stored in the array shall be returned. The values of array entries with  
 38892 indices greater than or equal to the value returned are undefined.

38893 If *gidsetsize* is 0, *getgroups()* shall return the number of group IDs that it would otherwise return  
 38894 without modifying the array pointed to by *grouplist*.

38895 If the effective group ID of the process is returned with the supplementary group IDs, the value  
 38896 returned shall always be greater than or equal to one and less than or equal to the value of  
 38897 {NGROUPS\_MAX}+1.

38898 **RETURN VALUE**

38899 Upon successful completion, the number of supplementary group IDs shall be returned. A  
 38900 return value of -1 indicates failure and *errno* shall be set to indicate the error.

38901 **ERRORS**38902 The *getgroups()* function shall fail if:

38903 [EINVAL] The *gidsetsize* argument is non-zero and less than the number of group IDs  
 38904 that would have been returned.

38905 **EXAMPLES**38906 **Getting the Supplementary Group IDs of the Calling Process**

38907 The following example places the current supplementary group IDs of the calling process into  
 38908 the *group* array.

```
38909 #include <sys/types.h>
38910 #include <unistd.h>
38911 ...
38912 gid_t *group;
38913 int ngroups;
38914 long ngroups_max;

38915 ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
38916 group = (gid_t *)malloc(ngroups_max *sizeof(gid_t));

38917 ngroups = getgroups(ngroups_max, group);
```

38918 **APPLICATION USAGE**

38919 None.

38920 **RATIONALE**

38921 The related function *setgroups()* is a privileged operation and therefore is not covered by this  
 38922 volume of POSIX.1-2024.

38923 As implied by the definition of supplementary groups, the effective group ID may appear in the

38924 array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but  
38925 the application needs to call *getegid()* to be sure of getting all of the information. Various  
38926 implementation variations and administrative sequences cause the set of groups appearing in  
38927 the result of *getgroups()* to vary in order and as to whether the effective group ID is included,  
38928 even when the set of groups is the same (in the mathematical sense of “set”). (The history of a  
38929 process and its parents could affect the details of the result.)

38930 Application developers should note that {NGROUPS\_MAX} is not necessarily a constant on all  
38931 implementations.

#### 38932 FUTURE DIRECTIONS

38933 None.

#### 38934 SEE ALSO

38935 *getegid()*, *setgid()*

38936 XBD <sys/types.h>, <unistd.h>

#### 38937 CHANGE HISTORY

38938 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 38939 Issue 5

38940 Normative text previously in the APPLICATION USAGE section is moved to the  
38941 DESCRIPTION.

#### 38942 Issue 6

38943 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

38944 The following new requirements on POSIX implementations derive from alignment with the  
38945 Single UNIX Specification:

38946 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
38947 required for conforming implementations of previous POSIX specifications, it was not  
38948 required for UNIX applications.

38949 • A return value of 0 is not permitted, because {NGROUPS\_MAX} cannot be 0. This is a FIPS  
38950 requirement.

38951 The following changes were made to align with the IEEE P1003.1a draft standard:

38952 • An explanation is added that the effective group ID may be included in the supplementary  
38953 group list.

#### 38954 Issue 8

38955 Austin Group Defect 1400 is applied, changing the EXAMPLES section.



38956 **NAME**  
38957       gethostent — network host database functions

38958 **SYNOPSIS**  
38959       #include <netdb.h>  
38960       struct hostent \*gethostent(void);

38961 **DESCRIPTION**  
38962       Refer to *endhostent()*.

38963 **NAME**  
38964       gethostid — get an identifier for the current host

38965 **SYNOPSIS**

```
38966 XSI       #include <unistd.h>  
38967       long gethostid(void);
```

38968 **DESCRIPTION**

38969       The *gethostid()* function shall retrieve a 32-bit identifier for the current host.

38970 **RETURN VALUE**

38971       Upon successful completion, *gethostid()* shall return an identifier for the current host.

38972 **ERRORS**

38973       No errors are defined.

38974 **EXAMPLES**

38975       None.

38976 **APPLICATION USAGE**

38977       This volume of POSIX.1-2024 does not define the domain in which the return value is unique.

38978 **RATIONALE**

38979       None.

38980 **FUTURE DIRECTIONS**

38981       None.

38982 **SEE ALSO**

38983       *initstate()*

38984       XBD <[unistd.h](#)>

38985 **CHANGE HISTORY**

38986       First released in Issue 4, Version 2.

38987 **Issue 5**

38988       Moved from X/OPEN UNIX extension to BASE.

**38989 NAME**

38990           gethostname — get name of current host

**38991 SYNOPSIS**

38992           #include <unistd.h>

38993           int gethostname(char \*name, size\_t namelen);

**38994 DESCRIPTION**

38995           The *gethostname()* function shall return the standard host name for the current machine. The  
38996           *namelen* argument shall specify the size of the array pointed to by the *name* argument. The  
38997           returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold  
38998           the host name, then the returned name shall be truncated and it is unspecified whether the  
38999           returned name is null-terminated.

39000           Host names are limited to {HOST\_NAME\_MAX} bytes.

**39001 RETURN VALUE**

39002           Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

**39003 ERRORS**

39004           No errors are defined.

**39005 EXAMPLES**

39006           None.

**39007 APPLICATION USAGE**

39008           None.

**39009 RATIONALE**

39010           None.

**39011 FUTURE DIRECTIONS**

39012           None.

**39013 SEE ALSO**

39014           [\*gethostid\(\)\*](#), [\*uname\(\)\*](#)

39015           XBD [\*\*<unistd.h>\*\*](#)

**39016 CHANGE HISTORY**

39017           First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39018           The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from  
39019           **socklen\_t** to **size\_t**.

39020 **NAME**

39021            getline — read a delimited record from *stream*

39022 **SYNOPSIS**

```
39023 CX        #include <stdio.h>
39024            ssize_t getline(char **restrict lineptr, size_t *restrict n,
39025            FILE *restrict stream);
```

39026 **DESCRIPTION**

39027            Refer to *getdelim()*.

39028 **NAME**

39029 getlocalename\_l — get a locale name from a locale object

39030 **SYNOPSIS**

```
39031 CX #include <locale.h>
39032 const char *getlocalename_l(int category, locale_t locobj);
```

39033 **DESCRIPTION**

39034 If *category* is not LC\_ALL, the *getlocalename\_l()* function shall return the locale name for the  
 39035 given locale category of the locale object *locobj*, or of the global locale if *locobj* is the special locale  
 39036 object LC\_GLOBAL\_LOCALE.

39037 If *category* is LC\_ALL, the *getlocalename\_l()* function shall return a string that encodes the locale  
 39038 settings for all locale categories of the locale object *locobj*, or of the global locale if *locobj* is the  
 39039 special locale object LC\_GLOBAL\_LOCALE, in the same form as is returned by *setlocale()*. The  
 39040 string returned is such that a subsequent call to *setlocale()*, from the same process, with a pointer  
 39041 to that string as *locale* and the LC\_ALL *category* shall set the global locale to the same locale for  
 39042 each category as was present in the queried object.

39043 If the value of the *category* argument is neither LC\_ALL nor a supported locale category value  
 39044 (see *setlocale()*), *getlocalename\_l()* shall fail.

39045 The behavior is undefined if the *locobj* argument is neither the special locale object  
 39046 LC\_GLOBAL\_LOCALE nor a valid locale object handle.

39047 **RETURN VALUE**

39048 Upon successful completion, *getlocalename\_l()* shall return a pointer to a string; otherwise, a null  
 39049 pointer shall be returned.

39050 If *locobj* is LC\_GLOBAL\_LOCALE, the returned string pointer might be invalidated or the string  
 39051 content might be overwritten by a subsequent call in the same thread to *getlocalename\_l()* with  
 39052 LC\_GLOBAL\_LOCALE; the returned string pointer might also be invalidated if the calling  
 39053 thread is terminated. Otherwise, the returned string pointer and content shall remain valid until  
 39054 the locale object *locobj* is used in a call to *freelocale()* or as the *base* argument in a successful call to  
 39055 *newlocale()*.

39056 **ERRORS**

39057 No errors are defined.

39058 **EXAMPLES**39059 **Determining the locale name for a category of the current locale**

39060 The following example shows how to obtain the locale name for the LC\_NUMERIC category of  
 39061 the current thread-local locale, or of the global locale if no thread-local locale is in use.

```
39062 #include <locale.h>
39063 ...
39064 const char *name;
39065 locale_t loc = uselocale(NULL);
39066 name = getlocalename_l(LC_NUMERIC, loc);
```

39067 **APPLICATION USAGE**

39068 In addition to the caveats regarding validity of the returned string pointer in RETURN VALUE,  
 39069 the content of the string returned when *category* is LC\_ALL is only required to be valid for the  
 39070 life of the process, so is not intended for storage or sharing between processes. As the internal  
 39071 format of the string is implementation-specific, there is nothing preventing a subsequent run of

39072 an application from being presented a different format, for example if the implementation is  
39073 updated.

39074 **RATIONALE**

39075 Historical versions of *getlocalename\_l()* did not handle the special locale object  
39076 LC\_GLOBAL\_LOCALE, requiring that applications used *setlocale(category, NULL)* to query the  
39077 global locale if *uselocale(NULL)* returned LC\_GLOBAL\_LOCALE. However, since *setlocale()* is  
39078 not required to be thread-safe (even when the only concurrent calls are ones that query the  
39079 locale), this method was problematic for multi-threaded processes. This standard requires that  
39080 *getlocalename\_l(category, LC\_GLOBAL\_LOCALE)* queries the global locale in a thread-safe  
39081 manner, for example by returning a pointer to a thread-local internal buffer instead of a process-  
39082 wide internal buffer.

39083 **FUTURE DIRECTIONS**

39084 None.

39085 **SEE ALSO**

39086 *freelocale()*, *newlocale()*, *setlocale()*, *uselocale()*

39087 XBD Chapter 7 (on page 127), `<locale.h>`

39088 **CHANGE HISTORY**

39089 First released in Issue 8.

39090 **NAME**

39091 getlogin, getlogin\_r — get login name

39092 **SYNOPSIS**

39093 #include &lt;unistd.h&gt;

39094 char \*getlogin(void);

39095 int getlogin\_r(char \*name, size\_t namesize);

39096 **DESCRIPTION**

39097 The *getlogin()* function shall return a pointer to a string containing the user name associated by  
 39098 the login activity with the controlling terminal of the current process. If *getlogin()* returns a non-  
 39099 null pointer, then that pointer points to the name that the user logged in under, even if there are  
 39100 several login names with the same user ID.

39101 The *getlogin()* function need not be thread-safe.

39102 The *getlogin\_r()* function shall put the name associated by the login activity with the controlling  
 39103 terminal of the current process in the character array pointed to by *name*. The array is *namesize*  
 39104 characters long and should have space for the name and the terminating null character. The  
 39105 maximum size of the login name is {LOGIN\_NAME\_MAX}.

39106 If *getlogin\_r()* is successful, *name* points to the name the user used at login, even if there are  
 39107 several login names with the same user ID.

39108 The *getlogin()* and *getlogin\_r()* functions may make use of file descriptors 0, 1, and 2 to find the  
 39109 controlling terminal of the current process, examining each in turn until the terminal is found. If  
 39110 in this case none of these three file descriptors is open to the controlling terminal, these functions  
 39111 may fail. The method used to find the terminal associated with a file descriptor may depend on  
 39112 the file descriptor being open to the actual terminal device, not */dev/tty*.

39113 **RETURN VALUE**

39114 Upon successful completion, *getlogin()* shall return a pointer to the login name or a null pointer  
 39115 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to  
 39116 indicate the error.

39117 The application shall not modify the string returned. The returned pointer might be invalidated  
 39118 or the string content might be overwritten by a subsequent call to *getlogin()*. The returned  
 39119 pointer and the string content might also be invalidated if the calling thread is terminated.

39120 If successful, the *getlogin\_r()* function shall return zero; otherwise, an error number shall be  
 39121 returned to indicate the error.

39122 **ERRORS**

39123 These functions may fail if:

39124 [EMFILE] All file descriptors available to the process are currently open.

39125 [ENFILE] The maximum allowable number of files is currently open in the system.

39126 [ENOTTY] None of the file descriptors 0, 1, or 2 is open to the controlling terminal of the  
 39127 current process.

39128 [ENXIO] The calling process has no controlling terminal.

39129 The *getlogin\_r()* function shall fail if:

39130 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
 39131 including the terminating null character.

39132 **EXAMPLES**39133 **Getting the User Login Name**

39134 The following example calls the *getlogin()* function to obtain the name of the user associated  
 39135 with the calling process, and passes this information to the *getpwnam()* function to get the  
 39136 associated user database information.

```

39137 #include <unistd.h>
39138 #include <sys/types.h>
39139 #include <pwd.h>
39140 #include <stdio.h>
39141 ...
39142 char *lgn;
39143 struct passwd *pw;
39144 ...
39145 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
39146     fprintf(stderr, "Get of user information failed.\n"); exit(1);
39147 }

```

39148 **APPLICATION USAGE**

39149 Three names associated with the current process can be determined: *getpwuid(geteuid())* shall  
 39150 return the name associated with the effective user ID of the process; *getlogin()* shall return the  
 39151 name associated with the current login activity; and *getpwuid(getuid())* shall return the name  
 39152 associated with the real user ID of the process.

39153 The *getlogin\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 39154 possibly using a static data area that may be overwritten by each call.

39155 **RATIONALE**

39156 The *getlogin()* function returns a pointer to the user's login name. The same user ID may be  
 39157 shared by several login names. If it is desired to get the user database entry that is used during  
 39158 login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()*  
 39159 function. (This might be used to determine the user's login shell, particularly where a single user  
 39160 has multiple login shells with distinct login names, but the same user ID.)

39161 The information provided by the *cuserid()* function, which was originally defined in the  
 39162 POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
39163 getpwuid(geteuid())
```

39164 while the information provided by historical implementations of *cuserid()* can be obtained by:

```
39165 getpwuid(getuid())
```

39166 The thread-safe version of this function places the user name in a user-supplied buffer and  
 39167 returns a non-zero value if it fails. The non-thread-safe version may return the name in a static  
 39168 data area that may be overwritten by each call.

39169 **FUTURE DIRECTIONS**

39170 None.

39171 **SEE ALSO**

39172 *getpwnam()*, *getpwuid()*, *geteuid()*, *getuid()*

39173 XBD [<limits.h>](#), [<unistd.h>](#)



**39174 CHANGE HISTORY**

39175 First released in Issue 1. Derived from System V Release 2.0.

**39176 Issue 5**

39177 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
39178 VALUE section.

39179 The *getlogin\_r()* function is included for alignment with the POSIX Threads Extension.

39180 A note indicating that the *getlogin()* function need not be reentrant is added to the  
39181 DESCRIPTION.

**39182 Issue 6**

39183 The *getlogin\_r()* function is marked as part of the Thread-Safe Functions option.

39184 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

39185 The following new requirements on POSIX implementations derive from alignment with the  
39186 Single UNIX Specification:

- 39187 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 39188 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

39189 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
39190 its avoidance of possibly using a static data area.

**39191 Issue 7**

39192 Austin Group Interpretation 1003.1-2001 #156 is applied.

39193 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

39194 The *getlogin\_r()* function is moved from the Thread-Safe Functions option to the Base.

39195 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0243 [172], XSH/TC1-2008/0244 [75],  
39196 and XSH/TC1-2008/0245 [172] are applied.

39197 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0168 [656] is applied.

**39198 Issue 8**

39199 Austin Group Defect 398 is applied, changing the [ERANGE] error from “may fail” to “shall  
39200 fail”.

39201 **NAME**

39202 getnameinfo — get name information

39203 **SYNOPSIS**

39204 #include &lt;sys/socket.h&gt;

39205 #include &lt;netdb.h&gt;

```
39206 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
39207 char *restrict node, socklen_t nodelen, char *restrict service,
39208 socklen_t servicelen, int flags);
```

39209 **DESCRIPTION**

39210 The *getnameinfo()* function shall translate a socket address to a node name and service location,  
 39211 all of which are defined as in *freeaddrinfo()*.

39212 The *sa* argument points to a socket address structure to be translated. The *salen* argument  
 39213 contains the length of the address pointed to by *sa*.

39214 IP6 If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible  
 39215 IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node  
 39216 name for that IPv4 address.

39217 If the address is the IPv6 unspecified address ("::"), a lookup shall not be performed and the  
 39218 behavior shall be the same as when the node's name cannot be located.

39219 If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument  
 39220 points to a buffer able to contain up to *nodelen* bytes that receives the node name as a null-  
 39221 terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name  
 39222 shall not be returned. If the node's name cannot be located, the numeric form of the address  
 39223 contained in the socket address structure pointed to by the *sa* argument is returned instead of its  
 39224 name.

39225 If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service*  
 39226 argument points to a buffer able to contain up to *servicelen* bytes that receives the service name  
 39227 as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero,  
 39228 the service name shall not be returned. If the service's name cannot be located, the numeric form  
 39229 of the service address (for example, its port number) shall be returned instead of its name.

39230 The *flags* argument is a flag that changes the default actions of the function. By default the fully-  
 39231 qualified domain name (FQDN) for the host shall be returned, but:

- 39232 • If the flag bit NI\_NOFQDN is set, only the node name portion of the FQDN shall be  
 39233 returned for local hosts.
- 39234 • If the flag bit NI\_NUMERICHOST is set, the numeric form of the address contained in the  
 39235 socket address structure pointed to by the *sa* argument shall be returned instead of its  
 39236 name.
- 39237 • If the flag bit NI\_NAMEREQD is set, an error shall be returned if the host's name cannot  
 39238 be located.
- 39239 • If the flag bit NI\_NUMERICSERV is set, the numeric form of the service address shall be  
 39240 returned (for example, its port number) instead of its name.
- 39241 • If the flag bit NI\_NUMERICSCOPE is set, the numeric form of the scope identifier shall be  
 39242 returned (for example, interface index) instead of its name. This flag shall be ignored if the  
 39243 *sa* argument is not an IPv6 address.

- 39244           • If the flag bit NI\_DGRAM is set, this indicates that the service is a datagram service  
 39245           (SOCK\_DGRAM). The default behavior shall assume that the service is a stream service  
 39246           (SOCK\_STREAM).

39247           **Notes:**

- 39248                     1. The two NI\_NUMERICxxx flags are required to support the `-n` flag that many  
 39249                     commands provide.
- 39250                     2. The NI\_DGRAM flag is required for the few AF\_INET and AF\_INET6 port numbers (for  
 39251                     example, [512,514]) that represent different services for UDP and TCP.

39252           The `getnameinfo()` function shall be thread-safe.

39253           **RETURN VALUE**

39254           A zero return value for `getnameinfo()` indicates successful completion; a non-zero return value  
 39255           indicates failure. The possible values for the failures are listed in the ERRORS section.

39256           Upon successful completion, `getnameinfo()` shall return the *node* and *service* names, if requested,  
 39257           in the buffers provided. The returned names are always null-terminated strings.

39258           **ERRORS**

39259           The `getnameinfo()` function shall fail and return the corresponding value if:

39260           [EAI\_AGAIN]   The name could not be resolved at this time. Future attempts may succeed.

39261           [EAI\_BADFLAGS]  
 39262                     The *flags* had an invalid value.

39263           [EAI\_FAIL]     A non-recoverable error occurred.

39264           [EAI\_FAMILY]   The address family was not recognized or the address length was invalid for  
 39265           the specified family.

39266           [EAI\_MEMORY]   There was a memory allocation failure.

39267           [EAI\_NONAME]   The name does not resolve for the supplied parameters.

39268                     NI\_NAMEREQD is set and the host's name cannot be located, or both  
 39269                     *nodename* and *servname* were null.

39270           [EAI\_OVERFLOW]  
 39271                     An argument buffer overflowed. The buffer pointed to by the *node* argument  
 39272                     or the *service* argument was too small.

39273           [EAI\_SYSTEM]   A system error occurred. The error code can be found in *errno*.

39274           **EXAMPLES**

39275           None.

39276           **APPLICATION USAGE**

39277           If the returned values are to be used as part of any further name resolution (for example, passed  
 39278           to `getaddrinfo()`), applications should provide buffers large enough to store any result possible on  
 39279           the system.

39280           Given the IPv4-mapped IPv6 address "`::ffff:1.2.3.4`", the implementation performs a  
 39281           lookup as if the socket address structure contains the IPv4 address "`1.2.3.4`".

39282           The IPv6 unspecified address ("`:::`") and the IPv6 loopback address ("`:::1`") are not  
 39283           IPv4-compatible addresses.

39284 **RATIONALE**

39285 None.

39286 **FUTURE DIRECTIONS**

39287 None.

39288 **SEE ALSO**39289 *endservent()*, *freeaddrinfo()*, *gai\_strerror()*, *inet\_ntop()*, *socket()*

39290 XBD &lt;netdb.h&gt;, &lt;sys/socket.h&gt;

39291 **CHANGE HISTORY**

39292 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39293 The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the  
39294 ISO/IEC 9899:1999 standard.39295 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in  
39296 the SYNOPSIS and DESCRIPTION for alignment with IPv6.39297 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the  
39298 [EAI\_OVERFLOW] error to the ERRORS section.39299 **Issue 7**39300 SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified  
39301 address.39302 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0246 [284] and XSH/TC1-2008/0247  
39303 [285] are applied.

39304 **NAME**

39305           getnetbyaddr, getnetbyname, getnetent — network database functions

39306 **SYNOPSIS**

39307           #include &lt;netdb.h&gt;

39308           struct netent \*getnetbyaddr(uint32\_t net, int type);

39309           struct netent \*getnetbyname(const char \*name);

39310           struct netent \*getnetent(void);

39311 **DESCRIPTION**39312           Refer to *endnetent()*.

39313 **NAME**

39314            getopt, optarg, opterr, optind, optopt — command option parsing

39315 **SYNOPSIS**

39316            #include &lt;unistd.h&gt;

39317            int getopt(int argc, char \* const argv[], const char \*optstring);

39318            extern char \*optarg;

39319            extern int opterr, optind, optopt;

39320 **DESCRIPTION**39321            The *getopt()* function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,  
39322            6, 7, 9, and 10 in XBD [Section 12.2](#) (on page 215).39323            The parameters *argc* and *argv* are the argument count and argument array as passed to *main()*  
39324            (see *exec()*). The argument *optstring* is a string of recognized option characters; if a character is  
39325            followed by a <colon>, the option takes an argument. All option characters allowed by Utility  
39326            Syntax Guideline 3 are allowed in *optstring*. The *optstring* argument can optionally start with a  
39327            <plus-sign> ('+'), which shall have no effect on behavior in a conforming environment. If a  
39328            <plus-sign> occurs anywhere besides the first character of *optstring*, the behavior is unspecified.  
39329            The implementation may accept other characters as an extension.39330            The variable *optind* is the index of the next element of the *argv[]* vector to be processed. It shall  
39331            be initialized to 1 by the system, and *getopt()* shall update it when it finishes with each element  
39332            of *argv[]*. If the application sets *optind* to zero before calling *getopt()*, the behavior is unspecified.  
39333            When an element of *argv[]* contains multiple option characters, it is unspecified how *getopt()*  
39334            determines which options have already been processed.39335            The *getopt()* function shall return the next option character (if one is found) from *argv* that  
39336            matches a character in *optstring* (excluding an optional leading <plus-sign>), if there is one that  
39337            matches. If the option takes an argument, *getopt()* shall set the variable *optarg* to point to the  
39338            option-argument as follows:

- 39339            1. If the option was the last character in the string pointed to by an element of
- argv*
- , then
- 
- 39340
- optarg*
- shall contain the next element of
- argv*
- , and
- optind*
- shall be incremented by 2. If the
- 
- 39341            resulting value of
- optind*
- is greater than
- argc*
- , this indicates a missing option-argument,
- 
- 39342            and
- getopt()*
- shall return an error indication.
- 
- 39343            2. Otherwise,
- optarg*
- shall point to the string following the option character in that element
- 
- 39344            of
- argv*
- , and
- optind*
- shall be incremented by 1.

39345            If, when *getopt()* is called, any of the following is true:39346            *argv[optind]* is a null pointer  
39347            \**argv[optind]* is not the character '-'  
39348            *argv[optind]* points to the string "--"39349            *getopt()* shall return -1 without changing *optind*. If:39350            *argv[optind]* points to the string "--"39351            *getopt()* shall return -1 after incrementing *optind*.39352            If *getopt()* encounters a <colon> as an option character, or an option character that is not  
39353            contained in *optstring* after an optional leading <plus-sign>, it shall return the <question-mark>  
39354            ('?') character. If it detects a missing option-argument, it shall return the <colon> character  
39355            (':') if the first character of *optstring* after an optional <plus-sign> was a <colon>, or a  
39356            <question-mark> character('?') otherwise. In either case, *getopt()* shall set the variable *optopt* to  
39357            the option character that caused the error. If the application has not set the variable *opterr* to 0,

39358 and the first character of *optstring* after an optional <plus-sign> is not a <colon>, *getopt()* shall  
 39359 also print a diagnostic message to *stderr* in the format specified for the *getopts* utility, unless the  
 39360 *stderr* stream has wide orientation, in which case the behavior is undefined.

39361 The *getopt()* function need not be thread-safe.

### 39362 RETURN VALUE

39363 The *getopt()* function shall return the next option character specified on the command line.

39364 A <colon> (':') shall be returned if *getopt()* detects a missing argument and the first character  
 39365 of *optstring* after an optional <plus-sign> was a <colon> (':').

39366 A <question-mark> ('?') shall be returned if *getopt()* encounters a <colon> as an option  
 39367 character, encounters an option character not in *optstring*, or detects a missing argument and the  
 39368 first character of *optstring* after an optional <plus-sign> was not a <colon> (':').

39369 Otherwise, *getopt()* shall return -1 when all command line options are parsed.

### 39370 ERRORS

39371 If the application has not set the variable *opterr* to 0, the first character of *optstring* is not a  
 39372 <colon>, and a write error occurs while *getopt()* is printing a diagnostic message to *stderr*, then  
 39373 the error indicator for *stderr* shall be set; but *getopt()* shall still succeed and the value of *errno*  
 39374 after *getopt()* is unspecified.

### 39375 EXAMPLES

#### 39376 Parsing Command Line Options

39377 The following code fragment shows how you might process the arguments for a utility that can  
 39378 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require  
 39379 arguments:

```

39380 #include <stdio.h>
39381 #include <stdlib.h>
39382 #include <unistd.h>

39383 int
39384 main(int argc, char *argv[ ])
39385 {
39386     int c;
39387     int bflg = 0, aflg = 0, errflg = 0;
39388     char *ifile;
39389     char *ofile;
39390     . . .
39391     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
39392         switch(c) {
39393             case 'a':
39394                 if (bflg)
39395                     errflg++;
39396                 else
39397                     aflg++;
39398                 break;
39399             case 'b':
39400                 if (aflg)
39401                     errflg++;
39402                 else
39403                     bflg++;

```

```

39404         break;
39405     case 'f':
39406         ifile = optarg;
39407         break;
39408     case 'o':
39409         ofile = optarg;
39410         break;
39411     case ':': /* -f or -o without operand */
39412         fprintf(stderr,
39413             "Option -%c requires an operand\n", optopt);
39414         errflg++;
39415         break;
39416     case '?':
39417         fprintf(stderr,
39418             "Unrecognized option: '-%c'\n", optopt);
39419         errflg++;
39420     }
39421 }
39422 if (errflg) {
39423     fprintf(stderr, "usage: . . . ");
39424     exit(2);
39425 }
39426 for ( ; optind < argc; optind++) {
39427     if (access(argv[optind], R_OK)) {
39428         . . .
39429     }
39430 }
39431 }

```

39432 This code accepts any of the following as equivalent:

```

39433 cmd -ao arg path path
39434 cmd -a -o arg path path
39435 cmd -o arg -a path path
39436 cmd -a -o arg -- path path
39437 cmd -a -oarg path path
39438 cmd -aoarg path path

```

### 39439 **Selecting Options from the Command Line**

39440 The following example selects the type of database routines the user wants to use based on the  
39441 *Options* argument.

```

39442 #include <unistd.h>
39443 #include <string.h>
39444 ...
39445 const char *Options = "hdbt1";
39446 ...
39447 int dbtype, c;
39448 char *st;
39449 ...
39450 dbtype = 0;
39451 while ((c = getopt(argc, argv, Options)) != -1) {
39452     if ((st = strchr(Options, c)) != NULL) {

```



```

39453         dbtype = st - Options;
39454         break;
39455     }
39456 }

```

### 39457 APPLICATION USAGE

39458 The `getopt()` function is only required to support option characters included in Utility Syntax  
 39459 Guideline 3. Many historical implementations of `getopt()` support other characters as options.  
 39460 This is an allowed extension, but applications that use extensions are not maximally portable.  
 39461 Note that support for multi-byte option characters is only possible when such characters can be  
 39462 represented as type **int**.

39463 Applications which use wide-character output functions with `stderr` should ensure that any calls  
 39464 to `getopt()` do not write to `stderr`, either by setting `opterr` to 0 or by ensuring the first character of  
 39465 `optstring` is always a `<colon>`.

39466 While `ferror(stderr)` may be used to detect failures to write a diagnostic to `stderr` when `getopt()`  
 39467 returns `'?'`, the value of `errno` is unspecified in such a condition. Applications desiring more  
 39468 control over handling write failures should set `opterr` to 0 and independently perform output to  
 39469 `stderr`, rather than relying on `getopt()` to do the output.

### 39470 RATIONALE

39471 The `optopt` variable represents historical practice and allows the application to obtain the identity  
 39472 of the invalid option.

39473 The description has been written to make it clear that `getopt()`, like the `getopts` utility, deals with  
 39474 option-arguments whether separated from the option by `<blank>` characters or not. Note that  
 39475 the requirements on `getopt()` and `getopts` are more stringent than the Utility Syntax Guidelines.

39476 The `getopt()` function shall return `-1`, rather than EOF, so that `<stdio.h>` is not required.

39477 The special significance of a `<colon>` as the first character of `optstring` makes `getopt()` consistent  
 39478 with the `getopts` utility. It allows an application to make a distinction between a missing  
 39479 argument and an incorrect option letter without having to examine the option letter. It is true  
 39480 that a missing argument can only be detected in one case, but that is a case that has to be  
 39481 considered.

39482 In some non-conforming environments, the use of a leading `<plus-sign>` in `optstring` forces  
 39483 `getopt()` to behave in a conforming way, when it would otherwise have non-conforming  
 39484 behavior. Its use has been standardized to allow applications to be written that can guarantee  
 39485 behavior consistent with this specification even in an otherwise non-conforming environment. If  
 39486 both `<plus-sign>` and `<colon>` are used at the beginning of `optstring`, the `<plus-sign>` must be  
 39487 first.

39488 Note that the use of a leading `<plus-sign>` in `optstring` is only standardized for `getopt()`. Use of a  
 39489 `<plus-sign>` is intentionally left unspecified for the `getopts` utility, where historical  
 39490 implementations did not require a leading `<plus-sign>` for conforming behavior, and because  
 39491 some historical `getopts` implementations used a leading `<plus-sign>` for a different extension.

### 39492 FUTURE DIRECTIONS

39493 None.

### 39494 SEE ALSO

39495 [exec](#)

39496 [XBD Section 12.2](#) (on page 215), [<unistd.h>](#)

39497 [XCU `getopts`](#)

39498 **CHANGE HISTORY**

39499 First released in Issue 1. Derived from Issue 1 of the SVID.

39500 **Issue 5**

39501 A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.

39502 **Issue 6**

39503 IEEE PASC Interpretation 1003.2 #150 is applied.

39504 Austin Group Interpretation 1003.1-2001 #156 is applied.

39505 **Issue 7**

39506 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0248 [318], XSH/TC1-2008/0249 [460],  
39507 XSH/TC1-2008/0250 [189], XSH/TC1-2008/0251 [189], XSH/TC1-2008/0252 [189], and  
39508 XSH/TC1-2008/0253 [460] are applied.

39509 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0169 [608] is applied.

39510 **Issue 8**

39511 Austin Group Defect 191 is applied, allowing a leading <plus-sign> in *optstring*.

39512 Austin Group Defect 1179 is applied, adding some missing '}' characters at the end of the  
39513 example code.

39514 Austin Group Defect 1523 is applied, clarifying the conditions under which *getopt()* returns -1  
39515 without changing *optind*.

39516 **NAME**

39517 getpeername — get the name of the peer socket

39518 **SYNOPSIS**

```
39519 #include <sys/socket.h>
39520 int getpeername(int socket, struct sockaddr *restrict address,
39521 socklen_t *restrict address_len);
```

39522 **DESCRIPTION**

39523 The *getpeername()* function shall retrieve the peer address of the specified socket, store this  
 39524 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this  
 39525 address in the object pointed to by the *address\_len* argument.

39526 The *address\_len* argument points to a **socklen\_t** object which on input specifies the length of the  
 39527 supplied **sockaddr** structure, and on output specifies the length of the peer address. If the actual  
 39528 length of the address is greater than the length of the supplied **sockaddr** structure, the stored  
 39529 address shall be truncated.

39530 If the protocol permits connections by unbound clients, and the peer is not bound, then the  
 39531 value stored in the object pointed to by *address* is unspecified.

39532 **RETURN VALUE**

39533 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 39534 indicate the error.

39535 **ERRORS**

39536 The *getpeername()* function shall fail if:

- 39537 [EBADF] The *socket* argument is not a valid file descriptor.
- 39538 [EINVAL] The socket has been shut down.
- 39539 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.
- 39540 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 39541 [EOPNOTSUPP] The operation is not supported for the socket protocol.

39542 The *getpeername()* function may fail if:

- 39543 [ENOBUFS] Insufficient resources were available in the system to complete the call.

39544 **EXAMPLES**

39545 None.

39546 **APPLICATION USAGE**

39547 For AF\_UNIX sockets, it is recommended that *address* points to a buffer of length greater than  
 39548 `sizeof(struct sockaddr_un)` which has been initialized with null bytes. That way, even if  
 39549 the implementation supports the use of all bytes of *sun\_path* without a terminating null byte, the  
 39550 larger buffer guarantees that the *sun\_path* member can then be passed to other interfaces that  
 39551 expect a null-terminated string. If no truncation occurred based on the input value of *address\_len*,  
 39552 it is unspecified whether the returned *address\_len* will be `sizeof(struct sockaddr_un)`, or  
 39553 merely a value at least as large as `offsetof(struct sockaddr_un, sun_path)` plus the  
 39554 number of non-null bytes stored in *sun\_path*.

39555 **RATIONALE**

39556 None.

39557 **FUTURE DIRECTIONS**

39558 None.

39559 **SEE ALSO**39560 *accept()*, *bind()*, *getsockname()*, *socket()*39561 XBD <[sys/socket.h](#)>39562 **CHANGE HISTORY**

39563 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

39564 The **restrict** keyword is added to the *getpeername()* prototype for alignment with the  
39565 ISO/IEC 9899:1999 standard.39566 **Issue 7**

39567 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0254 [464] is applied.

39568 **Issue 8**39569 Austin Group Defect 561 is applied, adding a paragraph about *sun\_path* to APPLICATION  
39570 USAGE.39571 Austin Group Defect 1565 is applied, changing the description of *address\_len*.

39572 **NAME**

39573 getpgid — get the process group ID for a process

39574 **SYNOPSIS**

39575 #include &lt;unistd.h&gt;

39576 pid\_t getpgid(pid\_t pid);

39577 **DESCRIPTION**39578 The *getpgid()* function shall return the process group ID of the process whose process ID is equal  
39579 to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.39580 **RETURN VALUE**39581 Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return  
39582 (**pid\_t**)-1 and set *errno* to indicate the error.39583 **ERRORS**39584 The *getpgid()* function shall fail if:39585 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the  
39586 calling process, and the implementation does not allow access to the process  
39587 group ID of that process from the calling process.39588 [ESRCH] There is no process with a process ID equal to *pid*.39589 The *getpgid()* function may fail if:39590 [EINVAL] The value of the *pid* argument is invalid.39591 **EXAMPLES**

39592 None.

39593 **APPLICATION USAGE**

39594 None.

39595 **RATIONALE**

39596 None.

39597 **FUTURE DIRECTIONS**

39598 None.

39599 **SEE ALSO**39600 *exec*, *fork()*, *getpgrp()*, *getpid()*, *getsid()*, *setpgid()*, *setsid()*39601 XBD <**unistd.h**>39602 **CHANGE HISTORY**

39603 First released in Issue 4, Version 2.

39604 **Issue 5**

39605 Moved from X/OPEN UNIX extension to BASE.

39606 **Issue 7**39607 The *getpgid()* function is moved from the XSI option to the Base.

39608 **NAME**

39609 getpgrp — get the process group ID of the calling process

39610 **SYNOPSIS**

39611 #include &lt;unistd.h&gt;

39612 pid\_t getpgrp(void);

39613 **DESCRIPTION**39614 The *getpgrp()* function shall return the process group ID of the calling process.39615 **RETURN VALUE**39616 The *getpgrp()* function shall always be successful and no return value is reserved to indicate an  
39617 error.39618 **ERRORS**

39619 No errors are defined.

39620 **EXAMPLES**

39621 None.

39622 **APPLICATION USAGE**

39623 None.

39624 **RATIONALE**39625 4.3 BSD provides a *getpgrp()* function that returns the process group ID for a specified process.  
39626 Although this function supports job control, all known job control shells always specify the  
39627 calling process with this function. Thus, the simpler System V *getpgrp()* suffices, and the added  
39628 complexity of the 4.3 BSD *getpgrp()* is provided by the *getpgid()* function.39629 **FUTURE DIRECTIONS**

39630 None.

39631 **SEE ALSO**39632 *exec*, *fork()*, *getpgid()*, *getpid()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*39633 XBD <[sys/types.h](#)>, <[unistd.h](#)>39634 **CHANGE HISTORY**

39635 First released in Issue 1. Derived from Issue 1 of the SVID.

39636 **Issue 6**39637 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.39638 The following new requirements on POSIX implementations derive from alignment with the  
39639 Single UNIX Specification:

- 39640
- The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was  
39641 required for conforming implementations of previous POSIX specifications, it was not  
39642 required for UNIX applications.

39643 **Issue 8**

39644 Austin Group Defect 1245 is applied, changing the RATIONALE section.

39645 **NAME**39646            **getpid** — get the process ID39647 **SYNOPSIS**

39648            #include &lt;unistd.h&gt;

39649            pid\_t getpid(void);

39650 **DESCRIPTION**39651            The *getpid()* function shall return the process ID of the calling process.39652 **RETURN VALUE**39653            The *getpid()* function shall always be successful and no return value is reserved to indicate an  
39654            error.39655 **ERRORS**

39656            No errors are defined.

39657 **EXAMPLES**

39658            None.

39659 **APPLICATION USAGE**

39660            None.

39661 **RATIONALE**

39662            None.

39663 **FUTURE DIRECTIONS**

39664            None.

39665 **SEE ALSO**39666            *exec, fork(), getpgrp(), getppid(), kill(), mkdtemp(), setpgid(), setsid()*

39667            XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

39668 **CHANGE HISTORY**

39669            First released in Issue 1. Derived from Issue 1 of the SVID.

39670 **Issue 6**

39671            In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

39672            The following new requirements on POSIX implementations derive from alignment with the  
39673            Single UNIX Specification:

- 39674
- 39675            • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
39676            required for conforming implementations of previous POSIX specifications, it was not  
            required for UNIX applications.

39677 **NAME**

39678           getppid — get the parent process ID

39679 **SYNOPSIS**

39680           #include &lt;unistd.h&gt;

39681           pid\_t getppid(void);

39682 **DESCRIPTION**39683           The *getppid()* function shall return the parent process ID of the calling process.39684 **RETURN VALUE**39685           The *getppid()* function shall always be successful and no return value is reserved to indicate an error.39687 **ERRORS**

39688           No errors are defined.

39689 **EXAMPLES**

39690           None.

39691 **APPLICATION USAGE**

39692           None.

39693 **RATIONALE**

39694           None.

39695 **FUTURE DIRECTIONS**

39696           None.

39697 **SEE ALSO**39698           *exec*, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*

39699           XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

39700 **CHANGE HISTORY**

39701           First released in Issue 1. Derived from Issue 1 of the SVID.

39702 **Issue 6**

39703           In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

39704           The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 39705
- 
- 39706
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- 39707
- 
- 39708



39709 **NAME**

39710 getpriority, setpriority — get and set the nice value

39711 **SYNOPSIS**

```
39712 XSI      #include <sys/resource.h>
39713         int getpriority(int which, id_t who);
39714         int setpriority(int which, id_t who, int value);
```

39715 **DESCRIPTION**

39716 The *getpriority()* function shall obtain the nice value of a process, process group, or user. The  
 39717 *setpriority()* function shall set the nice value of a process, process group, or user to  
 39718 *value*+{NZERO}.

39719 Target processes are specified by the values of the *which* and *who* arguments. The *which*  
 39720 argument may be one of the following values: PRIO\_PROCESS, PRIO\_PGRP, or PRIO\_USER,  
 39721 indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an  
 39722 effective user ID, respectively. A 0 value for the *who* argument specifies the current process,  
 39723 process group, or user.

39724 The nice value set with *setpriority()* shall be applied to the process. If the process is multi-  
 39725 threaded, the nice value shall affect all system scope threads in the process.

39726 If more than one process is specified, *getpriority()* shall return value {NZERO} less than the  
 39727 lowest nice value pertaining to any of the specified processes, and *setpriority()* shall set the nice  
 39728 values of all of the specified processes to *value*+{NZERO}.

39729 The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling.  
 39730 While the range of valid nice values is [0,{NZERO}\*2-1], implementations may enforce more  
 39731 restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value,  
 39732 *setpriority()* shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater  
 39733 than the system's highest supported nice value, *setpriority()* shall set the nice value to the  
 39734 highest supported value.

39735 Only a process with appropriate privileges can lower its nice value.

39736 PS|TPS Any processes or threads using SCHED\_FIFO or SCHED\_RR shall be unaffected by a call to  
 39737 *setpriority()*. This is not considered an error. A process which subsequently reverts to  
 39738 SCHED\_OTHER need not have its priority affected by such a *setpriority()* call.

39739 The effect of changing the nice value may vary depending on the process-scheduling algorithm  
 39740 in effect.

39741 Since *getpriority()* can return the value -1 upon successful completion, it is necessary to set *errno*  
 39742 to 0 prior to a call to *getpriority()*. If *getpriority()* returns the value -1, then *errno* can be checked  
 39743 to see if an error occurred or if the value is a legitimate nice value.

39744 **RETURN VALUE**

39745 Upon successful completion, *getpriority()* shall return an integer in the range -{NZERO} to  
 39746 {NZERO}-1. Otherwise, -1 shall be returned and *errno* set to indicate the error.

39747 Upon successful completion, *setpriority()* shall return 0; otherwise, -1 shall be returned and *errno*  
 39748 set to indicate the error.

39749 **ERRORS**

39750 The *getpriority()* and *setpriority()* functions shall fail if:

39751 [ESRCH] No process could be located using the *which* and *who* argument values  
39752 specified.

39753 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*  
39754 argument is not a valid process ID, process group ID, or user ID.

39755 In addition, *setpriority()* may fail if:

39756 [EPERM] A process was located, but neither the real nor effective user ID of the  
39757 executing process match the effective user ID of the process whose nice value  
39758 is being changed.

39759 [EACCES] A request was made to change the nice value to a lower numeric value and the  
39760 current process does not have appropriate privileges.

39761 **EXAMPLES**39762 **Using *getpriority()***

39763 The following example returns the current scheduling priority for the process ID returned by the  
39764 call to *getpid()*.

```
39765 #include <sys/resource.h>
39766 ...
39767 int which = PRIO_PROCESS;
39768 id_t pid;
39769 int ret;

39770 pid = getpid();
39771 ret = getpriority(which, pid);
```

39772 **Using *setpriority()***

39773 The following example sets the priority for the current process ID to -20.

```
39774 #include <sys/resource.h>
39775 ...
39776 int which = PRIO_PROCESS;
39777 id_t pid;
39778 int priority = -20;
39779 int ret;

39780 pid = getpid();
39781 ret = setpriority(which, pid, priority);
```

39782 **APPLICATION USAGE**

39783 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value  
39784  $-\{\text{NZERO}\}$ ). The nice value is in the range  $[0, 2 * \{\text{NZERO}\} - 1]$ , while the return value for  
39785 *getpriority()* and the third parameter for *setpriority()* are in the range  $[-\{\text{NZERO}\}, \{\text{NZERO}\} - 1]$ .

39786 **RATIONALE**

39787 None.

39788 **FUTURE DIRECTIONS**

39789 None.

39790 **SEE ALSO**39791 *nice()*, *sched\_get\_priority\_max()*, *sched\_setscheduler()*39792 XBD <[sys/resource.h](#)>39793 **CHANGE HISTORY**

39794 First released in Issue 4, Version 2.

39795 **Issue 5**

39796 Moved from X/OPEN UNIX extension to BASE.

39797 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion  
39798 with functionality in the POSIX Realtime Extension.

39799 **NAME**

39800 getprotobyname, getprotobynumber, getprotoent — network protocol database functions

39801 **SYNOPSIS**

39802 #include <netdb.h>

39803 struct protoent \*getprotobyname(const char \*name);

39804 struct protoent \*getprotobynumber(int proto);

39805 struct protoent \*getprotoent(void);

39806 **DESCRIPTION**

39807 Refer to *endprotoent()*.

39808 **NAME**

39809       getpwent — get user database entry

39810 **SYNOPSIS**

```
39811 XSI       #include <pwd.h>  
39812       struct passwd *getpwent(void);
```

39813 **DESCRIPTION**39814       Refer to *endpwent()*.

39815 **NAME**

39816 getpwnam, getpwnam\_r — search user database for a name

39817 **SYNOPSIS**

```
39818 #include <pwd.h>
39819 struct passwd *getpwnam(const char *name);
39820 int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
39821 size_t bufsize, struct passwd **result);
```

39822 **DESCRIPTION**39823 The *getpwnam()* function shall search the user database for an entry with a matching *name*.39824 The *getpwnam()* function need not be thread-safe.39825 Applications wishing to check for error situations should set *errno* to 0 before calling  
39826 *getpwnam()*. If *getpwnam()* returns a null pointer and *errno* is non-zero, an error occurred.

39827 The *getpwnam\_r()* function shall update the **passwd** structure pointed to by *pwd* and store a  
39828 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry  
39829 from the user database with a matching *name*. Storage referenced by the structure is allocated  
39830 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to  
39831 *sysconf(\_SC\_GETPW\_R\_SIZE\_MAX)* returns either  $-1$  without changing *errno* or an initial value  
39832 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to  
39833 by *result* on error or if the requested entry is not found.

39834 **RETURN VALUE**

39835 The *getpwnam()* function shall return a pointer to a **struct passwd** with the structure as defined  
39836 in **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested  
39837 entry is not found, or an error occurs. If the requested entry was not found, *errno* shall not be  
39838 changed. On error, *errno* shall be set to indicate the error.

39839 The application shall not modify the structure to which the return value points, nor any storage  
39840 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
39841 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
39842 subsequent call to *getpwent()*, *getpwnam()*, or *getpwuid()*. The returned pointer, and pointers  
39843 within the structure, might also be invalidated if the calling thread is terminated.

39844 The *getpwnam\_r()* function shall return zero on success or if the requested entry was not found  
39845 and no error has occurred. If an error has occurred, an error number shall be returned to indicate  
39846 the error.

39847 **ERRORS**

39848 These functions may fail if:

- 39849 [EIO] An I/O error has occurred.
- 39850 [EINTR] A signal was caught during *getpwnam()*.
- 39851 [EMFILE] All file descriptors available to the process are currently open.
- 39852 [ENFILE] The maximum allowable number of files is currently open in the system.

39853 The *getpwnam\_r()* function shall fail if:

- 39854 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be  
39855 referenced by the resulting **passwd** structure.

39856 **EXAMPLES**

39857 Note that `sysconf(_SC_GETPW_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size  
 39858 of the buffer needed to store all the groups returned. This example shows how an application  
 39859 can allocate a buffer of sufficient size to work with `getpwnam_r()`.

```

39860 long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
39861 size_t len;
39862 if (initlen == -1)
39863     /* Default initial length. */
39864     len = 1024;
39865 else
39866     len = (size_t) initlen;
39867 struct passwd result;
39868 struct passwd *resultp;
39869 char *buffer = malloc(len);
39870 if (buffer == NULL)
39871     ...handle error...
39872 int e;
39873 while ((e = getpwnam_r("someuser", &result, buffer, len, &resultp))
39874        == ERANGE)
39875     {
39876     size_t newlen = 2 * len;
39877     if (newlen < len)
39878         ...handle error...
39879     len = newlen;
39880     char *newbuffer = realloc(buffer, len);
39881     if (newbuffer == NULL)
39882         ...handle error...
39883     buffer = newbuffer;
39884     }
39885 if (e != 0)
39886     ...handle error...
39887 free (buffer);

```

39888 **Getting an Entry for the Login Name**

39889 The following example uses the `getlogin()` function to return the name of the user who logged in;  
 39890 this information is passed to the `getpwnam()` function to get the user database entry for that user.

```

39891 #include <sys/types.h>
39892 #include <pwd.h>
39893 #include <unistd.h>
39894 #include <stdio.h>
39895 #include <stdlib.h>
39896 ...
39897 char *lgn;
39898 struct passwd *pw;
39899 ...
39900 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
39901     fprintf(stderr, "Get of user information failed.\n"); exit(1);
39902 }
39903 ...

```

39904 **APPLICATION USAGE**

39905 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
 39906 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 39907 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 39908 with the real user ID of the process.

39909 The *getpwnam\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 39910 possibly using a static data area that may be overwritten by each call.

39911 Portable applications should take into account that it is usual for an implementation to return `-1`  
 39912 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

39913 **RATIONALE**

39914 None.

39915 **FUTURE DIRECTIONS**

39916 None.

39917 **SEE ALSO**

39918 *getpwuid()*, *sysconf()*

39919 XBD `<pwd.h>`, `<sys/types.h>`

39920 **CHANGE HISTORY**

39921 First released in Issue 1. Derived from System V Release 2.0.

39922 **Issue 5**

39923 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
 39924 VALUE section.

39925 The *getpwnam\_r()* function is included for alignment with the POSIX Threads Extension.

39926 A note indicating that the *getpwnam()* function need not be reentrant is added to the  
 39927 DESCRIPTION.

39928 **Issue 6**

39929 The *getpwnam\_r()* function is marked as part of the Thread-Safe Functions option.

39930 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
 39931 describing matching the *name*.

39932 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

39933 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

39934 The following new requirements on POSIX implementations derive from alignment with the  
 39935 Single UNIX Specification:

- 39936 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
 39937 required for conforming implementations of previous POSIX specifications, it was not  
 39938 required for UNIX applications.
- 39939 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 39940 • The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

39941 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 39942 its avoidance of possibly using a static data area.

39943 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
 39944 buffer from *bufsize* characters to bytes.



39945 **Issue 7**

- 39946 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 39947 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 39948 SD5-XSH-ERN-166 is applied.
- 39949 The *getpwnam\_r()* function is moved from the Thread-Safe Functions option to the Base.
- 39950 A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc()*.
- 39951
- 39952 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0255 [75,428] is applied.
- 39953 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0170 [808] and XSH/TC2-2008/0171
- 39954 [656] are applied.

39955 **Issue 8**

- 39956 Austin Group Defect 398 is applied, changing the [ERANGE] error from “may fail” to “shall fail”.
- 39957
- 39958 Austin Group Defect 1570 is applied, removing extra spacing in “==”.

39959 **NAME**39960 `getpwuid, getpwuid_r` — search user database for a user ID39961 **SYNOPSIS**

```
39962 #include <pwd.h>
39963 struct passwd *getpwuid(uid_t uid);
39964 int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
39965 size_t bufsize, struct passwd **result);
```

39966 **DESCRIPTION**39967 The `getpwuid()` function shall search the user database for an entry with a matching `uid`.39968 The `getpwuid()` function need not be thread-safe.39969 Applications wishing to check for error situations should set `errno` to 0 before calling `getpwuid()`.  
39970 If `getpwuid()` returns a null pointer and `errno` is set to non-zero, an error occurred.

39971 The `getpwuid_r()` function shall update the **passwd** structure pointed to by `pwd` and store a  
39972 pointer to that structure at the location pointed to by `result`. The structure shall contain an entry  
39973 from the user database with a matching `uid`. Storage referenced by the structure is allocated  
39974 from the memory provided with the `buffer` parameter, which is `bufsize` bytes in size. A call to  
39975 `sysconf(_SC_GETPW_R_SIZE_MAX)` returns either `-1` without changing `errno` or an initial value  
39976 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to  
39977 by `result` on error or if the requested entry is not found.

39978 **RETURN VALUE**

39979 The `getpwuid()` function shall return a pointer to a **struct passwd** with the structure as defined in  
39980 `<pwd.h>` with a matching entry if found. A null pointer shall be returned if the requested entry  
39981 is not found, or an error occurs. If the requested entry was not found, `errno` shall not be changed.  
39982 On error, `errno` shall be set to indicate the error.

39983 The application shall not modify the structure to which the return value points, nor any storage  
39984 areas pointed to by pointers within the structure. The returned pointer, and pointers within the  
39985 structure, might be invalidated or the structure or the storage areas might be overwritten by a  
39986 subsequent call to `getpwent()`, `getpwnam()`, or `getpwuid()`. The returned pointer, and pointers  
39987 within the structure, might also be invalidated if the calling thread is terminated.

39988 If successful, the `getpwuid_r()` function shall return zero; otherwise, an error number shall be  
39989 returned to indicate the error.39990 **ERRORS**

39991 These functions may fail if:

- 39992 [EIO] An I/O error has occurred.
- 39993 [EINTR] A signal was caught during `getpwuid()`.
- 39994 [EMFILE] All file descriptors available to the process are currently open.
- 39995 [ENFILE] The maximum allowable number of files is currently open in the system.

39996 The `getpwuid_r()` function shall fail if:

- 39997 [ERANGE] Insufficient storage was supplied via `buffer` and `bufsize` to contain the data to be  
39998 referenced by the resulting **passwd** structure.

39999 **EXAMPLES**

40000 Note that `sysconf(_SC_GETPW_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size  
 40001 of the buffer needed to store all the groups returned. This example shows how an application  
 40002 can allocate a buffer of sufficient size to work with `getpwuid_r()`.

```

40003 long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
40004 size_t len;
40005 if (initlen == -1)
40006     /* Default initial length. */
40007     len = 1024;
40008 else
40009     len = (size_t) initlen;
40010 struct passwd result;
40011 struct passwd *resultp;
40012 char *buffer = malloc(len);
40013 if (buffer == NULL)
40014     ...handle error...
40015 int e;
40016 while ((e = getpwuid_r(42, &result, buffer, len, &resultp)) == ERANGE)
40017     {
40018     size_t newlen = 2 * len;
40019     if (newlen < len)
40020         ...handle error...
40021     len = newlen;
40022     char *newbuffer = realloc(buffer, len);
40023     if (newbuffer == NULL)
40024         ...handle error...
40025     buffer = newbuffer;
40026     }
40027 if (e != 0)
40028     ...handle error...
40029 free (buffer);

```

40030 **Getting an Entry for the Root User**

40031 The following example gets the user database entry for the user with user ID 0 (root).

```

40032 #include <sys/types.h>
40033 #include <pwd.h>
40034 ...
40035 uid_t id = 0;
40036 struct passwd *pwd;
40037 pwd = getpwuid(id);

```

40038 **Finding the Name for the Effective User ID**

40039 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to  
 40040 store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function  
 40041 shall return the effective user ID of the calling process; this is used as the search criteria for the  
 40042 *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that  
 40043 user ID value.

```
40044 #include <unistd.h>
40045 #include <sys/types.h>
40046 #include <pwd.h>
40047 ...
40048 struct passwd *pws;
40049 pws = getpwuid(geteuid());
```

40050 **Finding an Entry in the User Database**

40051 The following example uses *getpwuid()* to search the user database for a user ID that was  
 40052 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not  
 40053 found, the program prints the numeric value of the user ID for the entry.

```
40054 #include <sys/types.h>
40055 #include <pwd.h>
40056 #include <stdio.h>
40057 ...
40058 struct stat statbuf;
40059 struct passwd *pwd;
40060 ...
40061 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
40062     printf(" %-8.8s", pwd->pw_name);
40063 else
40064     printf(" %-8d", statbuf.st_uid);
```

40065 **APPLICATION USAGE**

40066 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns  
 40067 the name associated with the effective user ID of the process; *getlogin()* returns the name  
 40068 associated with the current login activity; and *getpwuid(getuid())* returns the name associated  
 40069 with the real user ID of the process.

40070 The *getpwuid\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 40071 possibly using a static data area that may be overwritten by each call.

40072 Portable applications should take into account that it is usual for an implementation to return  $-1$   
 40073 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

40074 **RATIONALE**

40075 None.

40076 **FUTURE DIRECTIONS**

40077 None.

40078 **SEE ALSO**

40079 [\*getpwnam\(\)\*](#), [\*geteuid\(\)\*](#), [\*getuid\(\)\*](#), [\*getlogin\(\)\*](#), [\*sysconf\(\)\*](#)

40080 XBD [`<pwd.h>`](#), [`<sys/types.h>`](#)

40081 **CHANGE HISTORY**

40082 First released in Issue 1. Derived from System V Release 2.0.

40083 **Issue 5**40084 Normative text previously in the APPLICATION USAGE section is moved to the RETURN  
40085 VALUE section.40086 The `getpwuid_r()` function is included for alignment with the POSIX Threads Extension.40087 A note indicating that the `getpwuid()` function need not be reentrant is added to the  
40088 DESCRIPTION.40089 **Issue 6**40090 The `getpwuid_r()` function is marked as part of the Thread-Safe Functions option.40091 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION  
40092 describing matching the `uid`.40093 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

40094 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

40095 The following new requirements on POSIX implementations derive from alignment with the  
40096 Single UNIX Specification:40097 

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
40098 required for conforming implementations of previous POSIX specifications, it was not  
40099 required for UNIX applications.

40100 

- In the RETURN VALUE section, the requirement to set `errno` on error is added.

40101 

- The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

40102 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
40103 its avoidance of possibly using a static data area.40104 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the  
40105 buffer from `bufsize` characters to bytes.40106 **Issue 7**

40107 Austin Group Interpretation 1003.1-2001 #156 is applied.

40108 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

40109 SD5-XSH-ERN-166 is applied.

40110 The `getpwuid_r()` function is moved from the Thread-Safe Functions option to the Base.40111 A minor addition is made to the EXAMPLES section, reminding the application developer to  
40112 free memory allocated as if by `malloc()`.

40113 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0256 [75] is applied.

40114 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0172 [808] and XSH/TC2-2008/0173  
40115 [656] are applied.40116 **Issue 8**40117 Austin Group Defect 398 is applied, changing the [ERANGE] error from “may fail” to “shall  
40118 fail”.

40119 Austin Group Defect 1570 is applied, removing extra spacing in “==”.

40120 **NAME**

40121 getresgid — get real group ID, effective group ID, and saved set-group-ID

40122 **SYNOPSIS**

```
40123 XSI #include <unistd.h>
40124 int getresgid(gid_t *restrict rgid, gid_t *restrict egid,
40125 gid_t *restrict sgid);
```

40126 **DESCRIPTION**

40127 The *getresgid()* function shall store the real group ID, effective group ID, and saved set-group-ID  
40128 of the calling process in the locations pointed to by the arguments *rgid*, *egid*, and *sgid*,  
40129 respectively.

40130 **RETURN VALUE**

40131 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
40132 indicate the error.

40133 **ERRORS**

40134 No errors are defined.

40135 **EXAMPLES**

40136 None.

40137 **APPLICATION USAGE**

40138 None.

40139 **RATIONALE**

40140 None.

40141 **FUTURE DIRECTIONS**

40142 None.

40143 **SEE ALSO**

40144 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*,  
40145 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

40146 XBD &lt;unistd.h&gt;

40147 **CHANGE HISTORY**

40148 First released in Issue 8.

40149 **NAME**

40150 getresuid — get real user ID, effective user ID, and saved set-user-ID

40151 **SYNOPSIS**

```
40152 XSI #include <unistd.h>
40153 int getresuid(uid_t *restrict ruid, uid_t *restrict euid,
40154             uid_t *restrict suid);
```

40155 **DESCRIPTION**

40156 The *getresuid()* function shall store the real user ID, effective user ID, and saved set-user-ID of  
40157 the calling process in the locations pointed to by the arguments *ruid*, *euid*, and *suid*, respectively.

40158 **RETURN VALUE**

40159 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
40160 indicate the error.

40161 **ERRORS**

40162 No errors are defined.

40163 **EXAMPLES**

40164 None.

40165 **APPLICATION USAGE**

40166 None.

40167 **RATIONALE**

40168 None.

40169 **FUTURE DIRECTIONS**

40170 None.

40171 **SEE ALSO**

40172 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*,  
40173 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

40174 XBD <[unistd.h](#)>40175 **CHANGE HISTORY**

40176 First released in Issue 8.

40177 **NAME**

40178 getrlimit, setrlimit — control maximum resource consumption

40179 **SYNOPSIS**

```
40180 #include <sys/resource.h>
40181 int getrlimit(int resource, struct rlimit *rlp);
40182 int setrlimit(int resource, const struct rlimit *rlp);
```

40183 **DESCRIPTION**

40184 The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption  
40185 of a variety of resources.

40186 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as  
40187 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim\_cur*  
40188 member specifies the current or soft limit and the *rlim\_max* member specifies the maximum or  
40189 hard limit. Soft limits can be changed by a process to any value that is less than or equal to the  
40190 hard limit. A process can (irreversibly) lower its hard limit to any value that is greater than or  
40191 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both  
40192 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints  
40193 described above.

40194 The value RLIM\_INFINITY, defined in **<sys/resource.h>**, shall be considered to be larger than  
40195 any other limit value. If a call to *getrlimit()* returns RLIM\_INFINITY for a resource, it means the  
40196 implementation shall not enforce limits on that resource. Specifying RLIM\_INFINITY as any  
40197 resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource  
40198 limit.

40199 The following resources are defined:

40200 **RLIMIT\_CORE** This is the maximum size of a file containing a core image, in bytes, that  
40201 can be created by a process. A limit of 0 shall prevent the creation of such  
40202 a file. If this limit is exceeded, the writing of a file containing a core image  
40203 shall terminate at this size. Note that the production of such a file may be  
40204 one of the implementation-defined actions for abnormal termination.

40205 XSI **RLIMIT\_CPU** This is the maximum amount of CPU time, in seconds, used by a process.  
40206 If this limit is exceeded, SIGXCPU shall be generated for the process. If  
40207 the process is catching or ignoring SIGXCPU, or all threads belonging to  
40208 that process are blocking SIGXCPU, the behavior is unspecified.

40209 **RLIMIT\_DATA** This is the maximum size of a data segment of the process, in bytes. If  
40210 this limit is exceeded, the *malloc()* function shall fail with *errno* set to  
40211 [ENOMEM].

40212 **RLIMIT\_FSIZE** This is the maximum size of a file, in bytes, that can be created by a  
40213 process. If a write or truncate operation would cause this limit to be  
40214 XSI exceeded, a SIGXFSZ shall be generated for the thread; if the thread is  
40215 blocking, or the process is catching or ignoring SIGXFSZ, the operation  
40216 shall fail with an [EFBIG] error.

40217 **RLIMIT\_NOFILE** This is a number one greater than the maximum value that the system  
40218 shall assign to a newly-created descriptor. If this limit is exceeded,  
40219 functions that allocate a file descriptor shall fail with *errno* set to  
40220 [EMFILE]. This limit constrains the number of file descriptors that a  
40221 process can allocate.



40222 RLIMIT\_STACK This is the maximum size of the initial thread's stack, in bytes. The  
 40223 implementation does not automatically grow the stack beyond this limit.  
 40224 If this limit is exceeded, SIGSEGV shall be generated for the thread. If the  
 40225 thread is blocking SIGSEGV, or the process is ignoring or catching  
 40226 SIGSEGV and has not made arrangements to use an alternate stack, the  
 40227 disposition of SIGSEGV shall be set to SIG\_DFL before it is generated.

40228 RLIMIT\_AS This is the maximum size of total available memory of the process, in  
 40229 bytes. If this limit is exceeded, the *malloc()* and *mmap()* functions shall fail  
 40230 with *errno* set to [ENOMEM]. In addition, the automatic stack growth  
 40231 fails with the effects outlined above.

40232 When using the *getrlimit()* function, if a resource limit can be represented correctly in an object  
 40233 of type **rlim\_t**, then its representation is returned; otherwise, if the value of the resource limit is  
 40234 equal to that of the corresponding saved hard limit, the value returned shall be  
 40235 RLIM\_SAVED\_MAX; otherwise, the value returned shall be RLIM\_SAVED\_CUR.

40236 When using the *setrlimit()* function, if the requested new limit is RLIM\_INFINITY, the new limit  
 40237 shall be "no limit"; otherwise, if the requested new limit is RLIM\_SAVED\_MAX, the new limit  
 40238 shall be the corresponding saved hard limit; otherwise, if the requested new limit is  
 40239 RLIM\_SAVED\_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the  
 40240 new limit shall be the requested value. In addition, if the corresponding saved limit can be  
 40241 represented correctly in an object of type **rlim\_t** then it shall be overwritten with the new limit.

40242 The result of setting a limit to RLIM\_SAVED\_MAX or RLIM\_SAVED\_CUR is unspecified unless  
 40243 a previous call to *getrlimit()* returned that value as the soft or hard limit for the corresponding  
 40244 resource limit.

40245 The determination of whether a limit can be correctly represented in an object of type **rlim\_t** is  
 40246 implementation-defined. For example, some implementations permit a limit whose value is  
 40247 greater than RLIM\_INFINITY and others do not.

40248 The *exec* family of functions shall cause resource limits to be saved.

#### 40249 RETURN VALUE

40250 Upon successful completion, *getrlimit()* and *setrlimit()* shall return 0. Otherwise, these functions  
 40251 shall return -1 and set *errno* to indicate the error.

#### 40252 ERRORS

40253 The *getrlimit()* and *setrlimit()* functions shall fail if:

40254 [EINVAL] An invalid *resource* was specified; or in a *setrlimit()* call, the new *rlim\_cur*  
 40255 exceeds the new *rlim\_max*.

40256 [EPERM] The limit specified to *setrlimit()* would have raised the maximum limit value,  
 40257 and the calling process does not have appropriate privileges.

40258 The *setrlimit()* function may fail if:

40259 [EINVAL] The limit specified cannot be lowered because current usage is already higher  
 40260 than the limit.

**40261 EXAMPLES**

40262 None.

**40263 APPLICATION USAGE**

40264 If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the value  
40265 of {\_POSIX\_OPEN\_MAX} from <limits.h>, unexpected behavior may occur.

40266 If a process attempts to set the hard limit or soft limit for RLIMIT\_NOFILE to less than the  
40267 highest currently open file descriptor +1, unexpected behavior may occur.

**40268 RATIONALE**

40269 These functions were previously part of the XSI option and have been moved to the Base so that  
40270 portable shells, and other utilities that need to relay the wait status of a child process to a parent,  
40271 can terminate themselves with the same signal that terminated the child but without  
40272 overwriting a core image created by the child (through setting RLIMIT\_CORE to zero, which  
40273 disables core image creation). The RLIMIT\_CPU and RLIMIT\_FSIZE limits remain in the XSI  
40274 option because they relate to other XSI functionality (SIGXCPU and SIGXFSZ).

40275 It should be noted that RLIMIT\_STACK applies “at least” to the stack of the initial thread in the  
40276 process, and not to the sum of all the stacks in the process, as that would be very limiting unless  
40277 the value is so big as to provide no value at all with a single thread.

**40278 FUTURE DIRECTIONS**

40279 None.

**40280 SEE ALSO**

40281 *exec, fork(), malloc(), open(), sigaltstack(), sysconf()*

40282 XBD <sys/resource.h>

40283 XCU *ulimit*

**40284 CHANGE HISTORY**

40285 First released in Issue 4, Version 2.

**40286 Issue 5**

40287 Moved from X/OPEN UNIX extension to BASE.

40288 An APPLICATION USAGE section is added.

40289 Large File Summit extensions are added.

**40290 Issue 6**

40291 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for  
40292 RLIMIT\_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing  
40293 with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of  
40294 a process attempting to set the hard or soft limit for RLIMIT\_NOFILE less than the highest  
40295 currently open file descriptor +1.

40296 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/46 is applied, updating the definition of  
40297 RLIMIT\_STACK in the DESCRIPTION from “the maximum size of a process stack” to “the  
40298 maximum size of the initial thread’s stack”. Text is added to the RATIONALE section.

**40299 Issue 8**

40300 Austin Group Defects 51 and 1669 are applied, moving the *getrlimit()* and *setrlimit()* functions,  
40301 excluding the RLIMIT\_CPU limit, from the XSI option to the Base.

40302 Austin Group Defect 1141 is applied, changing the description of RLIMIT\_CORE.

40303 Austin Group Defect 1416 is applied, changing some uses of “may” to “can” and one to “shall”.

40304

Austin Group Defect 1418 is applied, changing the SEE ALSO section.

40305 **NAME**

40306 getrusage — get information about resource utilization

40307 **SYNOPSIS**

```
40308 XSI      #include <sys/resource.h>
40309      int getrusage(int who, struct rusage *r_usage);
```

40310 **DESCRIPTION**

40311 The *getrusage()* function shall provide measures of the resources used by the current process or  
 40312 its terminated and waited-for child processes. If the value of the *who* argument is  
 40313 RUSAGE\_SELF, information shall be returned about resources used by the current process. If the  
 40314 value of the *who* argument is RUSAGE\_CHILDREN, information shall be returned about  
 40315 resources used by the children of the current process that have terminated and been waited-for  
 40316 and their children that have terminated and been waited-for, recursively. If the child is never  
 40317 waited for (for example, if the parent has SA\_NOCLDWAIT set or sets SIGCHLD to SIG\_IGN),  
 40318 the resource information for the child process is discarded and not included in the resource  
 40319 information provided by *getrusage()*.

40320 The *r\_usage* argument is a pointer to an object of type **struct rusage** in which the returned  
 40321 information is stored.

40322 **RETURN VALUE**

40323 Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*  
 40324 set to indicate the error.

40325 **ERRORS**

40326 The *getrusage()* function shall fail if:

40327 [EINVAL] The value of the *who* argument is not valid.

40328 **EXAMPLES**40329 **Using getrusage()**

40330 The following example returns information about the resources used by the current process.

```
40331 #include <sys/resource.h>
40332 ...
40333 int who = RUSAGE_SELF;
40334 struct rusage usage;
40335 int ret;
40336
40337 ret = getrusage(who, &usage);
```

40337 **APPLICATION USAGE**

40338 None.

40339 **RATIONALE**

40340 None.

40341 **FUTURE DIRECTIONS**

40342 None.

40343 **SEE ALSO**

40344 [exit\(\)](#), [sigaction\(\)](#), [time\(\)](#), [times\(\)](#), [wait\(\)](#)

40345 XBD [<sys/resource.h>](#)

40346 **CHANGE HISTORY**

40347 First released in Issue 4, Version 2.

40348 **Issue 5**

40349 Moved from X/OPEN UNIX extension to BASE.

40350 **Issue 8**

40351 Austin Group Defect 1336 is applied, clarifying the requirements when the *who* argument is  
40352 RUSAGE\_CHILDREN.

40353 **NAME**

40354        getservbyname, getservbyport, getservent — network services database functions

40355 **SYNOPSIS**

40356        #include <netdb.h>

40357        struct servent \*getservbyname(const char \*name, const char \*proto);

40358        struct servent \*getservbyport(int port, const char \*proto);

40359        struct servent \*getservent(void);

40360 **DESCRIPTION**

40361        Refer to *endservent()*.

40362 **NAME**

40363 getsid — get the process group ID of a session leader

40364 **SYNOPSIS**

40365 #include &lt;unistd.h&gt;

40366 pid\_t getsid(pid\_t pid);

40367 **DESCRIPTION**40368 The *getsid()* function shall obtain the process group ID of the process that is the session leader of  
40369 the process specified by *pid*. If *pid* is (**pid\_t**)0, it specifies the calling process.40370 **RETURN VALUE**40371 Upon successful completion, *getsid()* shall return the process group ID of the session leader of  
40372 the specified process. Otherwise, it shall return  $-1$  and set *errno* to indicate the error.40373 **ERRORS**40374 The *getsid()* function shall fail if:40375 [EPERM] The process specified by *pid* is not in the same session as the calling process,  
40376 and the implementation does not allow access to the process group ID of the  
40377 session leader of that process from the calling process.40378 [ESRCH] There is no process with a process ID equal to *pid*.40379 **EXAMPLES**

40380 None.

40381 **APPLICATION USAGE**

40382 None.

40383 **RATIONALE**

40384 None.

40385 **FUTURE DIRECTIONS**

40386 None.

40387 **SEE ALSO**40388 *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*

40389 XBD &lt;unistd.h&gt;

40390 **CHANGE HISTORY**

40391 First released in Issue 4, Version 2.

40392 **Issue 5**

40393 Moved from X/OPEN UNIX extension to BASE.

40394 **Issue 7**40395 The *getsid()* function is moved from the XSI option to the Base.

40396 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0258 [421] is applied.

40397 **NAME**

40398       getsockname — get the socket name

40399 **SYNOPSIS**

40400       #include &lt;sys/socket.h&gt;

40401       int getsockname(int *socket*, struct sockaddr \*restrict *address*,  
40402                   socklen\_t \*restrict *address\_len*);40403 **DESCRIPTION**40404       The *getsockname()* function shall retrieve the locally-bound name of the specified socket, store  
40405       this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of  
40406       this address in the object pointed to by the *address\_len* argument.40407       The *address\_len* argument points to a **socklen\_t** object which on input specifies the length of the  
40408       supplied **sockaddr** structure, and on output specifies the length of the socket address. If the  
40409       actual length of the address is greater than the length of the supplied **sockaddr** structure, the  
40410       stored address shall be truncated.40411       If the socket has not been bound to a local name, the value stored in the object pointed to by  
40412       *address* is unspecified.40413 **RETURN VALUE**40414       Upon successful completion, 0 shall be returned, the *address* argument shall point to the address  
40415       of the socket, and the *address\_len* argument shall point to the length of the address. Otherwise, -1  
40416       shall be returned and *errno* set to indicate the error.40417 **ERRORS**40418       The *getsockname()* function shall fail if:40419       [EBADF]        The *socket* argument is not a valid file descriptor.40420       [ENOTSOCK]    The *socket* argument does not refer to a socket.

40421       [EOPNOTSUPP]  The operation is not supported for this socket's protocol.

40422       The *getsockname()* function may fail if:

40423       [EINVAL]       The socket has been shut down.

40424       [ENOBUFS]     Insufficient resources were available in the system to complete the function.

40425 **EXAMPLES**

40426       None.

40427 **APPLICATION USAGE**40428       For AF\_UNIX sockets, it is recommended that *address* points to a buffer of length greater than  
40429       sizeof(struct sockaddr\_un) which has been initialized with null bytes. That way, even if  
40430       the implementation supports the use of all bytes of *sun\_path* without a terminating null byte, the  
40431       larger buffer guarantees that the *sun\_path* member can then be passed to other interfaces that  
40432       expect a null-terminated string. If no truncation occurred based on the input value of *address\_len*,  
40433       it is unspecified whether the returned *address\_len* will be sizeof(struct sockaddr\_un), or  
40434       merely a value at least as large as offsetof(struct sockaddr\_un, sun\_path) plus the  
40435       number of non-null bytes stored in *sun\_path*.40436 **RATIONALE**

40437       None.



40438 **FUTURE DIRECTIONS**

40439 None.

40440 **SEE ALSO**40441 *accept()*, *bind()*, *getpeername()*, *socket()*40442 XBD <[sys/socket.h](#)>40443 **CHANGE HISTORY**

40444 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

40445 The **restrict** keyword is added to the *getsockname()* prototype for alignment with the  
40446 ISO/IEC 9899:1999 standard.40447 **Issue 7**

40448 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0259 [464] is applied.

40449 **Issue 8**40450 Austin Group Defect 561 is applied, adding a paragraph about *sun\_path* to APPLICATION  
40451 USAGE.40452 Austin Group Defect 1565 is applied, changing the description of *address\_len*.

40453 **NAME**

40454 getsockopt — get the socket options

40455 **SYNOPSIS**

```
40456 #include <sys/socket.h>
40457 int getsockopt(int socket, int level, int option_name,
40458 void *restrict option_value, socklen_t *restrict option_len);
```

40459 **DESCRIPTION**40460 The *getsockopt()* function manipulates options associated with a socket.

40461 The *getsockopt()* function shall retrieve the value for the option specified by the *option\_name*  
40462 argument for the socket specified by the *socket* argument. If the size of the option value is greater  
40463 than *option\_len*, the value stored in the object pointed to by the *option\_value* argument shall be  
40464 silently truncated. Otherwise, the object pointed to by the *option\_len* argument shall be modified  
40465 to indicate the actual length of the value.

40466 The *level* argument specifies the protocol level at which the option resides. To retrieve options at  
40467 the socket level, specify the *level* argument as SOL\_SOCKET. To retrieve options at other levels,  
40468 supply the appropriate level identifier for the protocol controlling the option. For example, to  
40469 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to  
40470 IPPROTO\_TCP as defined in the **<netinet/in.h>** header.

40471 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*  
40472 function.

40473 The *option\_name* argument specifies a single option to be retrieved. It can be one of the socket-  
40474 level options defined in **<sys/socket.h>** and described in [Section 2.10.16](#) (on page 554).

40475 **RETURN VALUE**

40476 Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno*  
40477 set to indicate the error.

40478 **ERRORS**40479 The *getsockopt()* function shall fail if:

40480 [EBADF] The *socket* argument is not a valid file descriptor.

40481 [EINVAL] The specified option is invalid at the specified socket level.

40482 [ENOPROTOOPT]  
40483 The option is not supported by the protocol.

40484 [ENOTSOCK] The *socket* argument does not refer to a socket.

40485 The *getsockopt()* function may fail if:

40486 [EACCES] The calling process does not have appropriate privileges.

40487 [EINVAL] The socket has been shut down.

40488 [ENOBUFS] Insufficient resources are available in the system to complete the function.

40489 **EXAMPLES**

40490 None.

40491 **APPLICATION USAGE**

40492 None.

40493 **RATIONALE**

40494 None.

40495 **FUTURE DIRECTIONS**

40496 None.

40497 **SEE ALSO**40498 [Section 2.10.16](#) (on page 554), [bind\(\)](#), [close\(\)](#), [endprotoent\(\)](#), [setsockopt\(\)](#), [socket\(\)](#)40499 XBD [<sys/socket.h>](#), [<netinet/in.h>](#)40500 **CHANGE HISTORY**

40501 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

40502 The **restrict** keyword is added to the `getsockopt()` prototype for alignment with the  
40503 ISO/IEC 9899:1999 standard.40504 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/47 is applied, updating the description of  
40505 SO\_LINGER in the DESCRIPTION so that it blocks the calling thread rather than the process.40506 **Issue 7**40507 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options  
40508 that is now in [Section 2.10.16](#) (on page 554).

40509 **NAME**

40510 getsubopt — parse suboption arguments from a string

40511 **SYNOPSIS**

```
40512 CX #include <stdlib.h>
40513 int getsubopt(char **restrict optionp,
40514 char * const *restrict keylistp, char **restrict valuep);
```

40515 **DESCRIPTION**

40516 The *getsubopt()* function shall parse suboption arguments in a flag argument. Such options often  
 40517 result from the use of *getopt()*.

40518 The *getsubopt()* argument *optionp* is a pointer to a pointer to the option argument string. The  
 40519 suboption arguments shall be separated by <comma> characters and each may consist of either  
 40520 a single token, or a token-value pair separated by an <equals-sign>.

40521 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified  
 40522 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in  
 40523 *\*optionp*. Since <comma> characters delimit suboption arguments in *optionp*, they should not  
 40524 appear in any of the strings pointed to by *keylistp*. Similarly, because an <equals-sign> separates  
 40525 a token from its value, the application should not include an <equals-sign> in any of the strings  
 40526 pointed to by *keylistp*. The *getsubopt()* function shall not modify the *keylistp* vector.

40527 The *valuep* argument is the address of a value string pointer.

40528 If a <comma> appears in *optionp*, it shall be interpreted as a suboption separator. After <comma>  
 40529 characters have been processed, if there are one or more <equals-sign> characters in a suboption  
 40530 string, the first <equals-sign> in any suboption string shall be interpreted as a separator between  
 40531 a token and a value. Subsequent <equals-sign> characters in a suboption string shall be  
 40532 interpreted as part of the value.

40533 If the string at *\*optionp* contains only one suboption argument (equivalently, no <comma>  
 40534 characters), *getsubopt()* shall update *\*optionp* to point to the null character at the end of the  
 40535 string. Otherwise, it shall isolate the suboption argument by replacing the <comma> separator  
 40536 with a null character, and shall update *\*optionp* to point to the start of the next suboption  
 40537 argument. If the suboption argument has an associated value (equivalently, contains an <equals-  
 40538 sign>), *getsubopt()* shall update *\*valuep* to point to the value's first character. Otherwise, it shall  
 40539 set *\*valuep* to a null pointer. The calling application may use this information to determine  
 40540 whether the presence or absence of a value for the suboption is an error.

40541 Additionally, when *getsubopt()* fails to match the suboption argument with a token in the *keylistp*  
 40542 array, the calling application should decide if this is an error, or if the unrecognized option  
 40543 should be processed in another way.

40544 **RETURN VALUE**

40545 The *getsubopt()* function shall return the index of the matched token string, or -1 if no token  
 40546 strings were matched.

40547 **ERRORS**

40548 No errors are defined.

40549 **EXAMPLES**40550 **Parsing Suboptions**

40551 The following example uses the `getsubopt()` function to parse a *value* argument in the *optarg*  
 40552 external variable returned by a call to `getopt()`.

```

40553 #include <stdio.h>
40554 #include <stdlib.h>
40555 #include <unistd.h>

40556 int do_all;
40557 const char *type;
40558 int read_size;
40559 int write_size;
40560 int read_only;

40561 enum
40562 {
40563     RO_OPTION = 0,
40564     RW_OPTION,
40565     READ_SIZE_OPTION,
40566     WRITE_SIZE_OPTION
40567 };

40568 const char *mount_opts[] =
40569 {
40570     [RO_OPTION] = "ro",
40571     [RW_OPTION] = "rw",
40572     [READ_SIZE_OPTION] = "rsize",
40573     [WRITE_SIZE_OPTION] = "wsize",
40574     NULL
40575 };

40576 int
40577 main(int argc, char *argv[])
40578 {
40579     char *subopts, *value;
40580     int opt;

40581     while ((opt = getopt(argc, argv, "at:o:")) != -1)
40582         switch(opt)
40583         {
40584             case 'a':
40585                 do_all = 1;
40586                 break;
40587             case 't':
40588                 type = optarg;
40589                 break;
40590             case 'o':
40591                 subopts = optarg;
40592                 while (*subopts != ' ')
40593                     {
40594                         char *saved = subopts;
40595                         switch(getsubopt (&subopts, (char **)mount_opts,
```

```

40596         &value))
40597     {
40598     case RO_OPTION:
40599         read_only = 1;
40600         break;
40601     case RW_OPTION:
40602         read_only = 0;
40603         break;
40604     case READ_SIZE_OPTION:
40605         if (value == NULL)
40606             abort();
40607         read_size = atoi(value);
40608         break;
40609     case WRITE_SIZE_OPTION:
40610         if (value == NULL)
40611             abort();
40612         write_size = atoi(value);
40613         break;
40614     default:
40615         /* Unknown suboption. */
40616         printf("Unknown suboption `%s'\n", saved);
40617         abort();
40618     }
40619     }
40620     break;
40621     default:
40622         abort();
40623     }
40624     /* Do the real work. */
40625     return 0;
40626 }

```

40627 If the above example is invoked with:

```
40628 program -o ro,rsize=512
```

40629 then after option parsing, the variable *do\_all* will be 0, *type* will be a null pointer, *read\_size* will be  
 40630 512, *write\_size* will be 0, and *read\_only* will be 1. If it is invoked with:

```
40631 program -o oops
```

40632 it will print:

```
40633 "Unknown suboption `oops'"
```

40634 before aborting.

#### 40635 APPLICATION USAGE

40636 The value of *\*valuep* when *getsubopt()* returns  $-1$  is unspecified. Historical implementations  
 40637 provide various incompatible extensions to allow an application to access the suboption text that  
 40638 was not found in the *keylistp* array.

40639 **RATIONALE**

40640 The *keylistp* argument of *getsubopt()* is typed as **char \* const \*** to match historical practice.  
40641 However, the standard is clear that implementations will not modify either the array or the  
40642 strings contained in the array, as if the argument had been typed **const char \* const \***.

40643 **FUTURE DIRECTIONS**

40644 None.

40645 **SEE ALSO**

40646 [getopt\(\)](#)

40647 XBD <stdlib.h>

40648 **CHANGE HISTORY**

40649 First released in Issue 4, Version 2.

40650 **Issue 5**

40651 Moved from X/OPEN UNIX extension to BASE.

40652 **Issue 6**

40653 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error  
40654 in the SYNOPSIS.

40655 **Issue 7**

40656 The *getsubopt()* function is moved from the XSI option to the Base.

40657 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0260 [196], XSH/TC1-2008/0261 [196],  
40658 XSH/TC1-2008/0262 [196], XSH/TC1-2008/0263 [196], XSH/TC1-2008/0264 [196],  
40659 XSH/TC1-2008/0265 [196], XSH/TC1-2008/0266 [196], XSH/TC1-2008/0267 [196],  
40660 XSH/TC1-2008/0268 [196], and XSH/TC1-2008/0269 [196] are applied.

40661 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0174 [791] is applied.

40662 **Issue 8**

40663 Austin Group Defect 444 is applied, adding the **restrict** keyword to the *getsubopt()* prototype.

40664 **NAME**

40665 `dgettext`, `dgettext_l`, `dcgettext`, `dcgettext_l`, `gettext`, `gettext_l`, `ngettext`, `ngettext_l`, `dngettext`,  
 40666 `dngettext_l`, `dcngettext`, `dcngettext_l` — message handling functions

40667 **SYNOPSIS**

```
40668 #include <libintl.h>
40669 char *dgettext(const char *domainname, const char *msgid);
40670 char *dgettext_l(const char *domainname, const char *msgid,
40671 locale_t locale);
40672 char *dcgettext(const char *domainname, const char *msgid,
40673 int category);
40674 char *dcgettext_l(const char *domainname, const char *msgid,
40675 int category, locale_t locale);
40676 char *dngettext(const char *domainname, const char *msgid,
40677 const char *msgid_plural, unsigned long int n);
40678 char *dngettext_l(const char *domainname, const char *msgid,
40679 const char *msgid_plural, unsigned long int n,
40680 locale_t locale);
40681 char *dcngettext(const char *domainname, const char *msgid,
40682 const char *msgid_plural, unsigned long int n,
40683 int category);
40684 char *dcngettext_l(const char *domainname, const char *msgid,
40685 const char *msgid_plural, unsigned long int n,
40686 int category, locale_t locale);
40687 char *gettext(const char *msgid);
40688 char *gettext_l(const char *msgid, locale_t locale);
40689 char *ngettext(const char *msgid, const char *msgid_plural,
40690 unsigned long int n);
40691 char *ngettext_l(const char *msgid, const char *msgid_plural,
40692 unsigned long int n, locale_t locale);
```

40693 **DESCRIPTION**

40694 The `gettext()` function shall:

- 40695 • attempt to locate a suitable messages object (described in detail below) for the  
 40696 `LC_MESSAGES` category in the current locale, and for the current text domain (see  
 40697 `bindtextdomain()`), containing the string identified by `msgid`,
- 40698 • retrieve the string identified by `msgid` from the messages object,
- 40699 • convert the string to the output codeset if necessary (described in detail below), and
- 40700 • return the result.

40701 If the locale name in effect is "POSIX" or "C" (i.e. the name associated with the `LC_MESSAGES`  
 40702 locale category in the current locale), or if no suitable messages object exists, or if no string  
 40703 identified by `msgid` exists in the messages object, or if an error occurs, `msgid` shall be returned.

40704 The `dgettext()` function shall be equivalent to `gettext()`, except `domainname` shall be used instead  
 40705 of the current text domain to locate the messages object.

40706 The `dcgettext()` function shall be equivalent to `dgettext()`, except the locale category identified by  
 40707 `category` shall be used instead of `LC_MESSAGES`.

40708 The `ngettext()` function shall be equivalent to `gettext()`, except:



- 40709           • The string to retrieve shall be identified by a combination of *msgid* and *n* (see *msgfmt*).
- 40710           • If the locale name in effect is "POSIX" or "C", or if no suitable messages object exists, or if
- 40711           no string identified by the combination of *msgid* and *n* exists in the messages object, or if an
- 40712           error occurs, the return value shall be *msgid* if *n* is 1, otherwise *msgid\_plural*.

40713           The *dngettext()* function shall be equivalent to *ngettext()*, except *domainname* shall be used

40714           instead of the current text domain to locate the messages object.

40715           The *dcngettext()* function shall be equivalent to *dngettext()*, except the locale category identified

40716           by *category* shall be used instead of *LC\_MESSAGES*.

40717           The *\*\_l()* functions shall be equivalent to their counterparts without the *\_l* suffix, except *locale*

40718           shall be used instead of the current locale. If *locale* is the special locale object

40719           LC\_GLOBAL\_LOCALE or is not a valid locale object handle, the behavior is undefined.

40720           The application shall ensure that the *msgid* and *msgid\_plural* arguments are strings. If either

40721           *msgid* or *msgid\_plural* is an empty string, or contains characters not in the portable character set,

40722           the results are unspecified. If the *category* argument is *LC\_ALL*, the results are unspecified.

40723           The location of the messages object shall be determined according to the following criteria,

40724           stopping when the first messages object is found:

- 40725 XSI           1. If the *NLSPATH* environment variable is set to a non-empty string, an *NLSPATH* search
- 40726           shall be performed as described in XBD Section 8.2 (on page 169). If *NLSPATH* identifies
- 40727           more than one template to use, each template in turn shall be used until a valid messages
- 40728           object is found.
- 40729           2. If the *LANGUAGE* environment variable is set to a non-empty string, a *LANGUAGE*
- 40730           search shall be performed as described below. If *LANGUAGE* identifies more than one
- 40731           directory to search, each directory shall be searched until a valid messages object is found.
- 40732           3. A single-locale search shall be performed as described below.

40733 XSI           For the *NLSPATH* search and the single-locale search, the single locale name used to locate the

40734           messages object shall be the locale name associated with the selected locale category from the

40735           current locale, or the provided locale object if calling one of the *\*\_l()* functions; additional

40736           searches of locale names without *.codeset* (if present), without *\_territory* (if present), and without

40737           *@modifier* (if present) may be performed.

40738           For the *LANGUAGE* search, the value of the *LANGUAGE* environment variable shall be a list of

40739           one or more locale names separated by a <colon> (' : ') character. Each locale name shall be

40740           tried in the specified order. If a messages object for the locale does not exist, or cannot be

40741           opened, or is unsuitable for implementation-defined reasons (such as security), the next locale

40742           name (if any) shall be tried. If:

- 40743           • a messages object for the locale can be opened but cannot be processed without error, or
- 40744           • the messages object does not contain a string identified by *msgid*, or *msgid* and *n* for the
- 40745           *ngettext* functions,

40746           it is unspecified whether the next locale name (if any) is tried. In all other cases, the messages

40747           object for the locale shall be used.

40748           For each locale name in *LANGUAGE*, or if *LANGUAGE* is not set or is empty, or no suitable

40749           messages object is found in processing *LANGUAGE*, the pathname used to locate the messages

40750           object shall be *dirname/localename/categoryname/textdomainname.mo*, where:

- 40751 • The *dirname* part is the *dirname* argument of the most recent successful call to  
40752 *bindtextdomain()* that had *textdomainname* as the *domainname* argument; any trailing <slash>  
40753 characters in *dirname* shall be discarded. If a successful call to *bindtextdomain()* has not  
40754 been made for *textdomainname*, an implementation-defined default directory shall be used.
- 40755 • For the *LANGUAGE* search, the *localename* part is each locale name from *LANGUAGE* in  
40756 turn; if a locale name has the format *language[\_territory][.codeset][@modifier]*, additional  
40757 searches of locale names without *.codeset* (if present), without *\_territory* (if present), and  
40758 without *@modifier* (if present) may be performed; if *.codeset* is not present, additional  
40759 searches of locale names with an added *.codeset* may be performed. For the single-locale  
40760 search, the *localename* part is the name of the current locale, or the locale specified in an  
40761 *\*\_l()* function call, for the category named by *categoryname*. Spellings of codeset names are  
40762 not standardized, and implementations may attempt to use different commonly known  
40763 spellings, for example "utf8" and "UTF-8".
- 40764 • The *categoryname* part is the string "LC\_MESSAGES" if *gettext()*, *dgettext()*, *ngettext()*, or  
40765 *dngettext()* is called, or the locale category name corresponding to the *category* argument to  
40766 *dcgettext()* or *dcngettext()*. Likewise for the *\*\_l()* variants of all these functions.
- 40767 • For *gettext()*, *gettext\_l()*, *ngettext()*, and *ngettext\_l()*, the *textdomainname* part is the text  
40768 domain set by the last successful call to *textdomain()*. For *dgettext()*, *dcgettext()*,  
40769 *dngettext()*, *dcngettext()*, and the *\*\_l()* variants of these functions, *textdomainname* is the text  
40770 domain specified by the *domainname* argument. The *domainname* argument shall be  
40771 equivalent in syntax and meaning to the *domainname* argument to *textdomain()*, except that  
40772 the selection of the text domain shall affect only the *dgettext()*, *dcgettext()*, *dngettext()*, and  
40773 *dcngettext()* function calls and their *\*\_l()* variants. If the *domainname* argument is a null  
40774 pointer, the text domain set by the last successful call to *textdomain()* shall be used. For all  
40775 of these functions, if a successful call to *textdomain()* has not been made the default text  
40776 domain "messages" shall be used.

40777 Resolution of the messages object pathname shall be performed the first time one of the *gettext*  
40778 family of functions is called for a given combination of *dirname*, *localename*, *categoryname*, and  
40779 *textdomainname*. It is unspecified whether the pathname is re-resolved if the combination has  
40780 been used before in a call to one of the *gettext* family of functions. If *bindtextdomain()* performs  
40781 pathname resolution of its *dirname* argument, only the part of the messages object pathname  
40782 after *dirname* shall be resolved by the *gettext* family of functions.

40783 When one of the *gettext* family of functions returns a message string that was found in a  
40784 messages object, it shall convert the codeset of the message string to the output codeset if a  
40785 codeset is specified in the messages object (see *msgfmt*) and the output codeset is not the same as  
40786 that codeset. If a successful call to *bind\_textdomain\_codeset()* has been made with the text domain  
40787 of the messages object as the *domainname* argument and a non-null *codeset* argument, the output  
40788 codeset shall be the *codeset* argument from the most recent such call. Otherwise, the output  
40789 codeset shall be the codeset of characters in the current locale, or the provided locale object if  
40790 calling one of the *\*\_l()* functions, as specified by the *LC\_CTYPE* category of the locale. The  
40791 conversion shall be performed as if by a call to *iconv()* using a conversion descriptor returned by  
40792 *iconv\_open(<output codeset>, <messages object codeset>)*, except that if the return value of *iconv()*  
40793 would be greater than zero, the non-identical conversions performed by the *gettext* family of  
40794 functions need not be the same as those that such an *iconv()* call would perform. If an error  
40795 prevents the codeset conversion from being performed, the *gettext* family of functions shall  
40796 behave as if no message string was found in the messages object. If at least one non-identical  
40797 conversion is performed that results in a fallback character (one that does not provide any  
40798 information about the character it was converted from, for example, a <question-mark> or  
40799 ``replacement-character''), the *gettext* family of functions may behave as if no message string was

40800 found in the messages object.

#### 40801 RETURN VALUE

40802 The `gettext()`, `gettext_l()`, `dgettext()`, `dgettext_l()`, `dcgettext()`, and `dcgettext_l()` functions shall  
40803 return the message string described in DESCRIPTION if successful. Otherwise, they shall return  
40804 `msgid`.

40805 The `ngettext()`, `ngettext_l()`, `dngettext()`, `dngettext_l()`, `dcngettext()`, and `dcngettext_l()` functions  
40806 shall return the message string described in DESCRIPTION if successful. Otherwise, `msgid` shall  
40807 be returned if `n` is equal to 1, or `msgid_plural` if `n` is not equal to 1.

40808 The application shall ensure that it does not modify the returned string. A subsequent call to a  
40809 `gettext` family function shall not overwrite or invalidate the returned string. The returned string  
40810 may be invalidated by a subsequent call to `bind_textdomain_codeset()`, `bindtextdomain()`,  
40811 `setlocale()`, or `textdomain()` in the same process, except for calls that only query values. The  
40812 returned string shall not be invalidated by a subsequent call to `uselocale()`.

#### 40813 ERRORS

40814 The `gettext` family of functions shall not modify `errno`. If an error occurs these functions shall  
40815 return a string as described in RETURN VALUE.

#### 40816 EXAMPLES

40817 The example code below assumes the following:

- 40818 • The implementation-defined default directory is `/system/gettextlib`.
- 40819 • The following locales are available on the target system: `en_US`, `en_GB`, `de_DE`. The  
40820 codeset used for all of these locales is UTF-8.
- 40821 • The `en_AU` locale is not available on the target system.
- 40822 • The target system supports conversion from ISO/IEC 8859-1 to UTF-8.
- 40823 • The codeset used for the POSIX locale is ASCII.
- 40824 • The target system does not support conversion from ISO/IEC 8859-1 to ASCII.

40825 Furthermore, the following `.mo` files (and only the following `.mo` files) are installed:

- 40826 • `/system/gettextlib/en_US/LC_MESSAGES/mail.mo`
- 40827 • `/messagecatalogs/example/en_US/LC_MESSAGES/mail.mo`

40828 These are compiled from a portable messages object source file (dot-po file) with the following  
40829 ISO/IEC 8859-1 encoded contents (see the EXTENDED DESCRIPTION of the `msgfmt` utility for a  
40830 description of the dot-po file format):

```
40831 msgid ""
40832 msgstr ""
40833 "Content-Type: text/plain; charset=ISO_8859-1\n"
40834 "Plural-Forms: nplurals=4; plural= n==1?0: (n>1&&n<10)?1: (n==0)?2:3;\n"
40835 msgid "recipient"
40836 msgid_plural "recipients"
40837 msgstr[0] "1 recipient"
40838 msgstr[1] "2 to 9 recipients"
40839 msgstr[2] "no recipients"
40840 msgstr[3] "more than 9 recipients"
```

40841 `/system/gettextlib/de_DE/LC_MESSAGES/mail.mo` is compiled from a dot-po file with the  
40842 following ISO/IEC 8859-1 encoded contents:

```

40843     msgid ""
40844     msgstr ""
40845     "Content-Type: text/plain; charset=ISO_8859-1\n"
40846     "Plural-Forms: nplurals=4; plural= n==1?0: (n>1&&n<5)?1: (n==0)?2:3;\n"
40847     msgid "recipient"
40848     msgid_plural "recipients"
40849     msgstr[0] "1 Empfänger"
40850     msgstr[1] "2 bis 4 Empfänger"
40851     msgstr[2] "keine Empfänger"
40852     msgstr[3] "mehr als 4 Empfänger"

40853     /messagecatalogs/example/en_GB/LC_MESSAGES/mail.mo is compiled from a dot-po file
40854     with the following ISO/IEC 8859-1 encoded contents:

40855     msgid ""
40856     msgstr ""
40857     "Content-Type: text/plain; charset=ISO_8859-1\n"
40858     "Plural-Forms: nplurals=4; plural= n==1?0: (n>1&&n<5)?1: (n==0)?2:3;\n"
40859     msgid "recipient"
40860     msgid_plural "recipients"
40861     msgstr[0] "1 recipient"
40862     msgstr[1] "2 to 4 recipients"
40863     msgstr[2] "no recipients"
40864     msgstr[3] "5 or more recipients"

40865     /messagecatalogs/example2/en_US/LC_MESSAGES/othermail.mo is not a suitable messages
40866     object file or is a suitable messages object file that does not contain the msgid "recipient".

40867     The following example demonstrates the interactions between bindtextdomain(),
40868     bind_textdomain_codeset(), textdomain(), and the gettext family of functions.

40869     unsigned long n_recipients;
40870     // strdup() is used to prevent default_domain from being invalidated by
40871     // a future call to bindtextdomain()
40872     const char *default_domain = strdup(bindtextdomain("mail", NULL));
40873     setlocale(LC_MESSAGES, "POSIX");
40874     setlocale(LC_CTYPE, "POSIX");

40875     n_recipients = 1;
40876     // The following outputs "recipient" with the same encoding as the
40877     // "recipient" argument to ngettext():
40878     printf("%s\n", ngettext("recipient", "recipients", n_recipients));

40879     n_recipients = 3;
40880     // The following outputs "recipients" with the same encoding as the
40881     // "recipients" argument to ngettext():
40882     printf("%s\n", ngettext("recipient", "recipients", n_recipients));

40883     setlocale(LC_MESSAGES, "en_US");
40884     setlocale(LC_CTYPE, "en_US");
40885     textdomain("mail");

40886     n_recipients = 1;
40887     // The following outputs "1 recipient", encoded in UTF-8:
40888     printf("%s\n", ngettext("recipient", "recipients", n_recipients));

40889     n_recipients = 3;

```

```
40890 // The following outputs "2 to 9 recipients", encoded in UTF-8:
40891 printf("%s\n", ngettext("recipient", "recipients", n_recipients));

40892 setlocale(LC_MESSAGES, "en_GB");
40893 setlocale(LC_CTYPE, "en_GB");
40894 bindtextdomain("mail", "/messagecatalogs/example/");

40895 n_recipients = 3;
40896 // The following outputs "2 to 4 recipients", encoded in UTF-8:
40897 printf("%s\n", ngettext("recipient", "recipients", n_recipients));

40898 setlocale(LC_MESSAGES, "en_US");
40899 setlocale(LC_CTYPE, "en_US");
40900 textdomain("othermail");
40901 bindtextdomain("othermail", "/messagecatalogs/example2/");

40902 n_recipients = 3;
40903 // The following outputs "recipients" with the same encoding as the
40904 // "recipients" argument to ngettext():
40905 printf("%s\n", ngettext("recipient", "recipients", n_recipients));

40906 // Because there is no locale named en_AU on the system, en_US is used:
40907 setenv("LANGUAGE", "en_AU:en_US:en_GB", 1);
40908 setlocale(LC_MESSAGES, "");
40909 setlocale(LC_CTYPE, "");
40910 bindtextdomain("mail", default_domain);

40911 // The following outputs "2 to 9 recipients", encoded in UTF-8:
40912 printf("%s\n", dngettext("mail", "recipient", "recipients", 3));

40913 textdomain("mail");
40914 bind_textdomain_codeset("mail", "UTF-8");
40915 setlocale(LC_MESSAGES, "de_DE");
40916 setlocale(LC_CTYPE, "de_DE");
40917 // Clear the LANGUAGE environment variable, otherwise it would take
40918 // precedence over the locale set above, and en_US would continue to
40919 // be used.
40920 setenv("LANGUAGE", "", 1);

40921 n_recipients = 1;
40922 // The following outputs "1 Empfänger", encoded in UTF-8:
40923 printf("%s\n", ngettext("recipient", "recipients", n_recipients));

40924 bind_textdomain_codeset("mail", "ASCII");
40925 setlocale(LC_CTYPE, "POSIX");

40926 n_recipients = 1;
40927 // The following outputs "recipient" with the same encoding as the
40928 // "recipient" argument to ngettext() - remember, the system is assumed
40929 // to not support conversion from ISO/IEC 8859-1 to ASCII:
40930 printf("%s\n", ngettext("recipient", "recipients", n_recipients));
40931 free(default_domain);
```

## 40932 APPLICATION USAGE

40933 These functions do not impose a limit on message length. Note that translated strings typically  
 40934 have a different length than the input strings, possibly much longer, and applications using  
 40935 these translations in formatted text (for example, aligned columns for a table) should take that  
 40936 into account.

40937 The *dcgettext()*, *dcgettext\_l()*, *dcngettext()*, and *dcngettext\_l()* functions are useful to retrieve  
 40938 locale-specific strings for a category other than *LC\_MESSAGES*. For example, they can be used  
 40939 to obtain a time format string from the *LC\_TIME* category; because the locale setting of *LC\_TIME*  
 40940 and *LC\_MESSAGES* can be different, using the other *gettext* family functions in such a case  
 40941 might cause an undesired result. All of the functions in the *gettext* family of functions, except  
 40942 *dcgettext()*, *dcgettext\_l()*, *dcngettext()*, and *dcngettext\_l()*, search for messages objects only in the  
 40943 *LC\_MESSAGES* category.

40944 Implementations typically, but are not required to, *mmap()* the messages object file the first time  
 40945 one of the *gettext* family of functions is called, and keep that map in place until it is no longer  
 40946 expected to be used. For example, a successful call to *bindtextdomain()* will typically cause the  
 40947 next call to one of the *gettext* family of functions to *munmap()* the previous file and *mmap()* the  
 40948 new file. Applications should not rely on this behavior, however: the implementation is allowed  
 40949 to cache previously used maps, or not use *mmap()* at all and reopen the file each time one of the  
 40950 *gettext* family of functions is called.

40951 The *msgid* and *msgid\_plural* arguments are typically in (US) English. The arguments are always  
 40952 used in the POSIX or C locale, and when a *gettext* family function encounters an error, so they  
 40953 should not be abstract message identifiers (for example, "message 123") and they should only  
 40954 use characters in the portable character set (to avoid outputting byte sequences that are not valid  
 40955 characters in the current output codeset). If the *xgettext* utility is used to extract the *msgid* and  
 40956 *msgid\_plural* arguments from C source files into a template dot-po file, the arguments must be  
 40957 string literals in order for the resulting file to be useful to translators.

40958 The strings returned by the *gettext* family of functions are not guaranteed to contain only  
 40959 characters that are valid in the current output codeset. In particular, byte sequences that do not  
 40960 form valid characters can occur when:

- 40961 • The *msgid* or *msgid\_plural* arguments use characters outside the portable character set.
- 40962 • The messages object file does not specify a character set and uses characters outside the  
 40963 portable character set.

40964 The strings returned by the *gettext* family of functions are guaranteed to remain valid until  
 40965 invalidated as described in the RETURN VALUE section. This includes strings that are created  
 40966 by codeset conversion; those strings are freed by the implementation, not the application. Thus,  
 40967 it is safe to call *gettext* family functions multiple times in situations such as:

```
40968 printf("%s %s\n", gettext("foo"), gettext("bar"));
```

## 40969 RATIONALE

40970 Although the return type of these functions ought to be **const char \***, it is **char \*** to match  
 40971 historical practice.

40972 The *gettext* family of functions is frequently used in reporting errors. In fact, it is possible to have  
 40973 an application that attempts to create an error message that combines a translated string via  
 40974 *gettext()* with an error string provided by *strerror()*. The standard requires that the *gettext* family  
 40975 of functions does not modify *errno*, so that an application need not worry about complications of  
 40976 providing sequencing points to capture a stable value of *errno* prior to the translation of the error  
 40977 message, and so that the user will still get a somewhat useful string (even if it is the untranslated  
 40978 original string) on any failure.

40979 There are no wide character equivalents for these functions; historically no implementation is  
40980 known to exist, and the multi-byte message returned from these functions can, in most instances,  
40981 be converted to wide characters by the application if desired.

40982 Some historical *gettext* implementations returned the translated string from the messages object  
40983 without codeset conversion if *iconv\_open()* fails. This is considered to be a bug in those  
40984 implementations.

#### 40985 **FUTURE DIRECTIONS**

40986 None.

#### 40987 **SEE ALSO**

40988 *bindtextdomain()*, *catopen()*, *iconv()*, *setlocale()*, *uselocale()*

40989 XBD [<libintl.h>](#), [<limits.h>](#)

40990 XCU *gettext*, *msgfmt*, *xgettext*

#### 40991 **CHANGE HISTORY**

40992 First released in Issue 8.

40993 **NAME**

40994       getuid — get a real user ID

40995 **SYNOPSIS**

40996       #include &lt;unistd.h&gt;

40997       uid\_t getuid(void);

40998 **DESCRIPTION**40999       The *getuid()* function shall return the real user ID of the calling process. The *getuid()* function  
41000       shall not modify *errno*.41001 **RETURN VALUE**41002       The *getuid()* function shall always be successful and no return value is reserved to indicate the  
41003       error.41004 **ERRORS**

41005       No errors are defined.

41006 **EXAMPLES**41007       **Setting the Effective User ID to the Real User ID**

41008       The following example sets the effective user ID of the calling process to the real user ID.

41009       #include &lt;unistd.h&gt;

41010       ...

41011       seteuid(getuid());

41012 **APPLICATION USAGE**

41013       None.

41014 **RATIONALE**41015       In a conforming environment, *getuid()* will always succeed. It is possible for implementations to  
41016       provide an extension where a process in a non-conforming environment will not be associated  
41017       with a user or group ID. It is recommended that such implementations return **(uid\_t)-1** and set  
41018       *errno* to indicate such an environment; doing so does not violate this standard, since such an  
41019       environment is already an extension.41020 **FUTURE DIRECTIONS**

41021       None.

41022 **SEE ALSO**41023       *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*,  
41024       *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*41025       XBD <[sys/types.h](#)>, <[unistd.h](#)>41026 **CHANGE HISTORY**

41027       First released in Issue 1. Derived from Issue 1 of the SVID.

41028 **Issue 6**41029       In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.41030       The following new requirements on POSIX implementations derive from alignment with the  
41031       Single UNIX Specification:

- 41032
- The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was  
41033       required for conforming implementations of previous POSIX specifications, it was not  
41034       required for UNIX applications.



41035 **Issue 7**

41036 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0175 [511] and XSH/TC2-2008/0176  
41037 [897] are applied.

41038 **Issue 8**

41039 Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to  
41040 SEE ALSO.

41041 **NAME**

41042           getutxent, getutxid, getutxline — get user accounting database entries

41043 **SYNOPSIS**

```
41044 XSI       #include <utmpx.h>
41045           struct utmpx *getutxent(void);
41046           struct utmpx *getutxid(const struct utmpx *id);
41047           struct utmpx *getutxline(const struct utmpx *line);
```

41048 **DESCRIPTION**

41049           Refer to *endutxent()*.

41050 **NAME**

41051 getwc — get a wide character from a stream

41052 **SYNOPSIS**

41053 #include &lt;stdio.h&gt;

41054 #include &lt;wchar.h&gt;

41055 wint\_t getwc(FILE \*stream);

41056 **DESCRIPTION**

41057 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
41058 conflict between the requirements described here and the ISO C standard is unintentional. This  
41059 volume of POSIX.1-2024 defers to the ISO C standard.

41060 The *getwc()* function shall be equivalent to *fgetwc()*, except that if it is implemented as a macro it  
41061 may evaluate *stream* more than once, so the argument should never be an expression with side-  
41062 effects.

41063 **RETURN VALUE**41064 Refer to *fgetwc()*.41065 **ERRORS**41066 Refer to *fgetwc()*.41067 **EXAMPLES**

41068 None.

41069 **APPLICATION USAGE**

41070 Since it may be implemented as a macro, *getwc()* may treat incorrectly a *stream* argument with  
41071 side-effects. In particular, *getwc(\*f++)* does not necessarily work as expected. Therefore, use of  
41072 this function is not recommended; *fgetwc()* should be used instead.

41073 **RATIONALE**

41074 None.

41075 **FUTURE DIRECTIONS**

41076 None.

41077 **SEE ALSO**41078 [Section 2.5](#) (on page 521), *fgetwc()*41079 XBD [<stdio.h>](#), [<wchar.h>](#)41080 **CHANGE HISTORY**

41081 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
41082 draft.

41083 **Issue 5**41084 The Optional Header (OH) marking is removed from [<stdio.h>](#).41085 **Issue 7**

41086 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0270 [14] is applied.

41087 **NAME**41088           getwchar — get a wide character from a *stdin* stream41089 **SYNOPSIS**

41090           #include &lt;wchar.h&gt;

41091           wint\_t getwchar(void);

41092 **DESCRIPTION**

41093 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
41094       conflict between the requirements described here and the ISO C standard is unintentional. This  
41095       volume of POSIX.1-2024 defers to the ISO C standard.

41096       The *getwchar()* function shall be equivalent to *getwc(stdin)*.41097 **RETURN VALUE**41098       Refer to *fgetwc()*.41099 **ERRORS**41100       Refer to *fgetwc()*.41101 **EXAMPLES**

41102       None.

41103 **APPLICATION USAGE**

41104       If the **wint\_t** value returned by *getwchar()* is stored into a variable of type **wchar\_t** and then  
41105       compared against the **wint\_t** macro WEOF, the result may be incorrect. Only the **wint\_t** type is  
41106       guaranteed to be able to represent any wide character and WEOF.

41107 **RATIONALE**

41108       None.

41109 **FUTURE DIRECTIONS**

41110       None.

41111 **SEE ALSO**41112       Section 2.5 (on page 521), *fgetwc()*, *getwc()*41113       XBD <**wchar.h**>41114 **CHANGE HISTORY**

41115       First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
41116       draft.

41117 **Issue 7**

41118       POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0271 [14] is applied.

41119 **NAME**

41120 glob, globfree — generate pathnames matching a pattern

41121 **SYNOPSIS**

```
41122 #include <glob.h>
41123 int glob(const char *restrict pattern, int flags,
41124         int(*errfunc)(const char *epath, int eerrno),
41125         glob_t *restrict pglob);
41126 void globfree(glob_t *pglob);
```

41127 **DESCRIPTION**

41128 The *glob()* function is a pathname generator that shall implement the rules defined in XCU  
 41129 [Section 2.14](#) (on page 2523), with optional support for rule 3 in XCU [Section 2.14.3](#) (on page  
 41130 2525).

41131 The structure type **glob\_t** is defined in **<glob.h>** and includes at least the following members:

Member Type	Member Name	Description
size_t	<i>gl_pathc</i>	Count of paths matched by <i>pattern</i> .
char **	<i>gl_pathv</i>	Pointer to a list of matched pathnames.
size_t	<i>gl_offs</i>	Slots to reserve at the beginning of <i>gl_pathv</i> .

41137 The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function  
 41138 shall match all accessible pathnames against this pattern and develop a list of all pathnames that  
 41139 match. In order to have access to a pathname, *glob()* requires search permission on every  
 41140 component of a path except the last, and read permission on each directory of any filename  
 41141 component of *pattern* that contains any of the following special characters: '\*', '?', and '['.

41142 The *glob()* function shall store the number of matched pathnames into *pglob->gl\_pathc* and a  
 41143 pointer to a list of pointers to pathnames into *pglob->gl\_pathv*. The pathnames shall be in sort  
 41144 order as defined by the current setting of the *LC\_COLLATE* category; see XBD [Section 7.3.2](#) (on  
 41145 page 139). The first pointer after the last pathname shall be a null pointer. If the pattern does not  
 41146 match any pathnames, the returned number of matched paths is set to 0, and the contents of  
 41147 *pglob->gl\_pathv* are implementation-defined.

41148 It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function  
 41149 shall allocate other space as needed, including the memory pointed to by *gl\_pathv*. The *globfree()*  
 41150 function shall free any space associated with *pglob* from a previous call to *glob()*. The *globfree()*  
 41151 function shall not modify *errno* if *pglob* was previously used by *glob()* and not yet freed.

41152 The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise-  
 41153 inclusive OR of zero or more of the following constants, which are defined in **<glob.h>**:

41154	<b>GLOB_APPEND</b>	Append pathnames generated to the ones from a previous call to <i>glob()</i> .
41155	<b>GLOB_DOOFFS</b>	Make use of <i>pglob-&gt;gl_offs</i> . If this flag is set, <i>pglob-&gt;gl_offs</i> is used to 41156 specify how many null pointers to add to the beginning of 41157 <i>pglob-&gt;gl_pathv</i> . In other words, <i>pglob-&gt;gl_pathv</i> shall point to 41158 <i>pglob-&gt;gl_offs</i> null pointers, followed by <i>pglob-&gt;gl_pathc</i> pathname 41159 pointers, followed by a null pointer.
41160	<b>GLOB_ERR</b>	Cause <i>glob()</i> to return when an attempt to open or search a pathname as a 41161 directory, or an attempt to read an opened directory, fails because of an 41162 error condition that is related to file system contents and prevents <i>glob()</i> 41163 from expanding the pattern. If this flag is not set, <i>glob()</i> shall not treat 41164 such conditions as an error, and shall continue to look for matches. Other

41165 error conditions may also be treated the same way as error conditions that  
41166 are related to file system contents.

41167 GLOB\_MARK For each pathname that matches *pattern* and is determined to be a  
41168 directory after pathname resolution, process the pathname so the result is  
41169 as if the following steps are applied in order:

- 41170 1. If the pathname is <slash>, do not modify the pathname and skip  
41171 the remaining steps.
- 41172 2. If the pathname is <slash><slash> and the implementation handles  
41173 pathname resolution of a pathname starting with exactly two  
41174 successive <slash> characters differently than it handles a  
41175 pathname starting with only a single <slash>, do not modify the  
41176 pathname and skip the remaining steps.
- 41177 3. If the pathname does not end with a <slash>, append a <slash> to  
41178 the pathname and skip the remaining steps.
- 41179 4. A <slash> may be appended to the pathname.
- 41180 5. If there are multiple <slash> characters at the end of the pathname,  
41181 all but one of those trailing <slash> characters may be removed  
41182 from the pathname.

41183 GLOB\_NOCHECK Supports rule 3 in XCU [Section 2.14.3](#) (on page 2525). If *pattern* does not  
41184 match any pathname, then *glob()* shall return a list consisting of only  
41185 *pattern*, and the number of matched pathnames is 1.

41186 GLOB\_NOESCAPE Disable backslash escaping.

41187 GLOB\_NOSORT Ordinarily, *glob()* sorts the matching pathnames according to the current  
41188 setting of the *LC\_COLLATE* category; see XBD [Section 7.3.2](#) (on page 139).  
41189 When this flag is used, the order of pathnames returned is unspecified.

41190 The GLOB\_APPEND flag can be used to append a new set of pathnames to those found in a  
41191 previous call to *glob()*. The following rules apply to applications when two or more calls to  
41192 *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:

- 41193 1. The first such call shall not set GLOB\_APPEND. All subsequent calls shall set it.
- 41194 2. All the calls shall set GLOB\_DOOFFS, or all shall not set it.
- 41195 3. After the second call, *pglob->gl\_pathv* points to a list containing the following:
  - 41196 a. Zero or more null pointers, as specified by GLOB\_DOOFFS and *pglob->gl\_offs*.
  - 41197 b. Pointers to the pathnames that were in the *pglob->gl\_pathv* list before the call, in  
41198 the same order as before.
  - 41199 c. Pointers to the new pathnames generated by the second call, in the specified order.
- 41200 4. The count returned in *pglob->gl\_pathc* shall be the total number of pathnames from the  
41201 two calls.
- 41202 5. The application can change any of the fields after a call to *glob()*. If it does, the  
41203 application shall reset them to the original value before a subsequent call, using the same  
41204 *pglob* value, to *globfree()* or *glob()* with the GLOB\_APPEND flag.

41205 If *errfunc* is not a null pointer and, during the search, an attempt to open or search a pathname as  
41206 a directory, or an attempt to read an opened directory, fails because of an error condition that

41207 prevents *glob()* from expanding the pattern, *glob()* calls (*\*errfunc()*) with two arguments:

- 41208 1. The *epath* argument is a pointer to the path that failed.
- 41209 2. The *eerrno* argument is the value of *errno* from the failure, as set by *opendir()*, *readdir()*, or
- 41210 *stat()*. (Other values may be used to report other errors not explicitly documented for
- 41211 those functions.)

41212 If (*\*errfunc()*) is called and returns non-zero, or, optionally, if *errfunc* is a null pointer and the

41213 attempt failed because of an error condition that is not related to file system contents, or if the

41214 GLOB\_ERR flag is set in *flags*, *glob()* shall stop the scan and return GLOB\_ABORTED after

41215 setting *gl\_pathc* and *gl\_pathv* in *pglob* to reflect the paths already scanned. If GLOB\_ERR is not set

41216 and either *errfunc* is a null pointer or (*\*errfunc()*) returns 0, the error shall be ignored.

41217 The set of error conditions that are considered to prevent *glob()* from expanding the pattern shall

41218 include [EACCES], [ENAMETOOLONG], and [ELOOP]. It is implementation-defined what

41219 other error conditions are included in the set.

41220 The *glob()* function shall not fail because of large files.

#### 41221 RETURN VALUE

41222 Upon successful completion, *glob()* shall return 0. The argument *pglob->gl\_pathc* shall return the

41223 number of matched pathnames and the argument *pglob->gl\_pathv* shall contain a pointer to a

41224 null-terminated list of matched and sorted pathnames. However, if *pglob->gl\_pathc* is 0, the

41225 content of *pglob->gl\_pathv* is undefined.

41226 The *globfree()* function shall not return a value.

41227 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in

41228 **<glob.h>**. The arguments *pglob->gl\_pathc* and *pglob->gl\_pathv* are still set as defined above.

#### 41229 ERRORS

41230 The *glob()* function shall fail and return the corresponding value if:

41231 GLOB\_ABORTED The scan was stopped because (*\*errfunc()*) was called and returned non-

41232 zero, or, optionally, *errfunc* was a null pointer and an attempt to open,

41233 read, or search a directory failed because of an error condition that is not

41234 related to file system contents, or GLOB\_ERR was set.

41235 GLOB\_NOMATCH The pattern does not match any existing pathname, and

41236 GLOB\_NOCHECK was not set in *flags*.

41237 GLOB\_NOSPACE An attempt to allocate memory failed.

#### 41238 EXAMPLES

41239 One use of the GLOB\_DOOFFS flag is by applications that build an argument list for use with

41240 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the

41241 equivalent of:

```
41242 ls -ld -- *.c
```

41243 but for some reason:

```
41244 system("ls -ld -- *.c")
```

41245 is not acceptable. The application could obtain the same result (except for error handling,

41246 omitted here for simplicity) using the sequence:

```
41247 globbuf.gl_offs = 3;
41248 glob("*.c", GLOB_DOOFFS|GLOB_NOCHECK, NULL, &globbuf);
41249 globbuf.gl_pathv[0] = "ls"; // to establish the initial arguments
```

```

41250     globbuf.gl_pathv[1] = "-ld"; // that sh -c "ls -ld --" would
41251     globbuf.gl_pathv[2] = "--"; // produce for both examples
41252     execvp("ls", &globbuf.gl_pathv[0]);

```

41253 Using the same example:

```

41254     ls -ld -- *.c *.h

```

41255 could be simulated using GLOB\_APPEND as follows:

```

41256     globbuf.gl_offs = 3;
41257     glob("*.c", GLOB_DOOFFS|GLOB_NOCHECK, NULL, &globbuf);
41258     glob("*.h", GLOB_DOOFFS|GLOB_NOCHECK|GLOB_APPEND, NULL, &globbuf);
41259     ...

```

## 41260 APPLICATION USAGE

41261 This function is not provided for the purpose of enabling utilities to perform pathname  
 41262 expansion on their arguments, as this operation is performed by the shell, and utilities are  
 41263 explicitly not expected to redo this. Instead, it is provided for applications that need to do  
 41264 pathname expansion on strings obtained from other sources, such as a pattern typed by a user or  
 41265 read from a file.

41266 If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

41267 Note that *gl\_pathc* and *gl\_pathv* have meaning even if *glob()* fails. This allows *glob()* to report  
 41268 partial results in the event of an error. However, if *gl\_pathc* is 0, *gl\_pathv* is unspecified even if  
 41269 *glob()* did not return an error.

41270 The GLOB\_NOCHECK option could be used when an application wants to expand a pathname  
 41271 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility  
 41272 might use this for option-arguments, for example.

41273 The new pathnames generated by a subsequent call with GLOB\_APPEND are not sorted  
 41274 together with the previous pathnames. This mirrors the way that the shell handles pathname  
 41275 expansion when multiple expansions are done on a command line.

41276 It is recommended that (*\*errfunc()*) should always return a non-zero value if the *eerrno*  
 41277 parameter indicates an error condition that is not related to file system contents. See XRAT  
 41278 [Section C.2.14.3](#) (on page 3911) for information about which error conditions are related to file  
 41279 system contents.

41280 Applications that need tilde and parameter expansion should use *wordexp()*.

## 41281 RATIONALE

41282 It was claimed that the GLOB\_DOOFFS flag is unnecessary because it could be simulated using:

```

41283     new = (char **)malloc((n + pglob->gl_pathc + 1)
41284         * sizeof(char *));
41285     (void) memcpy(new+n, pglob->gl_pathv,
41286         pglob->gl_pathc * sizeof(char *));
41287     (void) memset(new, 0, n * sizeof(char *));
41288     free(pglob->gl_pathv);
41289     pglob->gl_pathv = new;

```

41290 However, this assumes that the memory pointed to by *gl\_pathv* is a block that was separately  
 41291 created using *malloc()*. This is not necessarily the case. An application should make no  
 41292 assumptions about how the memory referenced by fields in *pglob* was allocated. It might have  
 41293 been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have  
 41294 been created using a different memory allocator. It is not the intent of the standard developers to



41295 specify or imply how the memory used by *glob()* is managed.

41296 The GLOB\_APPEND flag would be used when an application wants to expand several different  
41297 patterns into a single list.

41298 Earlier versions of this standard defined the behavior associated with the flag GLOB\_MARK as:  
41299 ``Each pathname that is a directory that matches *pattern* shall have a <slash> appended.'' This  
41300 was undesirable if the matched pathname was <slash> or if the matched pathname was  
41301 <slash><slash> and the implementation treats a leading <slash><slash> differently than it treats  
41302 a pathname with a single leading <slash>. Only a few implementations were known to conform  
41303 to this requirement (maybe only one) and there was a lot of variation in the way other  
41304 implementations behaved. The current wording allows many of the alternative behaviors that  
41305 were observed, except that the pathnames "/" and "/" (if it is treated differently than "/")  
41306 must not be modified.

41307 Implementations should consider the following much simpler requirement (which is allowed by  
41308 the current standard) when processing the GLOB\_MARK flag: ``Each pathname that matches  
41309 *pattern*, is determined to be a directory after pathname resolution, and does not end with a  
41310 <slash> shall have a <slash> appended.''

41311 Implementations differ as to which error conditions they consider prevent *glob()* from  
41312 expanding the pattern. The standard requires that [EACCES], [ENAMETOOLONG], and  
41313 [ELOOP] are included because in these cases the expansion could succeed if performed with a  
41314 different effective user or group ID, or with an alternative pathname (shorter than  
41315 {PATH\_MAX}, or traversing fewer symbolic links).

41316 Implementations are encouraged to call (*\*errfunc()*) for all error conditions that are related to file  
41317 system contents which occur when attempting to open or search a pathname as a directory or  
41318 attempting to read an opened directory. The appropriate way to handle such errors varies  
41319 according to the provenance of the pattern and what the application will do with the resulting  
41320 pathnames, and should therefore be for the application to decide. For example, given the pattern  
41321 "non-existing/\*", some applications may want *glob()* to succeed and return an empty list  
41322 because there are no existing files that match the pattern, but for others that would not be  
41323 appropriate, such as if an application asks the user to name a directory containing files to be  
41324 processed and the user makes a typing mistake when responding; the application will want to  
41325 alert the user to the mistake instead of behaving as if the user had named an empty directory. If  
41326 (*\*errfunc()*) is called for [ENOENT] errors, the first application can ignore them in that function,  
41327 but if (*\*errfunc()*) is not called, the second application cannot achieve what it wants using *glob()*.  
41328 Similar reasoning applies for the pattern "regular\_file/\*" and [ENOTDIR] errors.

#### 41329 FUTURE DIRECTIONS

41330 A future version of this standard may require that (*\*errfunc()*) is called for all error conditions  
41331 that are related to file system contents which occur when attempting to open or search a  
41332 pathname as a directory or attempting to read an opened directory.

#### 41333 SEE ALSO

41334 *exec*, *fdopendir()*, *fnmatch()*, *fstatat()*, *readdir()*, *wordexp()*

41335 XBD Section 7.3.2 (on page 139), <glob.h>

#### 41336 CHANGE HISTORY

41337 First released in Issue 4. Derived from the ISO POSIX-2 standard.

41338 **Issue 5**

41339 Moved from POSIX2 C-language Binding to BASE.

41340 **Issue 6**

41341 The normative text is updated to avoid use of the term “must” for application requirements.

41342 The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999  
41343 standard.

41344 **Issue 8**

41345 Austin Group Defect 385 is applied, adding a requirement that *globfree()* does not modify *errno*  
41346 when passed a pointer to a **glob\_t** that can be freed.

41347 Austin Group Defect 1255 is applied, changing the EXAMPLES section.

41348 Austin Group Defects 1273 and 1275 are applied, clarifying how errors are treated when  
41349 attempting to open or search a pathname as a directory or attempting to read an opened  
41350 directory.

41351 Austin Group Defect 1300 is applied, changing the description of GLOB\_MARK.

41352 Austin Group Defect 1444 is applied, correcting cross-references to *wordexp()*.

41353 **NAME**

41354 gmtime, gmtime\_r — convert a time value to a broken-down UTC time

41355 **SYNOPSIS**

```
41356 #include <time.h>
41357 struct tm *gmtime(const time_t *timer);
41358 CX struct tm *gmtime_r(const time_t *restrict timer,
41359 struct tm *restrict result);
```

41360 **DESCRIPTION**

41361 CX For `gmtime()`: The functionality described on this reference page is aligned with the ISO C  
 41362 standard. Any conflict between the requirements described here and the ISO C standard is  
 41363 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

41364 The `gmtime()` function shall convert the time in seconds since the Epoch pointed to by `timer` into  
 41365 a broken-down time, expressed as Coordinated Universal Time (UTC).

41366 CX The relationship between a time in seconds since the Epoch used as an argument to `gmtime()`  
 41367 and the `tm` structure (defined in the `<time.h>` header) is that the result shall be as specified in  
 41368 the expression given in the definition of seconds since the Epoch (see XBD Section 4.19, on page  
 41369 107), where the names in the structure and in the expression correspond.

41370 The same relationship shall apply for `gmtime_r()`.

41371 The `gmtime()` function need not be thread-safe; however, `gmtime()` shall avoid data races with all  
 41372 functions other than itself, `asctime()`, `ctime()`, and `localtime()`.

41373 The `asctime()`, `ctime()`, `gmtime()`, and `localtime()` functions shall return values in one of two static  
 41374 objects: a broken-down time structure and an array of type `char`. Execution of any of the  
 41375 functions that return a pointer to one of these object types may overwrite the information in any  
 41376 object of the same type pointed to by the value returned from any previous call to any of them.

41377 CX The `gmtime_r()` function shall convert the time in seconds since the Epoch pointed to by `timer`  
 41378 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down  
 41379 time is stored in the structure referred to by `result`. The `gmtime_r()` function shall also return the  
 41380 address of the same structure.

41381 **RETURN VALUE**

41382 Upon successful completion, the `gmtime()` function shall return a pointer to a `struct tm`. If an  
 41383 CX error is detected, `gmtime()` shall return a null pointer and set `errno` to indicate the error.

41384 Upon successful completion, `gmtime_r()` shall return the address of the structure pointed to by  
 41385 the argument `result`. The structure's `tm_zone` member shall be set to a pointer to the string  
 41386 "UTC", which shall have static storage duration. If an error is detected, `gmtime_r()` shall return a  
 41387 null pointer and set `errno` to indicate the error.

41388 **ERRORS**

41389 CX The `gmtime()` and `gmtime_r()` functions shall fail if:

41390 CX [EOVERFLOW] The result cannot be represented.

41391 **EXAMPLES**

41392 None.

41393 **APPLICATION USAGE**

41394 The *gmtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
41395 possibly using a static data area that may be overwritten by each call.

41396 **RATIONALE**

41397 None.

41398 **FUTURE DIRECTIONS**

41399 None.

41400 **SEE ALSO**41401 [asctime\(\)](#), [clock\(\)](#), [ctime\(\)](#), [difftime\(\)](#), [futimens\(\)](#), [localtime\(\)](#), [mktime\(\)](#), [strftime\(\)](#), [strptime\(\)](#), [time\(\)](#)41402 XBD Section 4.19 (on page 107), [<time.h>](#)41403 **CHANGE HISTORY**

41404 First released in Issue 1. Derived from Issue 1 of the SVID.

41405 **Issue 5**

41406 A note indicating that the *gmtime()* function need not be reentrant is added to the  
41407 DESCRIPTION.

41408 The *gmtime\_r()* function is included for alignment with the POSIX Threads Extension.41409 **Issue 6**41410 The *gmtime\_r()* function is marked as part of the Thread-Safe Functions option.

41411 Extensions beyond the ISO C standard are marked.

41412 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
41413 its avoidance of possibly using a static data area.

41414 The **restrict** keyword is added to the *gmtime\_r()* prototype for alignment with the  
41415 ISO/IEC 9899:1999 standard.

41416 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [Eoverflow]  
41417 error.

41418 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/48 is applied, updating the error handling  
41419 for *gmtime\_r()*.

41420 **Issue 7**

41421 Austin Group Interpretation 1003.1-2001 #156 is applied.

41422 The *gmtime\_r()* function is moved from the Thread-Safe Functions option to the Base.41423 **Issue 8**

41424 Austin Group Defect 1302 is applied, aligning the *gmtime()* function with the  
41425 ISO/IEC 9899:2018 standard.

41426 Austin Group Defect 1376 is applied, removing CX shading from some text derived from the  
41427 ISO C standard and updating it to match the ISO C standard.

41428 Austin Group Defect 1533 is applied, adding *tm\_gmtoff* and *tm\_zone* to the **tm** structure.

41429 **NAME**

41430 grantpt — grant access to the subsidiary pseudo-terminal device

41431 **SYNOPSIS**

```
41432 XSI #include <stdlib.h>  
41433 int grantpt(int fildev);
```

41434 **DESCRIPTION**

41435 The *grantpt()* function shall change the mode and ownership of the subsidiary pseudo-terminal  
41436 device associated with its manager pseudo-terminal counterpart. The *fildev* argument is a file  
41437 descriptor that refers to a manager pseudo-terminal device. The user ID of the subsidiary shall  
41438 be set to the real UID of the calling process and the group ID shall be set to an unspecified group  
41439 ID. The permission mode of the subsidiary pseudo-terminal shall be set to readable and writable  
41440 by the owner, and writable by the group.

41441 The behavior of the *grantpt()* function is unspecified if the application has installed a signal  
41442 handler to catch SIGCHLD signals.

41443 **RETURN VALUE**

41444 Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to  
41445 indicate the error.

41446 **ERRORS**

41447 The *grantpt()* function may fail if:

41448 [EACCES] The corresponding subsidiary pseudo-terminal device could not be accessed.

41449 [EBADF] The *fildev* argument is not a valid open file descriptor.

41450 [EINVAL] The *fildev* argument is not associated with a manager pseudo-terminal device.

41451 **EXAMPLES**

41452 None.

41453 **APPLICATION USAGE**

41454 None.

41455 **RATIONALE**

41456 See the RATIONALE section for *posix\_openpt()*.

41457 **FUTURE DIRECTIONS**

41458 None.

41459 **SEE ALSO**

41460 *open()*, *posix\_openpt()*, *ptsname()*, *unlockpt()*

41461 XBD <stdlib.h>

41462 **CHANGE HISTORY**

41463 First released in Issue 4, Version 2.

41464 **Issue 5**

41465 Moved from X/OPEN UNIX extension to BASE.

41466 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

41467 **Issue 7**

41468 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0272 [96] is applied.

41469 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0177 [506] is applied.

41470 **Issue 8**41471 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal  
41472 devices.

41473 **NAME**

41474 hcreate, hdestroy, hsearch — manage hash search table

41475 **SYNOPSIS**

```
41476 XSI #include <search.h>
41477 int hcreate(size_t nel);
41478 void hdestroy(void);
41479 ENTRY *hsearch(ENTRY item, ACTION action);
```

41480 **DESCRIPTION**41481 The *hcreate()*, *hdestroy()*, and *hsearch()* functions shall manage hash search tables.

41482 The *hcreate()* function shall allocate sufficient space for the table, and the application shall ensure it is called before *hsearch()* is used. The *nel* argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

41486 The *hdestroy()* function shall dispose of the search table, and may be followed by another call to *hcreate()*. After the call to *hdestroy()*, the data can no longer be considered accessible.

41488 The *hsearch()* function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The *item* argument is a structure of type **ENTRY** (defined in the **<search.h>** header) containing two pointers: *item.key* points to the comparison key (a **char \***), and *item.data* (a **void \***) points to any other data to be associated with that key. The comparison function used by *hsearch()* is *strcmp()*. The *action* argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

41497 These functions need not be thread-safe.

41498 **RETURN VALUE**41499 The *hcreate()* function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.41501 The *hdestroy()* function shall not return a value.41502 The *hsearch()* function shall return a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.41504 **ERRORS**41505 The *hcreate()* and *hsearch()* functions may fail if:

41506 [ENOMEM] Insufficient storage space is available.

41507 **EXAMPLES**

41508 The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```
41511 #include <stdio.h>
41512 #include <search.h>
41513 #include <string.h>
41514 struct info { /* This is the info stored in the table */
41515     int age, room; /* other than the key. */
41516 };
```

```

41517     #define NUM_EMPL      5000    /* # of elements in search table. */
41518     int main(void)
41519     {
41520         char string_space[NUM_EMPL*20]; /* Space to store strings. */
41521         struct info info_space[NUM_EMPL]; /* Space to store employee info. */
41522         char *str_ptr = string_space; /* Next space in string_space. */
41523         struct info *info_ptr = info_space;
41524                                     /* Next space in info_space. */
41525         ENTRY item;
41526         ENTRY *found_item; /* Name to look for in table. */
41527         char name_to_find[30];
41528
41529         int i = 0;
41529         /* Create table; no error checking is performed. */
41530         (void) hcreate(NUM_EMPL);
41531         while (scanf("%s%d%d", str_ptr, &info_ptr->age,
41532                    &info_ptr->room) != EOF && i++ < NUM_EMPL) {
41533             /* Put information in structure, and structure in item. */
41534             item.key = str_ptr;
41535             item.data = info_ptr;
41536             str_ptr += strlen(str_ptr) + 1;
41537             info_ptr++;
41538
41539             /* Put item into table. */
41539             (void) hsearch(item, ENTER);
41540         }
41541
41541         /* Access table. */
41542         item.key = name_to_find;
41543         while (scanf("%s", item.key) != EOF) {
41544             if ((found_item = hsearch(item, FIND)) != NULL) {
41545                 /* If item is in the table. */
41546                 (void)printf("found %s, age = %d, room = %d\n",
41547                             found_item->key,
41548                             ((struct info *)found_item->data)->age,
41549                             ((struct info *)found_item->data)->room);
41550             } else
41551                 (void)printf("no such employee %s\n", name_to_find);
41552         }
41553         return 0;
41554     }

```

**APPLICATION USAGE**

The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.



41561 **SEE ALSO**41562 *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tdelete()*41563 XBD <[search.h](#)>41564 **CHANGE HISTORY**

41565 First released in Issue 1. Derived from Issue 1 of the SVID.

41566 **Issue 6**

41567 The normative text is updated to avoid use of the term “must” for application requirements.

41568 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

41569 **Issue 7**

41570 Austin Group Interpretation 1003.1-2001 #156 is applied.

41571 **NAME**

41572           htobe16, htobe32, htobe64, htole16, htole32, htole64 — convert values between host and  
41573           specified byte order

41574 **SYNOPSIS**

```
41575       #include <endian.h>  
41576       uint16_t htobe16(uint16_t host_16bits);  
41577       uint32_t htobe32(uint32_t host_32bits);  
41578       uint64_t htobe64(uint64_t host_64bits);  
41579       uint16_t htole16(uint16_t host_16bits);  
41580       uint32_t htole32(uint32_t host_32bits);  
41581       uint64_t htole64(uint64_t host_64bits);
```

41582 **DESCRIPTION**

41583       Refer to [be16toh\(\)](#).

41584 **NAME**

41585 htonl, htons, ntohl, ntohs — convert values between host and network byte order

41586 **SYNOPSIS**

```
41587 #include <arpa/inet.h>
41588 uint32_t htonl(uint32_t hostlong);
41589 uint16_t htons(uint16_t hostshort);
41590 uint32_t ntohl(uint32_t netlong);
41591 uint16_t ntohs(uint16_t netshort);
```

41592 **DESCRIPTION**

41593 These functions shall convert 16-bit and 32-bit quantities between network byte order and host  
41594 byte order.

41595 On some implementations, these functions are defined as macros.

41596 The `uint32_t` and `uint16_t` types are defined in `<inttypes.h>`.

41597 **RETURN VALUE**

41598 The `htonl()` and `htons()` functions shall return the argument value converted from host to  
41599 network byte order.

41600 The `ntohl()` and `ntohs()` functions shall return the argument value converted from network to  
41601 host byte order.

41602 **ERRORS**

41603 No errors are defined.

41604 **EXAMPLES**

41605 None.

41606 **APPLICATION USAGE**

41607 These functions are most often used in conjunction with IPv4 addresses and ports as returned by  
41608 `gethostent()` and `getservent()`.

41609 **RATIONALE**

41610 None.

41611 **FUTURE DIRECTIONS**

41612 None.

41613 **SEE ALSO**

41614 [be16toh\(\)](#), [endhostent\(\)](#), [endservent\(\)](#)  
41615 XBD [<arpa/inet.h>](#), [<endian.h>](#), [<inttypes.h>](#)

41616 **CHANGE HISTORY**

41617 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

41618 **Issue 8**

41619 Austin Group Defect 162 is applied, adding `be16toh()` and `<endian.h>` to the SEE ALSO section.

41620 **NAME**

41621 hypot, hypotf, hypotl — Euclidean distance function

41622 **SYNOPSIS**

41623 #include &lt;math.h&gt;

41624 double hypot(double *x*, double *y*);41625 float hypotf(float *x*, float *y*);41626 long double hypotl(long double *x*, long double *y*);41627 **DESCRIPTION**

41628 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 41629 conflict between the requirements described here and the ISO C standard is unintentional. This  
 41630 volume of POSIX.1-2024 defers to the ISO C standard.

41631 These functions shall compute the value of the square root of  $x^2+y^2$  without undue overflow or  
 41632 underflow.

41633 An application wishing to check for error situations should set *errno* to zero and call  
 41634 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 41635 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 41636 zero, an error has occurred.

41637 **RETURN VALUE**

41638 Upon successful completion, these functions shall return the length of the hypotenuse of a right-  
 41639 angled triangle with sides of length *x* and *y*.

41640 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and  
 41641 *hypotl()* shall return the value of the macro HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL,  
 41642 respectively.

41643 MX If *x* or *y* is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned (even if one of *x* or *y* is NaN).

41644 If *x* or *y* is NaN, and the other is not  $\pm\text{Inf}$ , a NaN shall be returned.

41645 MXX If both arguments are subnormal and the correct result is subnormal, a range error may occur  
 41646 and the correct result shall be returned.

41647 **ERRORS**

41648 These functions shall fail if:

41649 Range Error The result overflows.

41650 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41651 then *errno* shall be set to [ERANGE]. If the integer expression  
 41652 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 41653 floating-point exception shall be raised.

41654 These functions may fail if:

41655 MX Range Error The result underflows.

41656 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 41657 then *errno* shall be set to [ERANGE]. If the integer expression  
 41658 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 41659 floating-point exception shall be raised.

41660 **EXAMPLES**41661 See the EXAMPLES section in *atan2()*.41662 **APPLICATION USAGE**41663 *hypot(x,y)*, *hypot(y,x)*, and *hypot(x,-y)* are equivalent.41664 *hypot(x, ±0)* is equivalent to *fabs(x)*.41665 Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also subnormal.

41667 These functions take precautions against overflow during intermediate steps of the computation.

41668 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* & MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.41670 **RATIONALE**

41671 None.

41672 **FUTURE DIRECTIONS**

41673 None.

41674 **SEE ALSO**41675 *atan2()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*

41676 XBD Section 4.23 (on page 109), &lt;math.h&gt;

41677 **CHANGE HISTORY**

41678 First released in Issue 1. Derived from Issue 1 of the SVID.

41679 **Issue 5**

41680 The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

41682 **Issue 6**41683 The *hypot()* function is no longer marked as an extension.41684 The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

41686 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

41688 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

41690 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/49 is applied, updating the EXAMPLES section.

41692 **Issue 7**

41693 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0273 [68] is applied.

41694 **NAME**

41695 iconv — codeset conversion function

41696 **SYNOPSIS**

```
41697 #include <iconv.h>
41698 size_t iconv(iconv_t cd, char **restrict inbuf,
41699             size_t *restrict inbytesleft, char **restrict outbuf,
41700             size_t *restrict outbytesleft);
```

41701 **DESCRIPTION**

41702 The *iconv()* function shall convert the sequence of characters from one codeset, in the array  
 41703 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array  
 41704 specified by *outbuf*. The codesets are those specified in the *iconv\_open()* call that returned the  
 41705 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first  
 41706 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer  
 41707 to be converted. The *outbuf* argument points to a variable that points to the first available byte in  
 41708 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the  
 41709 buffer.

41710 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by  
 41711 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv()* is  
 41712 called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*  
 41713 points to a positive value, *iconv()* shall place, into the output buffer, the byte sequence to change  
 41714 the output buffer to its initial shift state. If the output buffer is not large enough to hold the  
 41715 entire reset sequence, *iconv()* shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as  
 41716 other than a null pointer or a pointer to a null pointer cause the conversion to take place from  
 41717 the current state of the conversion descriptor.

41718 If a sequence of input bytes does not form a valid character or shift sequence in the input  
 41719 codeset:

- 41720 • If the //IGNORE indicator suffix was specified when the conversion descriptor *cd* was  
 41721 opened and the byte sequence is immediately followed by a valid character or shift  
 41722 sequence, the sequence of bytes shall be discarded and conversion shall continue from the  
 41723 immediately following valid character or shift sequence. This shall not be treated as an  
 41724 error.
- 41725 • If the //IGNORE indicator suffix was not specified when the conversion descriptor *cd* was  
 41726 opened, conversion shall stop after the previous successfully converted character or shift  
 41727 sequence.

41728 If the input buffer ends with an incomplete character or shift sequence, conversion shall stop  
 41729 after the previous successfully converted bytes. If the output buffer is not large enough to hold  
 41730 the entire converted input, conversion shall stop just prior to the input bytes that would cause  
 41731 the output buffer to overflow. The variable pointed to by *inbuf* shall be updated to point to the  
 41732 byte following the last byte successfully used in the conversion. The value pointed to by  
 41733 *inbytesleft* shall be decremented to reflect the number of bytes still not converted in the input  
 41734 buffer. The variable pointed to by *outbuf* shall be updated to point to the byte following the last  
 41735 byte of converted output data. The value pointed to by *outbytesleft* shall be decremented to  
 41736 reflect the number of bytes still available in the output buffer. For state-dependent encodings,  
 41737 the conversion descriptor shall be updated to reflect the shift state in effect at the end of the last  
 41738 successfully converted byte sequence.

41739 If *iconv()* encounters a character in the input buffer that is valid, but for which an identical  
 41740 character does not exist in the output codeset:

- 41741 • If either the `//IGNORE` or the `//NON_IDENTICAL_DISCARD` indicator suffix was
- 41742 specified when the conversion descriptor `cd` was opened, the character shall be discarded
- 41743 but shall still be counted in the return value of the `iconv()` call.
  
- 41744 • If the `//TRANSLIT` indicator suffix was specified when the conversion descriptor `cd` was
- 41745 opened, an implementation-defined transliteration shall be performed, if possible, to
- 41746 convert the character into one or more characters of the output codeset that best resemble
- 41747 the input character. The character shall be counted as one character in the return value of
- 41748 the `iconv()` call, regardless of the number of output characters.
  
- 41749 • If no indicator suffix was specified when the conversion descriptor `cd` was opened, or the
- 41750 `//TRANSLIT` indicator suffix was specified but no transliteration of the character is
- 41751 possible, `iconv()` shall perform an implementation-defined conversion on the character and
- 41752 it shall be counted in the return value of the `iconv()` call.

#### 41753 RETURN VALUE

41754 The `iconv()` function shall update the variables pointed to by the arguments to reflect the extent  
 41755 of the conversion and shall return the number of input characters that could not be converted to  
 41756 an identical output character. If the entire string in the input buffer is converted, except for any  
 41757 byte sequences discarded as a result of the `//IGNORE` indicator suffix, the value pointed to by  
 41758 `inbytesleft` shall be 0. If the input conversion is stopped due to any conditions mentioned above,  
 41759 the value pointed to by `inbytesleft` shall be non-zero and `errno` shall be set to indicate the  
 41760 condition. If an error occurs, `iconv()` shall return `(size_t)-1` and set `errno` to indicate the error.

#### 41761 ERRORS

41762 The `iconv()` function shall fail if:

- |       |          |  |
|-------|----------|--|
| 41763 | [EILSEQ] | Input conversion stopped due to an input byte that does not belong to the    |
| 41764 |          | input codeset.   |
| 41765 | [E2BIG]  | Input conversion stopped due to lack of space in the output buffer.          |
| 41766 | [EINVAL] | Input conversion stopped due to an incomplete character or shift sequence at |
| 41767 |          | the end of the input buffer.   |

41768 The `iconv()` function may fail if:

- |       |         |   |
|-------|---------|---|
| 41769 | [EBADF] | The <code>cd</code> argument is not a valid open conversion descriptor. |
|-------|---------|---|

#### 41770 EXAMPLES

41771 None.

#### 41772 APPLICATION USAGE

41773 The `inbuf` argument indirectly points to the memory area which contains the conversion input  
 41774 data. The `outbuf` argument indirectly points to the memory area which is to contain the result of  
 41775 the conversion. The objects indirectly pointed to by `inbuf` and `outbuf` are not restricted to  
 41776 containing data that is directly representable in the ISO C standard language `char` data type. The  
 41777 type of `inbuf` and `outbuf`, `char **`, does not imply that the objects pointed to are interpreted as  
 41778 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that  
 41779 represents a character in a given character set encoding scheme is done internally within the  
 41780 codeset converters. For example, the area pointed to indirectly by `inbuf` and/or `outbuf` can  
 41781 contain all zero octets that are not interpreted as string terminators but as coded character data  
 41782 according to the respective codeset encoding scheme. The type of the data (`char`, `short`, `long`, and  
 41783 so on) read or stored in the objects is not specified, but may be inferred for both the input and  
 41784 output data by the converters determined by the `fromcode` and `toencode` arguments of `iconv_open()`.

41785 Regardless of the data type inferred by the converter, the size of the remaining space in both  
 41786 input and output objects (the `inbytesleft` and `outbytesleft` arguments) is always measured in bytes.

41787 For implementations that support the conversion of state-dependent encodings, the conversion  
 41788 descriptor must be able to accurately reflect the shift-state in effect at the end of the last  
 41789 successful conversion. It is not required that the conversion descriptor itself be updated, which  
 41790 would require it to be a pointer type. Thus, implementations are free to implement the  
 41791 descriptor as a handle (other than a pointer type) by which the conversion information can be  
 41792 accessed and updated.

41793 It is the responsibility of the application to ensure that, if the output codeset has a locking-shift  
 41794 encoding, the output buffer is returned to its initial shift state when conversion is completed.  
 41795 This can be accomplished by calling *iconv()* with *inbuf* as a null pointer, or with *inbuf* pointing to  
 41796 a null pointer, before calling *iconv\_close()*. Since the standard does not provide a way to query  
 41797 whether a codeset has a locking-shift encoding, it is recommended that applications always call  
 41798 *iconv()* in this way before calling *iconv\_close()*.

41799 When the //IGNORE indicator suffix was used to open the conversion descriptor, *iconv()* does  
 41800 not provide any indication of whether any invalid input byte sequences were discarded.  
 41801 Applications which need to detect the discarding of invalid input byte sequences can open the  
 41802 conversion descriptor without using //IGNORE and then call *iconv()* in a loop such that if it  
 41803 returns an [EILSEQ] error, the application increments the variable pointed to by *inbuf* and  
 41804 decrements the variable pointed to by *inbytesleft* before the next call. This technique can also be  
 41805 used by applications which need to use //TRANSLIT but also discard invalid input byte  
 41806 sequences.

#### 41807 RATIONALE

41808 None.

#### 41809 FUTURE DIRECTIONS

41810 None.

#### 41811 SEE ALSO

41812 [\*iconv\\_open\(\)\*](#), [\*iconv\\_close\(\)\*](#), [\*mbsrtowcs\(\)\*](#)

41813 XBD [\*\*<iconv.h>\*\*](#)

#### 41814 CHANGE HISTORY

41815 First released in Issue 4. Derived from the HP-UX Manual.

#### 41816 Issue 6

41817 The SYNOPSIS has been corrected to align with the [\*\*<iconv.h>\*\*](#) reference page.

41818 The **restrict** keyword is added to the *iconv()* prototype for alignment with the  
 41819 ISO/IEC 9899:1999 standard.

#### 41820 Issue 7

41821 The *iconv()* function is moved from the XSI option to the Base.

#### 41822 Issue 8

41823 Austin Group Defect 1007 is applied, adding support for indicator suffixes in the *to*  
 41824 *code* argument to *iconv\_open()*.

41825 Austin Group Defect 1008 is applied, adding a paragraph about locking-shift encodings to the  
 41826 APPLICATION USAGE section.

41827 Austin Group Defect 1438 is applied, changing “valid character in the specified codeset” to  
 41828 “valid character in the specified input codeset”.



41829 **NAME**

41830 iconv\_close — codeset conversion deallocation function

41831 **SYNOPSIS**

41832 #include &lt;iconv.h&gt;

41833 int iconv\_close(iconv\_t *cd*);41834 **DESCRIPTION**41835 The *iconv\_close()* function shall deallocate the conversion descriptor *cd* and all other associated  
41836 resources allocated by *iconv\_open()*.41837 If a file descriptor is used to implement the type **iconv\_t**, that file descriptor shall be closed.41838 **RETURN VALUE**41839 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
41840 indicate the error.41841 **ERRORS**41842 The *iconv\_close()* function may fail if:

41843 [EBADF] The conversion descriptor is invalid.

41844 **EXAMPLES**

41845 None.

41846 **APPLICATION USAGE**

41847 None.

41848 **RATIONALE**

41849 None.

41850 **FUTURE DIRECTIONS**

41851 None.

41852 **SEE ALSO**41853 *iconv()*, *iconv\_open()*41854 XBD <**iconv.h**>41855 **CHANGE HISTORY**

41856 First released in Issue 4. Derived from the HP-UX Manual.

41857 **Issue 7**41858 The *iconv\_close()* function is moved from the XSI option to the Base.

41859 **NAME**

41860 iconv\_open — codeset conversion allocation function

41861 **SYNOPSIS**

41862 #include &lt;iconv.h&gt;

41863 iconv\_t iconv\_open(const char \*tocode, const char \*fromcode);

41864 **DESCRIPTION**

41865 The *iconv\_open()* function shall return a conversion descriptor that describes a conversion from  
 41866 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified  
 41867 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion  
 41868 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with  
 41869 *iconv()*.

41870 The codeset names that can be specified in *fromcode* and *tocode* and their permitted combinations  
 41871 are implementation-defined. Any one of the following indicator suffixes can be appended to the  
 41872 codeset name in *tocode*:

41873 //IGNORE Discard input bytes that do not form a valid character or shift sequence, and  
 41874 discard input characters for which an identical character does not exist in the  
 41875 output codeset.

41876 //NON\_IDENTICAL\_DISCARD  
 41877 Discard input characters for which an identical character does not exist in the  
 41878 output codeset.

41879 //TRANSLIT Transliterate input characters for which an identical character does not exist in  
 41880 the output codeset into one or more characters of the output codeset that best  
 41881 resemble the input character.

41882 See the description of *iconv()* for details of how these indicator suffixes alter the conversion  
 41883 performed by *iconv()*. Additional implementation-defined indicator suffixes may be supported.

41884 A conversion descriptor shall remain valid until it is closed by *iconv\_close()* or an implicit close.

41885 If a file descriptor is used to implement conversion descriptors, the FD\_CLOEXEC flag shall be  
 41886 set; see <fcntl.h>.

41887 **RETURN VALUE**

41888 Upon successful completion, *iconv\_open()* shall return a conversion descriptor for use on  
 41889 subsequent calls to *iconv()*. Otherwise, *iconv\_open()* shall return (**iconv\_t**)-1 and set *errno* to  
 41890 indicate the error.

41891 **ERRORS**

41892 The *iconv\_open()* function may fail if:

41893 [EMFILE] All file descriptors available to the process are currently open.

41894 [ENFILE] Too many files are currently open in the system.

41895 [ENOMEM] Insufficient storage space is available.

41896 [EINVAL] Conversion from the codeset specified in *fromcode* to the codeset specified in  
 41897 *tocode* is not supported by the implementation, or the codeset name in *tocode* is  
 41898 followed by an indicator suffix that is unrecognized or not supported.

**41899 EXAMPLES**

41900 None.

**41901 APPLICATION USAGE**

41902 Some implementations of *iconv\_open()* use *malloc()* to allocate space for internal buffer areas.  
41903 The *iconv\_open()* function may fail if there is insufficient storage space to accommodate these  
41904 buffers.

41905 Conforming applications must assume that conversion descriptors are not valid after a call to  
41906 one of the *exec* functions.

41907 Application developers should consult the system documentation to determine the supported  
41908 codesets and their naming schemes.

41909 Some implementations of *iconv\_open()* allow appending multiple indicator suffixes to the  
41910 codeset name in *toctype*, and some allow appending an indicator suffix (or suffixes) in both  
41911 *fromcode* and *toctype*. Portable applications should append at most one indicator suffix, and  
41912 append it only in *toctype*.

**41913 RATIONALE**

41914 None.

**41915 FUTURE DIRECTIONS**

41916 None.

**41917 SEE ALSO**

41918 *iconv()*, *iconv\_close()*

41919 XBD <[fcntl.h](#)>, <[iconv.h](#)>

**41920 CHANGE HISTORY**

41921 First released in Issue 4. Derived from the HP-UX Manual.

**41922 Issue 7**

41923 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

41924 The *iconv\_open()* function is moved from the XSI option to the Base.

**41925 Issue 8**

41926 Austin Group Defect 1007 is applied, adding support for indicator suffixes in the *toctype*  
41927 argument to *iconv\_open()*.

41928 **NAME**

41929 if\_freenameindex — free memory allocated by if\_nameindex

41930 **SYNOPSIS**

41931 #include &lt;net/if.h&gt;

41932 void if\_freenameindex(struct if\_nameindex \*ptr);

41933 **DESCRIPTION**

41934 The *if\_freenameindex()* function shall free the memory allocated by *if\_nameindex()*. The *ptr*  
41935 argument shall be a pointer that was returned by *if\_nameindex()*. After *if\_freenameindex()* has  
41936 been called, the application shall not use the array of which *ptr* is the address. The  
41937 *if\_freenameindex()* function shall not modify *errno* if *ptr* was previously returned by  
41938 *if\_nameindex()* and not yet freed.

41939 **RETURN VALUE**

41940 None.

41941 **ERRORS**

41942 No errors are defined.

41943 **EXAMPLES**

41944 None.

41945 **APPLICATION USAGE**

41946 None.

41947 **RATIONALE**

41948 None.

41949 **FUTURE DIRECTIONS**

41950 None.

41951 **SEE ALSO**41952 [\*getsockopt\(\)\*](#), [\*if\\_indextoname\(\)\*](#), [\*if\\_nameindex\(\)\*](#), [\*if\\_nametoindex\(\)\*](#), [\*setsockopt\(\)\*](#)41953 XBD [\*\*<net/if.h>\*\*](#)41954 **CHANGE HISTORY**

41955 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

41956 **Issue 8**

41957 Austin Group Defect 385 is applied, adding a requirement that *if\_freenameindex()* does not  
41958 modify *errno* when passed a pointer to an **if\_nameindex** structure than can be freed.

**41959 NAME**

41960 if\_indextoname — map a network interface index to its corresponding name

**41961 SYNOPSIS**

41962 #include <net/if.h>

41963 char \*if\_indextoname(unsigned *ifindex*, char \**ifname*);

**41964 DESCRIPTION**

41965 The *if\_indextoname()* function shall map an interface index to its corresponding name.

41966 When this function is called, *ifname* shall point to a buffer of at least {IF\_NAMESIZE} bytes. The  
41967 function shall place in this buffer the name of the interface with index *ifindex*.

**41968 RETURN VALUE**

41969 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which  
41970 points to a buffer now containing the interface name. Otherwise, the function shall return a null  
41971 pointer and set *errno* to indicate the error.

**41972 ERRORS**

41973 The *if\_indextoname()* function shall fail if:

41974 [ENXIO] The interface does not exist.

**41975 EXAMPLES**

41976 None.

**41977 APPLICATION USAGE**

41978 None.

**41979 RATIONALE**

41980 None.

**41981 FUTURE DIRECTIONS**

41982 None.

**41983 SEE ALSO**

41984 [getsockopt\(\)](#), [if\\_freenameindex\(\)](#), [if\\_nameindex\(\)](#), [if\\_nametoindex\(\)](#), [setsockopt\(\)](#)

41985 XBD [<net/if.h>](#)

**41986 CHANGE HISTORY**

41987 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

41988 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to  
41989 {IF\_NAMESIZ} in the DESCRIPTION.

41990 **NAME**

41991 if\_nameindex — return all network interface names and indexes

41992 **SYNOPSIS**

41993 #include <net/if.h>

41994 struct if\_nameindex \*if\_nameindex(void);

41995 **DESCRIPTION**

41996 The *if\_nameindex()* function shall return an array of *if\_nameindex* structures, one structure per  
41997 interface. The end of the array is indicated by a structure with an *if\_index* field of zero and an  
41998 *if\_name* field of NULL.

41999 Applications should call *if\_freenameindex()* to release the memory that may be dynamically  
42000 allocated by this function, after they have finished using it.

42001 **RETURN VALUE**

42002 An array of structures identifying local interfaces. A null pointer is returned upon an error, with  
42003 *errno* set to indicate the error.

42004 **ERRORS**

42005 The *if\_nameindex()* function may fail if:

42006 [ENOBUFFS] Insufficient resources are available to complete the function.

42007 **EXAMPLES**

42008 None.

42009 **APPLICATION USAGE**

42010 None.

42011 **RATIONALE**

42012 None.

42013 **FUTURE DIRECTIONS**

42014 None.

42015 **SEE ALSO**

42016 [\*getsockopt\(\)\*](#), [\*if\\_freenameindex\(\)\*](#), [\*if\\_indextoname\(\)\*](#), [\*if\\_nametoindex\(\)\*](#), [\*setsockopt\(\)\*](#)

42017 XBD [\*\*<net/if.h>\*\*](#)

42018 **CHANGE HISTORY**

42019 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

42020 **NAME**

42021 if\_nametoindex — map a network interface name to its corresponding index

42022 **SYNOPSIS**

42023 #include <net/if.h>

42024 unsigned if\_nametoindex(const char \*ifname);

42025 **DESCRIPTION**

42026 The *if\_nametoindex()* function shall return the interface index corresponding to name *ifname*.

42027 **RETURN VALUE**

42028 The corresponding index if *ifname* is the name of an interface; otherwise, zero.

42029 **ERRORS**

42030 No errors are defined.

42031 **EXAMPLES**

42032 None.

42033 **APPLICATION USAGE**

42034 None.

42035 **RATIONALE**

42036 None.

42037 **FUTURE DIRECTIONS**

42038 None.

42039 **SEE ALSO**

42040 *getsockopt()*, *if\_freenameindex()*, *if\_indextoname()*, *if\_nameindex()*, *setsockopt()*

42041 XBD <net/if.h>

42042 **CHANGE HISTORY**

42043 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

42044 **NAME**

42045           ilogb, ilogbf, ilogbl — return an unbiased exponent

42046 **SYNOPSIS**

```
42047     #include <math.h>
42048     int ilogb(double x);
42049     int ilogbf(float x);
42050     int ilogbl(long double x);
```

42051 **DESCRIPTION**

42052 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
 42053 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42054 volume of POSIX.1-2024 defers to the ISO C standard.

42055       These functions shall return the exponent part of their argument  $x$ . Formally, the return value is  
 42056 the integral part of  $\log_r |x|$  as a signed integral value, for non-zero  $x$ , where  $r$  is the radix of the  
 42057 machine's floating-point arithmetic, which is the value of `FLT_RADIX` defined in `<float.h>`.

42058       An application wishing to check for error situations should set `errno` to zero and call  
 42059 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 42060 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 42061 zero, an error has occurred.

42062 **RETURN VALUE**

42063       Upon successful completion, these functions shall return the exponent part of  $x$  as a signed  
 42064 integer value. They are equivalent to calling the corresponding `logb()` function and casting the  
 42065 returned value to type `int`.

42066 MX       When the correct result is representable in the range of the return type, the returned value shall  
 42067 be exact and shall be independent of the current rounding direction mode.

42068 XSI|MX   If  $x$  is 0, the value `FP_ILOGB0` shall be returned. On XSI-conformant systems and on systems  
 42069 that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a  
 42070 CX       domain error may occur.

42071 XSI|MX   If  $x$  is  $\pm\text{Inf}$ , the value `{INT_MAX}` shall be returned. On XSI-conformant systems and on  
 42072 systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise,  
 42073 CX       a domain error may occur.

42074 XSI|MX   If  $x$  is a NaN, the value `FP_ILOGBNAN` shall be returned. On XSI-conformant systems and on  
 42075 systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise,  
 42076 CX       a domain error may occur.

42077       If the correct value is greater than `{INT_MAX}` or less than `{INT_MIN}`, an unspecified value  
 42078 XSI       shall be returned. On XSI-conformant systems, a domain error shall occur and `{INT_MAX}` or  
 42079 `{INT_MIN}`, respectively, shall be returned;

42080 MX       if the IEC 60559 Floating-Point option is supported, a domain error shall occur; otherwise, a  
 42081 domain error or range error may occur.

42082 **ERRORS**

42083       These functions shall fail if:

42084 XSI|MX   **Domain Error**   The correct value is not representable as an integer.

42085           The  $x$  argument is zero, NaN, or  $\pm\text{Inf}$ .

42086           If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 42087 then `errno` shall be set to `[EDOM]`. If the integer expression `(math_errhandling`  
 42088 `& MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception



42089 shall be raised.

42090 These functions may fail if:

42091 Domain Error The  $x$  argument is zero, NaN, or  $\pm\text{Inf}$ .

42092 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
42093 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
42094 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
42095 shall be raised.

#### 42096 EXAMPLES

42097 None.

#### 42098 APPLICATION USAGE

42099 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
42100 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

#### 42101 RATIONALE

42102 The errors come from taking the expected floating-point value and converting it to **int**, which is  
42103 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not  
42104 representable in a type **int**), so should be a domain error.

42105 There are no known implementations that overflow. For overflow to happen, {INT\_MAX} must  
42106 be less than  $\text{LDBL\_MAX\_EXP} * \log_2(\text{FLT\_RADIX})$  or {INT\_MIN} must be greater than  
42107  $\text{LDBL\_MIN\_EXP} * \log_2(\text{FLT\_RADIX})$  if subnormals are not supported, or {INT\_MIN} must be  
42108 greater than  $(\text{LDBL\_MIN\_EXP} - \text{LDBL\_MANT\_DIG}) * \log_2(\text{FLT\_RADIX})$  if subnormals are  
42109 supported.

#### 42110 FUTURE DIRECTIONS

42111 None.

#### 42112 SEE ALSO

42113 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [logb\(\)](#), [scalbln\(\)](#)

42114 XBD Section 4.23 (on page 109), [<float.h>](#), [<math.h>](#)

#### 42115 CHANGE HISTORY

42116 First released in Issue 4, Version 2.

#### 42117 Issue 5

42118 Moved from X/OPEN UNIX extension to BASE.

#### 42119 Issue 6

42120 The *ilogb()* function is no longer marked as an extension.

42121 The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999  
42122 standard.

42123 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

42124 Functionality relating to the XSI option is marked.

#### 42125 Issue 7

42126 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79  
42127 (SD5-XSH-ERN-72) are applied.

42128 **Issue 8**

42129 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
42130 standard.

42131 **NAME**

42132 imaxabs — return absolute value

42133 **SYNOPSIS**

42134 #include &lt;inttypes.h&gt;

42135 intmax\_t imaxabs(intmax\_t j);

42136 **DESCRIPTION**

42137 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
42138 conflict between the requirements described here and the ISO C standard is unintentional. This  
42139 volume of POSIX.1-2024 defers to the ISO C standard.

42140 The *imaxabs()* function shall compute the absolute value of an integer *j*. If the result cannot be  
42141 represented, the behavior is undefined.

42142 **RETURN VALUE**42143 The *imaxabs()* function shall return the absolute value.42144 **ERRORS**

42145 No errors are defined.

42146 **EXAMPLES**

42147 None.

42148 **APPLICATION USAGE**

42149 Since POSIX.1 requires a two's complement representation of **intmax\_t**, the absolute value of the  
42150 negative integer with the largest magnitude {INTMAX\_MIN} is not representable, thus  
42151 *imaxabs*(INTMAX\_MIN) is undefined.

42152 **RATIONALE**

42153 None.

42154 **FUTURE DIRECTIONS**

42155 None.

42156 **SEE ALSO**42157 [\*imaxdiv\(\)\*](#)42158 XBD [<inttypes.h>](#)42159 **CHANGE HISTORY**

42160 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42161 **Issue 8**

42162 Austin Group Defect 1108 is applied, changing the APPLICATION USAGE section.

42163 **NAME**

42164 imaxdiv — return quotient and remainder

42165 **SYNOPSIS**

42166 #include &lt;inttypes.h&gt;

42167 imaxdiv\_t imaxdiv(intmax\_t numer, intmax\_t denom);

42168 **DESCRIPTION**

42169 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
42170 conflict between the requirements described here and the ISO C standard is unintentional. This  
42171 volume of POSIX.1-2024 defers to the ISO C standard.

42172 The *imaxdiv()* function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.42173 **RETURN VALUE**

42174 The *imaxdiv()* function shall return a structure of type **imaxdiv\_t**, comprising both the quotient  
42175 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)  
42176 and *rem* (the remainder), each of which has type **intmax\_t**.

42177 If either part of the result cannot be represented, the behavior is undefined.

42178 **ERRORS**

42179 No errors are defined.

42180 **EXAMPLES**

42181 None.

42182 **APPLICATION USAGE**

42183 None.

42184 **RATIONALE**

42185 None.

42186 **FUTURE DIRECTIONS**

42187 None.

42188 **SEE ALSO**42189 [imaxabs\(\)](#)42190 XBD [<inttypes.h>](#)42191 **CHANGE HISTORY**

42192 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

**42193 NAME**

42194 in6addr\_any, in6addr\_loopback — IPv6 address variables

**42195 SYNOPSIS**

```
42196 IP6 #include <netinet/in.h>
42197     const struct in6_addr in6addr_any;
42198     const struct in6_addr in6addr_loopback;
```

**42199 DESCRIPTION**

42200 The *in6addr\_any* variable is initialized by the system to contain the wildcard IPv6 address (:::).

42201 The *in6addr\_loopback* variable is initialized by the system to contain the loopback IPv6 address  
42202 (:::1).

**42203 RETURN VALUE**

42204 None.

**42205 ERRORS**

42206 No errors are defined.

**42207 EXAMPLES**

42208 None.

**42209 APPLICATION USAGE**

42210 None.

**42211 RATIONALE**

42212 These variables were only described on the XBD <netinet/in.h> page in earlier versions of this  
42213 standard.

**42214 FUTURE DIRECTIONS**

42215 None.

**42216 SEE ALSO**

42217 [bind\(\)](#), [connect\(\)](#)

42218 XBD <netinet/in.h>

**42219 CHANGE HISTORY**

42220 First released in Issue 8. Derived from Issue 7 XBD <netinet/in.h>.

42221 **NAME**

42222 inet\_addr, inet\_ntoa — IPv4 address manipulation

42223 **SYNOPSIS**

```
42224 OB #include <arpa/inet.h>
42225 in_addr_t inet_addr(const char *cp);
42226 char *inet_ntoa(struct in_addr in);
```

42227 **DESCRIPTION**

42228 The *inet\_addr()* function shall convert the string pointed to by *cp*, in the standard IPv4 dotted  
42229 decimal notation, to an integer value suitable for use as an Internet address.

42230 The *inet\_ntoa()* function shall convert the Internet host address specified by *in* to a string in the  
42231 Internet standard dot notation.

42232 The *inet\_ntoa()* function need not be thread-safe.

42233 All Internet addresses shall be returned in network order (bytes ordered from left to right).

42234 Values specified using IPv4 dotted decimal notation take one of the following forms:

42235 a.b.c.d When four parts are specified, each shall be interpreted as a byte of data and  
42236 assigned, from left to right, to the four bytes of an Internet address.

42237 a.b.c When a three-part address is specified, the last part shall be interpreted as a 16-bit  
42238 quantity and placed in the rightmost two bytes of the network address. This makes  
42239 the three-part address format convenient for specifying Class B network addresses  
42240 as "128.net.host".

42241 a.b When a two-part address is supplied, the last part shall be interpreted as a 24-bit  
42242 quantity and placed in the rightmost three bytes of the network address. This  
42243 makes the two-part address format convenient for specifying Class A network  
42244 addresses as "net.host".

42245 a When only one part is given, the value shall be stored directly in the network  
42246 address without any byte rearrangement.

42247 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or  
42248 hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal;  
42249 otherwise, a leading '0' implies octal; otherwise, the number is interpreted as decimal).

42250 **RETURN VALUE**

42251 Upon successful completion, *inet\_addr()* shall return the Internet address. Otherwise, it shall  
42252 return (**in\_addr\_t**)(-1).

42253 The *inet\_ntoa()* function shall return a pointer to the network address in Internet standard dot  
42254 notation.

42255 **ERRORS**

42256 No errors are defined.

42257 **EXAMPLES**

42258 None.

42259 **APPLICATION USAGE**42260 The return value of *inet\_ntoa()* may point to static data that may be overwritten by subsequent  
42261 calls to *inet\_ntoa()*.42262 Applications should prefer *inet\_pton()* over *inet\_addr()* for the following reasons:

- 42263
- The return value from *inet\_addr()* when converting 255.255.255.255 is indistinguishable  
42264 from an error.
  - The *inet\_pton()* function supports multiple address families.
  - The alternative textual representations supported by *inet\_addr()* (but not by *inet\_pton()*)  
42266 are often used maliciously to confuse or mislead users (e.g., for phishing).  
42267

42268 Applications should prefer *inet\_ntop()* over *inet\_ntoa()* as it supports multiple address families  
42269 and is thread-safe.42270 **RATIONALE**

42271 None.

42272 **FUTURE DIRECTIONS**42273 These functions are included only for compatibility with older implementations and may be  
42274 removed in a future version.42275 **SEE ALSO**42276 [\*endhostent\(\)\*, \*endnetent\(\)\*, \*inet\\_ntop\(\)\*](#)42277 XBD <[arpa/inet.h](#)>42278 **CHANGE HISTORY**

42279 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

42280 **Issue 7**

42281 Austin Group Interpretation 1003.1-2001 #156 is applied.

42282 **Issue 8**42283 Austin Group Defects 1101 and 1102 are applied, marking *inet\_addr()* and *inet\_ntoa()* as  
42284 obsolescent.

42285 **NAME**

42286 inet\_ntop, inet\_pton — convert IPv4 and IPv6 addresses between binary and text form

42287 **SYNOPSIS**

42288 #include &lt;arpa/inet.h&gt;

42289 const char \*inet\_ntop(int af, const void \*restrict src,  
42290 char \*restrict dst, socklen\_t size);

42291 int inet\_pton(int af, const char \*restrict src, void \*restrict dst);

42292 **DESCRIPTION**

42293 The *inet\_ntop()* function shall convert a numeric address into a text string suitable for  
 42294 IP6 presentation. The *af* argument shall specify the family of the address. This can be AF\_INET or  
 42295 AF\_INET6. The *src* argument points to a buffer holding an IPv4 address if the *af* argument is  
 42296 IP6 AF\_INET, or an IPv6 address if the *af* argument is AF\_INET6; the address needs to be in  
 42297 network byte order. The *dst* argument points to a buffer where the function stores the resulting  
 42298 text string; it shall not be NULL. The *size* argument specifies the size of this buffer, which shall  
 42299 IP6 be large enough to hold the text string (INET\_ADDRSTRLEN characters for IPv4,  
 42300 INET6\_ADDRSTRLEN characters for IPv6).

42301 The *inet\_pton()* function shall convert an address in its standard text presentation form into its  
 42302 IP6 numeric binary form. The *af* argument shall specify the family of the address. The AF\_INET and  
 42303 AF\_INET6 address families shall be supported. The *src* argument points to the string being  
 42304 passed in. The *dst* argument points to a buffer into which the function stores the numeric  
 42305 IP6 address; this shall be large enough to hold the numeric address (32 bits for AF\_INET, 128 bits  
 42306 for AF\_INET6).

42307 If the *af* argument of *inet\_pton()* is AF\_INET, the *src* string shall be in the standard IPv4 dotted-  
 42308 decimal form:

42309 ddd.ddd.ddd.ddd

42310 where "ddd" is a one to three digit decimal number between 0 and 255. Leading zeros shall be  
 42311 allowed. The *inet\_pton()* function does not accept other formats (such as the octal numbers,  
 42312 hexadecimal numbers, and fewer than four numbers that *inet\_addr()* accepts).

42313 IP6 If the *af* argument of *inet\_pton()* is AF\_INET6, the *src* string shall be in one of the following  
 42314 standard IPv6 text forms:

- 42315 1. The preferred form is "x:x:x:x:x:x:x:x", where the 'x's are the hexadecimal values  
 42316 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be  
 42317 omitted, but there shall be one to four hexadecimal digits in every field.
- 42318 2. A string of contiguous zero fields in the preferred form can be shown as "::". The "::"  
 42319 can only appear once in an address. Unspecified addresses ("0:0:0:0:0:0:0:0") may  
 42320 be represented simply as "::".
- 42321 3. A third form that is sometimes more convenient when dealing with a mixed environment  
 42322 of IPv4 and IPv6 nodes is "x:x:x:x:x:x.d.d.d.d", where the 'x's are the  
 42323 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the  
 42324 decimal values of the four low-order 8-bit pieces of the address (standard IPv4  
 42325 representation).

42326 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in  
 42327 RFC 4291.



42328 **RETURN VALUE**

42329 The *inet\_ntop()* function shall return a pointer to the buffer containing the text string if the  
 42330 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

42331 The *inet\_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by  
 42332 IP6 *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string  
 42333 or a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is  
 42334 unknown.

42335 **ERRORS**

42336 The *inet\_ntop()* and *inet\_pton()* functions shall fail if:

42337 [EAFNOSUPPORT]

42338 The *af* argument is invalid.

42339 [ENOSPC] The size of the *inet\_ntop()* result buffer is inadequate.

42340 **EXAMPLES**

42341 None.

42342 **APPLICATION USAGE**

42343 None.

42344 **RATIONALE**

42345 None.

42346 **FUTURE DIRECTIONS**

42347 None.

42348 **SEE ALSO**

42349 [\*inet\\_addr\(\)\*](#)

42350 XBD <[arpa/inet.h](#)>

42351 **CHANGE HISTORY**

42352 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

42353 IPv6 extensions are marked.

42354 The **restrict** keyword is added to the *inet\_ntop()* and *inet\_pton()* prototypes for alignment with  
 42355 the ISO/IEC 9899:1999 standard.

42356 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding “the address must  
 42357 be in network byte order” to the end of the fourth sentence of the first paragraph in the  
 42358 DESCRIPTION.

42359 **Issue 7**

42360 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0178 [777] is applied.

42361 **Issue 8**

42362 Austin Group Defect 1102 is applied, removing a reference to the [\*inet\\_addr\(\)\*](#) page from the  
 42363 DESCRIPTION.

42364 Austin Group Defect 1573 is applied, clarifying that leading zeros are allowed in the IPv4  
 42365 dotted-decimal form accepted by *inet\_pton()*.

42366 Austin Group Defect 1685 is applied, updating RFC references.

42367 **NAME**

42368           initstate, random, setstate, srandom — pseudo-random number functions

42369 **SYNOPSIS**

```
42370 XSI       #include <stdlib.h>
42371       char *initstate(unsigned seed, char *state, size_t size);
42372       long random(void);
42373       char *setstate(char *state);
42374       void srandom(unsigned seed);
```

42375 **DESCRIPTION**

42376       The *random()* function shall use a non-linear additive feedback random-number generator  
 42377       employing a default state array size of 31 **long** integers to return successive pseudo-random  
 42378       numbers in the range from 0 to  $2^{31}-1$ . The period of this random-number generator is  
 42379       approximately  $16 \times (2^{31}-1)$ . The size of the state array determines the period of the random-  
 42380       number generator. Increasing the state array size shall increase the period.

42381       With 256 bytes of state information, the period of the random-number generator shall be greater  
 42382       than  $2^{69}$ .

42383       Like *rand()*, *random()* shall produce by default a sequence of numbers that can be duplicated by  
 42384       calling *srandom()* with 1 as the seed.

42385       The *srandom()* function shall initialize the current state array using the value of *seed*.

42386       The *initstate()* and *setstate()* functions handle restarting and changing random-number  
 42387       generators. The *initstate()* function allows a state array, pointed to by the *state* argument, to be  
 42388       initialized for future use. The *size* argument, which specifies the size in bytes of the state array,  
 42389       shall be used by *initstate()* to decide what type of random-number generator to use; the larger  
 42390       the state array, the more random the numbers. Values for the amount of state information are 8,  
 42391       32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one  
 42392       of these values. If *initstate()* is called with  $8 \leq \text{size} < 32$ , then *random()* shall use a simple linear  
 42393       congruential random number generator. The *seed* argument specifies a starting point for the  
 42394       random-number sequence and provides for restarting at the same point. The *initstate()* function  
 42395       shall return a pointer to the previous state information array.

42396       If *initstate()* has not been called, then *random()* shall behave as though *initstate()* had been called  
 42397       with *seed*=1 and *size*=128.

42398       Once a state has been initialized, *setstate()* allows switching between state arrays. The array  
 42399       defined by the *state* argument shall be used for further random-number generation until  
 42400       *initstate()* is called or *setstate()* is called again. The *setstate()* function shall return a pointer to the  
 42401       previous state array.

42402 **RETURN VALUE**

42403       If *initstate()* is called with *size* less than 8, it shall return NULL.

42404       The *random()* function shall return the generated pseudo-random number.

42405       The *srandom()* function shall not return a value.

42406       Upon successful completion, *initstate()* and *setstate()* shall return a pointer to the previous state  
 42407       array; otherwise, a null pointer shall be returned.

42408 **ERRORS**

42409 No errors are defined.

42410 **EXAMPLES**

42411 None.

42412 **APPLICATION USAGE**

42413 After initialization, a state array can be restarted at a different point in one of two ways:

- 42414 1. The *initstate()* function can be used, with the desired seed, state array, and size of the
- 42415 array.
- 42416 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with
- 42417 the desired seed. The advantage of using both of these functions is that the size of the
- 42418 state array does not have to be saved once it is initialized.

42419 Although some implementations of *random()* have written messages to standard error, such

42420 implementations do not conform to POSIX.1-2024.

42421 Issue 5 restored the historical behavior of this function.

42422 Threaded applications should use *erand48()*, *nrand48()*, or *jrand48()* instead of *random()* when

42423 an independent random number sequence in multiple threads is required.

42424 These functions should be avoided whenever non-trivial requirements (including safety) have to

42425 be fulfilled, unless seeded using *getentropy()*.

42426 **RATIONALE**

42427 None.

42428 **FUTURE DIRECTIONS**

42429 None.

42430 **SEE ALSO**42431 *drand48()*, *getentropy()*, *rand()*

42432 XBD &lt;stdlib.h&gt;

42433 **CHANGE HISTORY**

42434 First released in Issue 4, Version 2.

42435 **Issue 5**

42436 Moved from X/OPEN UNIX extension to BASE.

42437 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than

42438 or equal to 8, or less than 32”, “size<8” is replaced with “8≤size <32”, and a new first paragraph

42439 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE

42440 indicating that these changes restore the historical behavior of the function.

42441 **Issue 6**

42442 In the DESCRIPTION, duplicate text “For values greater than or equal to 8 . . .” is removed.

42443 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand\_r()* from the

42444 list of suggested functions in the APPLICATION USAGE section.

42445 **Issue 7**42446 The type of the first argument to *setstate()* is changed from **const char \*** to **char \***.

42447 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0179 [743] is applied.



42448 **Issue 8**  
42449

Austin Group Defect 1134 is applied, adding *getentropy()*.



42450 **NAME**

42451 insque, remque — insert or remove an element in a queue

42452 **SYNOPSIS**

```
42453 XSI      #include <search.h>
42454         void insque(void *element, void *pred);
42455         void remque(void *element);
```

42456 **DESCRIPTION**

42457 The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The  
 42458 queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it  
 42459 defines a structure in which the first two members of the structure are pointers to the same type  
 42460 of structure, and any further members are application-specific. The first member of the structure  
 42461 is a forward pointer to the next entry in the queue. The second member is a backward pointer to  
 42462 the previous entry in the queue. If the queue is linear, the queue is terminated with null  
 42463 pointers. The names of the structure and of the pointer members are not subject to any special  
 42464 restriction.

42465 The *insque()* function shall insert the element pointed to by *element* into a queue immediately  
 42466 after the element pointed to by *pred*.

42467 The *remque()* function shall remove the element pointed to by *element* from a queue.

42468 If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the  
 42469 initial element of the queue, shall initialize the forward and backward pointers of *element* to null  
 42470 pointers.

42471 If the queue is to be used as a circular list, the application shall ensure it initializes the forward  
 42472 pointer and the backward pointer of the initial element of the queue to the element's own  
 42473 address.

42474 **RETURN VALUE**42475 The *insque()* and *remque()* functions do not return a value.42476 **ERRORS**

42477 No errors are defined.

42478 **EXAMPLES**42479 **Creating a Linear Linked List**

42480 The following example creates a linear linked list.

```
42481 #include <search.h>
42482 ...
42483 struct myque element1;
42484 struct myque element2;
42485
42486 char *data1 = "DATA1";
42487 char *data2 = "DATA2";
42488 ...
42489 element1.data = data1;
42490 element2.data = data2;
42491
42492 insque (&element1, NULL);
42493 insque (&element2, &element1);
```

42492 **Creating a Circular Linked List**

42493 The following example creates a circular linked list.

```

42494 #include <search.h>
42495 ...
42496 struct myque element1;
42497 struct myque element2;
42498
42499 char *data1 = "DATA1";
42500 char *data2 = "DATA2";
42501 ...
42502 element1.data = data1;
42503 element2.data = data2;
42504
42505 element1.fwd = &element1;
42506 element1.bck = &element1;
42507
42508 insque (&element2, &element1);

```

42506 **Removing an Element**42507 The following example removes the element pointed to by *element1*.

```

42508 #include <search.h>
42509 ...
42510 struct myque element1;
42511 ...
42512 remque (&element1);

```

42513 **APPLICATION USAGE**

42514 The historical implementations of these functions described the arguments as being of type  
 42515 **struct qelem \*** rather than as being of type **void \*** as defined here. In those implementations,  
 42516 **struct qelem** was commonly defined in **<search.h>** as:

```

42517 struct qelem {
42518     struct qelem *q_forw;
42519     struct qelem *q_back;
42520 };

```

42521 Applications using these functions, however, were never able to use this structure directly since  
 42522 it provided no room for the actual data contained in the elements. Most applications defined  
 42523 structures that contained the two pointers as the initial elements and also provided space for, or  
 42524 pointers to, the object's data. Applications that used these functions to update more than one  
 42525 type of table also had the problem of specifying two or more different structures with the same  
 42526 name, if they literally used **struct qelem** as specified.

42527 As described here, the implementations were actually expecting a structure type where the first  
 42528 two members were forward and backward pointers to structures. With C compilers that didn't  
 42529 provide function prototypes, applications used structures as specified in the DESCRIPTION  
 42530 above and the compiler did what the application expected.

42531 If this method had been carried forward with an ISO C standard compiler and the historical  
 42532 function prototype, most applications would have to be modified to cast pointers to the  
 42533 structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By  
 42534 specifying **void \*** as the argument type, applications do not need to change (unless they  
 42535 specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

42536 **RATIONALE**

42537 None.

42538 **FUTURE DIRECTIONS**

42539 None.

42540 **SEE ALSO**

42541 XBD [<search.h>](#)

42542 **CHANGE HISTORY**

42543 First released in Issue 4, Version 2.

42544 **Issue 5**

42545 Moved from X/OPEN UNIX extension to BASE.

42546 **Issue 6**

42547 The normative text is updated to avoid use of the term “must” for application requirements.

42548 **NAME**

42549 isalnum, isalnum\_l — test for an alphanumeric character

42550 **SYNOPSIS**

```
42551 #include <ctype.h>
42552 int isalnum(int c);
42553 CX int isalnum_l(int c, locale_t locale);
```

42554 **DESCRIPTION**

42555 CX For *isalnum()*: The functionality described on this reference page is aligned with the ISO C  
 42556 standard. Any conflict between the requirements described here and the ISO C standard is  
 42557 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

42558 CX The *isalnum()* and *isalnum\_l()* functions shall test whether *c* is a character of class **alpha** or  
 42559 CX **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7  
 42560 (on page 127).

42561 The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
 42562 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
 42563 behavior is undefined.

42564 CX The behavior is undefined if the *locale* argument to *isalnum\_l()* is the special locale object  
 42565 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

42566 **RETURN VALUE**

42567 CX The *isalnum()* and *isalnum\_l()* functions shall return non-zero if *c* is an alphanumeric character;  
 42568 otherwise, they shall return 0.

42569 **ERRORS**

42570 No errors are defined.

42571 **EXAMPLES**

42572 None.

42573 **APPLICATION USAGE**

42574 To ensure applications portability, especially across natural languages, only these functions and  
 42575 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 42576 classification.

42577 **RATIONALE**

42578 None.

42579 **FUTURE DIRECTIONS**

42580 None.

42581 **SEE ALSO**

42582 *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
 42583 *isxdigit()*, *setlocale()*, *uselocale()*

42584 XBD Chapter 7 (on page 127), [<ctype.h>](#), [<stdio.h>](#)42585 **CHANGE HISTORY**

42586 First released in Issue 1. Derived from Issue 1 of the SVID.

42587 **Issue 6**

42588 The normative text is updated to avoid use of the term “must” for application requirements.



42589 **Issue 7**

42590 The *isalnum\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
42591 API Set Part 4.

42592 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0274 [302], XSH/TC1-2008/0275 [283],  
42593 and XSH/TC1-2008/0276 [283] are applied.

42594 **NAME**

42595 isalpha, isalpha\_l — test for an alphabetic character

42596 **SYNOPSIS**

42597 #include &lt;ctype.h&gt;

42598 int isalpha(int c);

42599 CX int isalpha\_l(int c, locale\_t locale);

42600 **DESCRIPTION**42601 CX For *isalpha()*: The functionality described on this reference page is aligned with the ISO C  
42602 standard. Any conflict between the requirements described here and the ISO C standard is  
42603 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.42604 CX The *isalpha()* and *isalpha\_l()* functions shall test whether *c* is a character of class **alpha** in the  
42605 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page  
42606 127).42607 The *c* argument is an **int**, the value of which the application shall ensure is representable as an  
42608 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the  
42609 behavior is undefined.42610 CX The behavior is undefined if the *locale* argument to *isalpha\_l()* is the special locale object  
42611 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.42612 **RETURN VALUE**42613 CX The *isalpha()* and *isalpha\_l()* functions shall return non-zero if *c* is an alphabetic character;  
42614 otherwise, they shall return 0.42615 **ERRORS**

42616 No errors are defined.

42617 **EXAMPLES**

42618 None.

42619 **APPLICATION USAGE**42620 To ensure applications portability, especially across natural languages, only these functions and  
42621 the functions in the reference pages listed in the SEE ALSO section should be used for character  
42622 classification.42623 **RATIONALE**

42624 None.

42625 **FUTURE DIRECTIONS**

42626 None.

42627 **SEE ALSO**42628 *isalnum()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
42629 *isxdigit()*, *setlocale()*, *uselocale()*

42630 XBD Chapter 7 (on page 127), &lt;ctype.h&gt;, &lt;locale.h&gt;, &lt;stdio.h&gt;

42631 **CHANGE HISTORY**

42632 First released in Issue 1. Derived from Issue 1 of the SVID.

42633 **Issue 6**

42634 The normative text is updated to avoid use of the term “must” for application requirements.

42635 **Issue 7**42636  
42637

The *isalpha\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

42638  
42639

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0277 [302], XSH/TC1-2008/0278 [283], and XSH/TC1-2008/0279 [283] are applied.

42640 **NAME**

42641 isatty — test for a terminal device

42642 **SYNOPSIS**

42643 #include &lt;unistd.h&gt;

42644 int isatty(int *fildes*);42645 **DESCRIPTION**42646 The *isatty()* function shall test whether *fildes*, an open file descriptor, is associated with a  
42647 terminal device.42648 **RETURN VALUE**42649 The *isatty()* function shall return 1 if *fildes* is associated with a terminal; otherwise, it shall return  
42650 0 and may set *errno* to indicate the error.42651 **ERRORS**42652 The *isatty()* function may fail if:42653 [EBADF] The *fildes* argument is not a valid open file descriptor.42654 [ENOTTY] The file associated with the *fildes* argument is not a terminal.42655 **EXAMPLES**

42656 None.

42657 **APPLICATION USAGE**42658 The *isatty()* function does not necessarily indicate that a human being is available for interaction  
42659 via *fildes*. It is quite possible that non-terminal devices are connected to the communications  
42660 line.42661 **RATIONALE**

42662 None.

42663 **FUTURE DIRECTIONS**

42664 None.

42665 **SEE ALSO**

42666 XBD &lt;unistd.h&gt;

42667 **CHANGE HISTORY**

42668 First released in Issue 1. Derived from Issue 1 of the SVID.

42669 **Issue 6**42670 The following new requirements on POSIX implementations derive from alignment with the  
42671 Single UNIX Specification:

- 42672
- The optional setting of *errno* to indicate an error is added.
  - The [EBADF] and [ENOTTY] optional error conditions are added.

42674 **Issue 7**

42675 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

42676 **NAME**

42677 isblank, isblank\_l — test for a blank character

42678 **SYNOPSIS**

42679 #include &lt;ctype.h&gt;

42680 int isblank(int c);

42681 CX int isblank\_l(int c, locale\_t locale);

42682 **DESCRIPTION**42683 CX For *isblank()*: The functionality described on this reference page is aligned with the ISO C  
42684 standard. Any conflict between the requirements described here and the ISO C standard is  
42685 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.42686 CX The *isblank()* and *isblank\_l()* functions shall test whether *c* is a character of class **blank** in the  
42687 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page  
42688 127).42689 The *c* argument is a type **int**, the value of which the application shall ensure is a character  
42690 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
42691 any other value, the behavior is undefined.42692 CX The behavior is undefined if the *locale* argument to *isblank\_l()* is the special locale object  
42693 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.42694 **RETURN VALUE**42695 CX The *isblank()* and *isblank\_l()* functions shall return non-zero if *c* is a <blank>; otherwise, they  
42696 shall return 0.42697 **ERRORS**

42698 No errors are defined.

42699 **EXAMPLES**

42700 None.

42701 **APPLICATION USAGE**42702 To ensure applications portability, especially across natural languages, only these functions and  
42703 the functions in the reference pages listed in the SEE ALSO section should be used for character  
42704 classification.42705 **RATIONALE**

42706 None.

42707 **FUTURE DIRECTIONS**

42708 None.

42709 **SEE ALSO**42710 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
42711 *isxdigit()*, *setlocale()*, *uselocale()*

42712 XBD Chapter 7 (on page 127), &lt;ctype.h&gt;, &lt;locale.h&gt;

42713 **CHANGE HISTORY**

42714 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42715 **Issue 7**42716 The *isblank\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
42717 Set Part 4.

42718  
42719

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0280 [302], XSH/TC1-2008/0281 [283], and XSH/TC1-2008/0282 [283] are applied.

42720 **NAME**

42721           iscntrl, iscntrl\_l — test for a control character

42722 **SYNOPSIS**

42723           #include &lt;ctype.h&gt;

42724           int iscntrl(int c);

42725 CX        int iscntrl\_l(int c, locale\_t locale);

42726 **DESCRIPTION**42727 CX        For *iscntrl()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.42730 CX        The *iscntrl()* and *iscntrl\_l()* functions shall test whether *c* is a character of class **cntrl** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).42733           The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.42736 CX        The behavior is undefined if the *locale* argument to *iscntrl\_l()* is the special locale object LC\_GLOBAL\_LOCALE or is not a valid locale object handle.42738 **RETURN VALUE**42739 CX        The *iscntrl()* and *iscntrl\_l()* functions shall return non-zero if *c* is a control character; otherwise, they shall return 0.42741 **ERRORS**

42742           No errors are defined.

42743 **EXAMPLES**

42744           None.

42745 **APPLICATION USAGE**

42746           To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

42749 **RATIONALE**

42750           None.

42751 **FUTURE DIRECTIONS**

42752           None.

42753 **SEE ALSO**42754           *isalnum()*, *isalpha()*, *isblank()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

42756           XBD Chapter 7 (on page 127), &lt;ctype.h&gt;, &lt;locale.h&gt;

42757 **CHANGE HISTORY**

42758           First released in Issue 1. Derived from Issue 1 of the SVID.

42759 **Issue 6**

42760           The normative text is updated to avoid use of the term “must” for application requirements.

42761 **Issue 7**

42762 The *iscntrl\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
42763 Set Part 4.

42764 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0283 [302], XSH/TC1-2008/0284 [283],  
42765 and XSH/TC1-2008/0285 [283] are applied.



42766 **NAME**

42767        isdigit, isdigit\_l — test for a decimal digit

42768 **SYNOPSIS**

42769        #include &lt;ctype.h&gt;

42770        int isdigit(int c);

42771 CX     int isdigit\_l(int c, locale\_t locale);

42772 **DESCRIPTION**42773 CX     For *isdigit()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.42776 CX     The *isdigit()* and *isdigit\_l()* functions shall test whether *c* is a character of class **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).42779        The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.42782 CX     The behavior is undefined if the *locale* argument to *isdigit\_l()* is the special locale object LC\_GLOBAL\_LOCALE or is not a valid locale object handle.42784 **RETURN VALUE**42785 CX     The *isdigit()* and *isdigit\_l()* functions shall return non-zero if *c* is a decimal digit; otherwise, they shall return 0.42787 **ERRORS**

42788        No errors are defined.

42789 **EXAMPLES**

42790        None.

42791 **APPLICATION USAGE**

42792        To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

42795 **RATIONALE**

42796        None.

42797 **FUTURE DIRECTIONS**

42798        None.

42799 **SEE ALSO**42800        *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*

42802        XBD Chapter 7 (on page 127), &lt;ctype.h&gt;, &lt;locale.h&gt;

42803 **CHANGE HISTORY**

42804        First released in Issue 1. Derived from Issue 1 of the SVID.

42805 **Issue 6**

42806        The normative text is updated to avoid use of the term “must” for application requirements.

42807 **Issue 7**42808  
42809

The *isdigit\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

42810  
42811

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0286 [302], XSH/TC1-2008/0287 [283], and XSH/TC1-2008/0288 [283] are applied.

42812 **NAME**

42813 isfinite — test for finite value

42814 **SYNOPSIS**

42815 #include &lt;math.h&gt;

42816 int isfinite(real-floating x);

42817 **DESCRIPTION**

42818 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
42819 conflict between the requirements described here and the ISO C standard is unintentional. This  
42820 volume of POSIX.1-2024 defers to the ISO C standard.

42821 The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or  
42822 normal, and not infinite or NaN). First, an argument represented in a format wider than its  
42823 semantic type is converted to its semantic type. Then determination is based on the type of the  
42824 argument.

42825 **RETURN VALUE**42826 The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.42827 **ERRORS**

42828 No errors are defined.

42829 **EXAMPLES**

42830 None.

42831 **APPLICATION USAGE**

42832 None.

42833 **RATIONALE**

42834 None.

42835 **FUTURE DIRECTIONS**

42836 None.

42837 **SEE ALSO**42838 *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

42839 XBD &lt;math.h&gt;

42840 **CHANGE HISTORY**

42841 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42842 **NAME**

42843 isgraph, isgraph\_l — test for a visible character

42844 **SYNOPSIS**

```
42845 #include <ctype.h>
42846 int isgraph(int c);
42847 CX int isgraph_l(int c, locale_t locale);
```

42848 **DESCRIPTION**

42849 CX For *isgraph()*: The functionality described on this reference page is aligned with the ISO C  
 42850 standard. Any conflict between the requirements described here and the ISO C standard is  
 42851 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

42852 CX The *isgraph()* and *isgraph\_l()* functions shall test whether *c* is a character of class **graph** in the  
 42853 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page  
 42854 127).

42855 The *c* argument is an **int**, the value of which the application shall ensure is a character  
 42856 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 42857 any other value, the behavior is undefined.

42858 CX The behavior is undefined if the *locale* argument to *isgraph\_l()* is the special locale object  
 42859 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

42860 **RETURN VALUE**

42861 CX The *isgraph()* and *isgraph\_l()* functions shall return non-zero if *c* is a character with a visible  
 42862 representation; otherwise, they shall return 0.

42863 **ERRORS**

42864 No errors are defined.

42865 **EXAMPLES**

42866 None.

42867 **APPLICATION USAGE**

42868 To ensure applications portability, especially across natural languages, only these functions and  
 42869 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 42870 classification.

42871 **RATIONALE**

42872 None.

42873 **FUTURE DIRECTIONS**

42874 None.

42875 **SEE ALSO**

42876 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,  
 42877 *isxdigit()*, *setlocale()*, *uselocale()*

42878 XBD Chapter 7 (on page 127), [<ctype.h>](#), [<locale.h>](#)42879 **CHANGE HISTORY**

42880 First released in Issue 1. Derived from Issue 1 of the SVID.

42881 **Issue 6**

42882 The normative text is updated to avoid use of the term “must” for application requirements.

42883 **Issue 7**

42884

42885

The *isgraph\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

42886

42887

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0289 [302], XSH/TC1-2008/0290 [283], and XSH/TC1-2008/0291 [283] are applied.

42888 **NAME**

42889 isgreater, isgreaterequal, isless, islessequal, islessgreater — real-floating relational tests

42890 **SYNOPSIS**

```
42891 #include <math.h>
42892 int isgreater(real-floating x, real-floating y);
42893 int isgreaterequal(real-floating x, real-floating y);
42894 int isless(real-floating x, real-floating y);
42895 int islessequal(real-floating x, real-floating y);
42896 int islessgreater(real-floating x, real-floating y);
```

42897 **DESCRIPTION**

42898 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 42899 conflict between the requirements described here and the ISO C standard is unintentional. This  
 42900 volume of POSIX.1-2024 defers to the ISO C standard.

42901 The *isgreater()* macro shall determine whether its first argument is greater than its second  
 42902 argument. The value of *isgreater(x, y)* shall be equal to  $(x) > (y)$ ; however, unlike  $(x) > (y)$ ,  
 42903 *isgreater(x, y)* shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered.

42904 The *isgreaterequal()* macro shall determine whether its first argument is greater than or equal to  
 42905 its second argument. The value of *isgreaterequal(x, y)* shall be equal to  $(x) \geq (y)$ ; however, unlike  
 42906  $(x) \geq (y)$ , *isgreaterequal(x, y)* shall not raise the invalid floating-point exception when  $x$  and  $y$  are  
 42907 unordered.

42908 The *isless()* macro shall determine whether its first argument is less than its second argument.  
 42909 The value of *isless(x, y)* shall be equal to  $(x) < (y)$ ; however, unlike  $(x) < (y)$ , *isless(x, y)* shall not  
 42910 raise the invalid floating-point exception when  $x$  and  $y$  are unordered.

42911 The *islessequal()* macro shall determine whether its first argument is less than or equal to its  
 42912 second argument. The value of *islessequal(x, y)* shall be equal to  $(x) \leq (y)$ ; however, unlike  
 42913  $(x) \leq (y)$ , *islessequal(x, y)* shall not raise the invalid floating-point exception when  $x$  and  $y$  are  
 42914 unordered.

42915 The *islessgreater()* macro shall determine whether its first argument is less than or greater than  
 42916 its second argument. The *islessgreater(x, y)* macro is similar to  $(x) < (y) \mid \mid (x) > (y)$ ; however,  
 42917 *islessgreater(x, y)* shall not raise the invalid floating-point exception when  $x$  and  $y$  are unordered  
 42918 (nor shall it evaluate  $x$  and  $y$  twice).

42919 MX Relational operators and their corresponding comparison macros shall produce equivalent result  
 42920 values, even if argument values are represented in wider formats. Thus, comparison macro  
 42921 arguments represented in formats wider than their semantic types shall not be converted to the  
 42922 semantic types, unless the wide evaluation method converts operands of relational operators to  
 42923 their semantic types. The standard wide evaluation methods characterized by  
 42924 FLT\_EVAL\_METHOD equal to 1 or 2 (see [<float.h>](#)) do not convert operands of relational  
 42925 operators to their semantic types.

42926 **RETURN VALUE**

42927 Upon successful completion, the *isgreater()* macro shall return the value of  $(x) > (y)$ .

42928 Upon successful completion, the *isgreaterequal()* macro shall return the value of  $(x) \geq (y)$ .

42929 Upon successful completion, the *isless()* macro shall return the value of  $(x) < (y)$ .

42930 Upon successful completion, the *islessequal()* macro shall return the value of  $(x) \leq (y)$ .

42931 Upon successful completion, the *islessgreater()* macro shall return the value of  
 42932  $(x) < (y) \mid \mid (x) > (y)$ .

42933 If  $x$  or  $y$  is NaN, these functions shall return 0.

42934 **ERRORS**

42935 No errors are defined.

42936 **EXAMPLES**

42937 None.

42938 **APPLICATION USAGE**

42939 The relational and equality operators support the usual mathematical relationships between  
42940 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
42941 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
42942 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
42943 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
42944 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
42945 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
42946 indicates that the argument shall be an expression of **real-floating** type.

42947 **RATIONALE**

42948 None.

42949 **FUTURE DIRECTIONS**

42950 None.

42951 **SEE ALSO**

42952 [\*isunordered\(\)\*](#)

42953 XBD [`<float.h>`](#), [`<math.h>`](#)

42954 **CHANGE HISTORY**

42955 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42956 **Issue 8**

42957 The individual pages for these functions have been merged to form a single page, to reduce  
42958 duplication.

42959 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
42960 standard.

42961 **NAME**

42962           isinf — test for infinity

42963 **SYNOPSIS**

42964           #include &lt;math.h&gt;

42965           int isinf(real-floating x);

42966 **DESCRIPTION**

42967 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
42968       conflict between the requirements described here and the ISO C standard is unintentional. This  
42969       volume of POSIX.1-2024 defers to the ISO C standard.

42970       The *isinf()* macro shall determine whether its argument value is an infinity (positive or  
42971       negative). First, an argument represented in a format wider than its semantic type is converted  
42972       to its semantic type. Then determination is based on the type of the argument.

42973 **RETURN VALUE**42974       The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.42975 **ERRORS**

42976       No errors are defined.

42977 **EXAMPLES**

42978       None.

42979 **APPLICATION USAGE**

42980       None.

42981 **RATIONALE**

42982       None.

42983 **FUTURE DIRECTIONS**

42984       None.

42985 **SEE ALSO**42986       [\*fpclassify\(\)\*](#), [\*isfinite\(\)\*](#), [\*isnan\(\)\*](#), [\*isnormal\(\)\*](#), [\*signbit\(\)\*](#)42987       XBD <[\*\*math.h\*\*](#)>42988 **CHANGE HISTORY**

42989       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.



42990 **NAME**

42991 isless, islessequal, islessgreater — real-floating relational tests

42992 **SYNOPSIS**

42993 #include &lt;math.h&gt;

42994 int isless(real-floating *x*, real-floating *y*);42995 int islessequal(real-floating *x*, real-floating *y*);42996 int islessgreater(real-floating *x*, real-floating *y*);42997 **DESCRIPTION**42998 Refer to *isgreater()*.

42999 **NAME**

43000 islower, islower\_l — test for a lowercase letter

43001 **SYNOPSIS**

```
43002     #include <ctype.h>
43003     int islower(int c);
43004 CX    int islower_l(int c, locale_t locale);
```

43005 **DESCRIPTION**

43006 CX For *islower()*: The functionality described on this reference page is aligned with the ISO C  
 43007 standard. Any conflict between the requirements described here and the ISO C standard is  
 43008 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

43009 CX The *islower()* and *islower\_l()* functions shall test whether *c* is a character of class **lower** in the  
 43010 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page  
 43011 127).

43012 The *c* argument is an **int**, the value of which the application shall ensure is a character  
 43013 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 43014 any other value, the behavior is undefined.

43015 CX The behavior is undefined if the *locale* argument to *islower\_l()* is the special locale object  
 43016 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

43017 **RETURN VALUE**

43018 CX The *islower()* and *islower\_l()* functions shall return non-zero if *c* is a lowercase letter; otherwise,  
 43019 they shall return 0.

43020 **ERRORS**

43021 No errors are defined.

43022 **EXAMPLES**43023 **Testing for a Lowercase Letter**

43024 Two examples follow, the first using *islower()*, the second using multiple concurrent locales and  
 43025 *islower\_l()*.

43026 The examples test whether the value is a lowercase letter, based on the current locale, then use it  
 43027 as part of a key value.

```
43028     /* Example 1 -- using islower() */
43029     #include <ctype.h>
43030     #include <stdlib.h>
43031     #include <locale.h>
43032     ...
43033     char *keystr;
43034     int elementlen, len;
43035     unsigned char c;
43036     ...
43037     setlocale(LC_ALL, "");
43038     ...
43039     len = 0;
43040     while (len < elementlen) {
43041         c = (unsigned char) (rand() % 256);
43042         ...
```

```

43043         if (islower(c))
43044             keystr[len++] = c;
43045         }
43046     ...
43047     /* Example 2 -- using islower_l() */
43048     #include <ctype.h>
43049     #include <stdlib.h>
43050     #include <locale.h>
43051     ...
43052     char *keystr;
43053     int elementlen, len;
43054     unsigned char c;
43055     ...
43056     locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
43057     ...
43058     len = 0;
43059     while (len < elementlen) {
43060         c = (unsigned char) (rand() % 256);
43061         ...
43062         if (islower_l(c, loc))
43063             keystr[len++] = c;
43064     }
43065     ...

```

#### 43066 APPLICATION USAGE

43067 To ensure applications portability, especially across natural languages, only these functions and  
 43068 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 43069 classification.

#### 43070 RATIONALE

43071 None.

#### 43072 FUTURE DIRECTIONS

43073 None.

#### 43074 SEE ALSO

43075 [isalnum\(\)](#), [isalpha\(\)](#), [isblank\(\)](#), [iscntrl\(\)](#), [isdigit\(\)](#), [isgraph\(\)](#), [isprint\(\)](#), [ispunct\(\)](#), [isspace\(\)](#), [isupper\(\)](#),  
 43076 [isxdigit\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#)

43077 XBD Chapter 7 (on page 127), [<ctype.h>](#), [<locale.h>](#)

#### 43078 CHANGE HISTORY

43079 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 43080 Issue 6

43081 The normative text is updated to avoid use of the term “must” for application requirements and  
 43082 an example is added.

#### 43083 Issue 7

43084 The [islower\\_l\(\)](#) function is added from The Open Group Technical Standard, 2006, Extended API  
 43085 Set Part 4.

43086  
43087  
43088

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0292 [302], XSH/TC1-2008/0293 [283], XSH/TC1-2008/0294 [283], XSH/TC1-2008/0295 [302], and XSH/TC1-2008/0296 [304] are applied.

43089 **NAME**

43090           isnan — test for a NaN

43091 **SYNOPSIS**

43092           #include &lt;math.h&gt;

43093           int isnan(real-floating x);

43094 **DESCRIPTION**

43095 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
43096       conflict between the requirements described here and the ISO C standard is unintentional. This  
43097       volume of POSIX.1-2024 defers to the ISO C standard.

43098       The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument  
43099       represented in a format wider than its semantic type is converted to its semantic type. Then  
43100       determination is based on the type of the argument.

43101 **RETURN VALUE**43102       The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.43103 **ERRORS**

43104       No errors are defined.

43105 **EXAMPLES**

43106       None.

43107 **APPLICATION USAGE**

43108       None.

43109 **RATIONALE**

43110       None.

43111 **FUTURE DIRECTIONS**

43112       None.

43113 **SEE ALSO**43114       [\*fpclassify\(\)\*](#), [\*isfinite\(\)\*](#), [\*isinf\(\)\*](#), [\*isnormal\(\)\*](#), [\*signbit\(\)\*](#)43115       XBD [\*\*<math.h>\*\*](#)43116 **CHANGE HISTORY**

43117       First released in Issue 3.

43118 **Issue 5**

43119       The DESCRIPTION is updated to indicate the return value when NaN is not supported. This  
43120       text was previously published in the APPLICATION USAGE section.

43121 **Issue 6**

43122       Re-written for alignment with the ISO/IEC 9899:1999 standard.

43123 **NAME**

43124           isnormal — test for a normal value

43125 **SYNOPSIS**

43126           #include &lt;math.h&gt;

43127           int isnormal(real-floating x);

43128 **DESCRIPTION**

43129 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
43130 conflict between the requirements described here and the ISO C standard is unintentional. This  
43131 volume of POSIX.1-2024 defers to the ISO C standard.

43132       The *isnormal()* macro shall determine whether its argument value is normal (neither zero,  
43133 subnormal, infinite, nor NaN). First, an argument represented in a format wider than its  
43134 semantic type is converted to its semantic type. Then determination is based on the type of the  
43135 argument.

43136 **RETURN VALUE**

43137       The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal  
43138 value.

43139 **ERRORS**

43140       No errors are defined.

43141 **EXAMPLES**

43142       None.

43143 **APPLICATION USAGE**

43144       None.

43145 **RATIONALE**

43146       None.

43147 **FUTURE DIRECTIONS**

43148       None.

43149 **SEE ALSO**43150       *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*

43151       XBD &lt;math.h&gt;

43152 **CHANGE HISTORY**

43153       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

43154 **NAME**

43155 isprint, isprint\_l — test for a printable character

43156 **SYNOPSIS**

```
43157 #include <ctype.h>
43158 int isprint(int c);
43159 CX int isprint_l(int c, locale_t locale);
```

43160 **DESCRIPTION**

43161 CX For *isprint()*: The functionality described on this reference page is aligned with the ISO C  
 43162 standard. Any conflict between the requirements described here and the ISO C standard is  
 43163 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

43164 CX The *isprint()* and *isprint\_l()* functions shall test whether *c* is a character of class **print** in the  
 43165 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page  
 43166 127).

43167 The *c* argument is an **int**, the value of which the application shall ensure is a character  
 43168 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 43169 any other value, the behavior is undefined.

43170 CX The behavior is undefined if the *locale* argument to *isprint\_l()* is the special locale object  
 43171 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

43172 **RETURN VALUE**

43173 CX The *isprint()* and *isprint\_l()* functions shall return non-zero if *c* is a printable character;  
 43174 otherwise, they shall return 0.

43175 **ERRORS**

43176 No errors are defined.

43177 **EXAMPLES**

43178 None.

43179 **APPLICATION USAGE**

43180 To ensure applications portability, especially across natural languages, only these functions and  
 43181 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 43182 classification.

43183 **RATIONALE**

43184 None.

43185 **FUTURE DIRECTIONS**

43186 None.

43187 **SEE ALSO**

43188 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *ispunct()*, *isspace()*, *isupper()*,  
 43189 *isxdigit()*, *setlocale()*, *uselocale()*

43190 XBD Chapter 7 (on page 127), [<ctype.h>](#), [<locale.h>](#)43191 **CHANGE HISTORY**

43192 First released in Issue 1. Derived from Issue 1 of the SVID.

43193 **Issue 6**

43194 The normative text is updated to avoid use of the term “must” for application requirements.

43195 **Issue 7**

43196 The *isprint\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
43197 Set Part 4.

43198 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0297 [302], XSH/TC1-2008/0298 [283],  
43199 and XSH/TC1-2008/0299 [283] are applied.



43200 **NAME**

43201           ispunct, ispunct\_l — test for a punctuation character

43202 **SYNOPSIS**

43203           #include &lt;ctype.h&gt;

43204           int ispunct(int c);

43205 CX        int ispunct\_l(int c, locale\_t locale);

43206 **DESCRIPTION**43207 CX        For *ispunct()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43210 CX        The *ispunct()* and *ispunct\_l()* functions shall test whether *c* is a character of class **punct** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).43213           The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.43216 CX        The behavior is undefined if the *locale* argument to *ispunct\_l()* is the special locale object LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43218 **RETURN VALUE**43219 CX        The *ispunct()* and *ispunct\_l()* functions shall return non-zero if *c* is a punctuation character; otherwise, they shall return 0.43221 **ERRORS**

43222           No errors are defined.

43223 **EXAMPLES**

43224           None.

43225 **APPLICATION USAGE**

43226           To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

43229 **RATIONALE**

43230           None.

43231 **FUTURE DIRECTIONS**

43232           None.

43233 **SEE ALSO**43234           *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

43236           XBD Chapter 7 (on page 127), &lt;ctype.h&gt;, &lt;locale.h&gt;

43237 **CHANGE HISTORY**

43238           First released in Issue 1. Derived from Issue 1 of the SVID.

43239 **Issue 6**

43240           The normative text is updated to avoid use of the term “must” for application requirements.

43241 **Issue 7**

43242

43243

The *ispunct\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

43244

43245

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0300 [302], XSH/TC1-2008/0301 [283], and XSH/TC1-2008/0302 [283] are applied.

43246 **NAME**

43247           isspace, isspace\_l — test for a white-space character

43248 **SYNOPSIS**

43249           #include &lt;ctype.h&gt;

43250           int isspace(int c);

43251 CX        int isspace\_l(int c, locale\_t locale);

43252 **DESCRIPTION**43253 CX        For *isspace()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43256 CX        The *isspace()* and *isspace\_l()* functions shall test whether *c* is a character of class **space** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).43259        The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.43262 CX        The behavior is undefined if the *locale* argument to *isspace\_l()* is the special locale object LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43264 **RETURN VALUE**43265 CX        The *isspace()* and *isspace\_l()* functions shall return non-zero if *c* is a white-space character; otherwise, they shall return 0.43267 **ERRORS**

43268        No errors are defined.

43269 **EXAMPLES**

43270        None.

43271 **APPLICATION USAGE**

43272        To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

43275 **RATIONALE**

43276        None.

43277 **FUTURE DIRECTIONS**

43278        None.

43279 **SEE ALSO**43280        *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

43282        XBD Chapter 7 (on page 127), &lt;ctype.h&gt;, &lt;locale.h&gt;

43283 **CHANGE HISTORY**

43284        First released in Issue 1. Derived from Issue 1 of the SVID.

43285 **Issue 6**

43286        The normative text is updated to avoid use of the term “must” for application requirements.

43287 **Issue 7**

43288  
43289

The *isspace\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

43290  
43291

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0303 [302], XSH/TC1-2008/0304 [283], and XSH/TC1-2008/0305 [283] are applied.

43292 **NAME**

43293 isunordered — test if arguments are unordered

43294 **SYNOPSIS**

43295 #include &lt;math.h&gt;

43296 int isunordered(real-floating *x*, real-floating *y*);43297 **DESCRIPTION**

43298 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
43299 conflict between the requirements described here and the ISO C standard is unintentional. This  
43300 volume of POSIX.1-2024 defers to the ISO C standard.

43301 The *isunordered()* macro shall determine whether its arguments are unordered.43302 **RETURN VALUE**43303 Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are  
43304 unordered, and 0 otherwise.43305 If *x* or *y* is NaN, 1 shall be returned.43306 **ERRORS**

43307 No errors are defined.

43308 **EXAMPLES**

43309 None.

43310 **APPLICATION USAGE**

43311 The relational and equality operators support the usual mathematical relationships between  
43312 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,  
43313 greater, and equal) is true. Relational operators may raise the invalid floating-point exception  
43314 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the  
43315 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)  
43316 version of a relational operator. It facilitates writing efficient code that accounts for NaNs  
43317 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**  
43318 indicates that the argument shall be an expression of **real-floating** type.

43319 **RATIONALE**

43320 None.

43321 **FUTURE DIRECTIONS**

43322 None.

43323 **SEE ALSO**43324 [isgreater\(\)](#)43325 XBD [<math.h>](#)43326 **CHANGE HISTORY**

43327 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

43328 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/50 is applied, correcting the RETURN  
43329 VALUE section when *x* or *y* is NaN.

43330 **NAME**

43331 isupper, isupper\_l — test for an uppercase letter

43332 **SYNOPSIS**

```
43333 #include <ctype.h>
43334 int isupper(int c);
43335 CX int isupper_l(int c, locale_t locale);
```

43336 **DESCRIPTION**

43337 CX For *isupper()*: The functionality described on this reference page is aligned with the ISO C  
43338 standard. Any conflict between the requirements described here and the ISO C standard is  
43339 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

43340 CX The *isupper()* and *isupper\_l()* functions shall test whether *c* is a character of class **upper** in the  
43341 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page  
43342 127).

43343 The *c* argument is an **int**, the value of which the application shall ensure is a character  
43344 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
43345 any other value, the behavior is undefined.

43346 CX The behavior is undefined if the *locale* argument to *isupper\_l()* is the special locale object  
43347 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

43348 **RETURN VALUE**

43349 CX The *isupper()* and *isupper\_l()* functions shall return non-zero if *c* is an uppercase letter;  
43350 otherwise, they shall return 0.

43351 **ERRORS**

43352 No errors are defined.

43353 **EXAMPLES**

43354 None.

43355 **APPLICATION USAGE**

43356 To ensure applications portability, especially across natural languages, only these functions and  
43357 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43358 classification.

43359 **RATIONALE**

43360 None.

43361 **FUTURE DIRECTIONS**

43362 None.

43363 **SEE ALSO**

43364 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,  
43365 *isxdigit()*, *setlocale()*, *uselocale()*

43366 XBD Chapter 7 (on page 127), [<ctype.h>](#), [<locale.h>](#)43367 **CHANGE HISTORY**

43368 First released in Issue 1. Derived from Issue 1 of the SVID.

43369 **Issue 6**

43370 The normative text is updated to avoid use of the term “must” for application requirements.

43371 **Issue 7**43372  
43373

The *isupper\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

43374  
43375

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0306 [302], XSH/TC1-2008/0307 [283], and XSH/TC1-2008/0308 [283] are applied.

43376 **NAME**

43377 iswalnum, iswalnum\_l — test for an alphanumeric wide-character code

43378 **SYNOPSIS**

43379 #include &lt;wctype.h&gt;

43380 int iswalnum(wint\_t wc);

43381 CX int iswalnum\_l(wint\_t wc, locale\_t locale);

43382 **DESCRIPTION**43383 CX For *iswalnum()*: The functionality described on this reference page is aligned with the ISO C  
43384 standard. Any conflict between the requirements described here and the ISO C standard is  
43385 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43386 CX The *iswalnum()* and *iswalnum\_l()* functions shall test whether *wc* is a wide-character code  
43387 CX representing a character of class **alpha** or **digit** in the current locale, or in the locale represented  
43388 by *locale*, respectively; see XBD Chapter 7 (on page 127).43389 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
43390 code corresponding to a valid character in the locale used by the function, or equal to the value  
43391 of the macro WEOF. If the argument has any other value, the behavior is undefined.43392 CX The behavior is undefined if the *locale* argument to *iswalnum\_l()* is the special locale object  
43393 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43394 **RETURN VALUE**43395 CX The *iswalnum()* and *iswalnum\_l()* functions shall return non-zero if *wc* is an alphanumeric  
43396 wide-character code; otherwise, they shall return 0.43397 **ERRORS**

43398 No errors are defined.

43399 **EXAMPLES**

43400 None.

43401 **APPLICATION USAGE**43402 To ensure applications portability, especially across natural languages, only these functions and  
43403 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43404 classification.43405 **RATIONALE**

43406 None.

43407 **FUTURE DIRECTIONS**

43408 None.

43409 **SEE ALSO**43410 *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
43411 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43412 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;stdio.h&gt;, &lt;wctype.h&gt;

43413 **CHANGE HISTORY**

43414 First released as a World-wide Portability Interface in Issue 4.

43415 **Issue 5**43416 The following change has been made in this version for alignment with  
43417 ISO/IEC 9899:1990/Amendment 1:1995 (E):



- 43418 • The SYNOPSIS has been changed to indicate that this function and associated data types  
43419 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

**Issue 6**

43420 The normative text is updated to avoid use of the term “must” for application requirements.  
43421

**Issue 7**

43422 The `iswalnum_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43423 API Set Part 4.  
43424

43425 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0309 [302], XSH/TC1-2008/0310 [283],  
43426 and XSH/TC1-2008/0311 [283] are applied.

43427 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0180 [685] is applied.

43428 **NAME**

43429 iswalpha, iswalpha\_l — test for an alphabetic wide-character code

43430 **SYNOPSIS**

```
43431 #include <wctype.h>
43432 int iswalpha(wint_t wc);
43433 CX int iswalpha_l(wint_t wc, locale_t locale);
```

43434 **DESCRIPTION**

43435 CX For *iswalpha()*: The functionality described on this reference page is aligned with the ISO C  
 43436 standard. Any conflict between the requirements described here and the ISO C standard is  
 43437 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

43438 CX The *iswalpha()* and *iswalpha\_l()* functions shall test whether *wc* is a wide-character code  
 43439 CX representing a character of class **alpha** in the current locale, or in the locale represented by *locale*,  
 43440 respectively; see XBD Chapter 7 (on page 127).

43441 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 43442 code corresponding to a valid character in the locale used by the function, or equal to the value  
 43443 of the macro WEOF. If the argument has any other value, the behavior is undefined.

43444 CX The behavior is undefined if the *locale* argument to *iswalpha\_l()* is the special locale object  
 43445 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

43446 **RETURN VALUE**

43447 CX The *iswalpha()* and *iswalpha\_l()* functions shall return non-zero if *wc* is an alphabetic wide-  
 43448 character code; otherwise, they shall return 0.

43449 **ERRORS**

43450 No errors are defined.

43451 **EXAMPLES**

43452 None.

43453 **APPLICATION USAGE**

43454 To ensure applications portability, especially across natural languages, only these functions and  
 43455 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 43456 classification.

43457 **RATIONALE**

43458 None.

43459 **FUTURE DIRECTIONS**

43460 None.

43461 **SEE ALSO**

43462 *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
 43463 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43464 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43465 **CHANGE HISTORY**

43466 First released in Issue 4.

43467 **Issue 5**

43468 The following change has been made in this version for alignment with  
 43469 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 43470                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
43471                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43472 **Issue 6**

43473                   The normative text is updated to avoid use of the term “must” for application requirements.

43474 **Issue 7**

43475                   The `iswalpha_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43476                   API Set Part 4.

43477                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0312 [302], XSH/TC1-2008/0313 [283],  
43478                   and XSH/TC1-2008/0314 [283] are applied.

43479                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0181 [685] is applied.

43480 **NAME**

43481 iswblank, iswblank\_l — test for a blank wide-character code

43482 **SYNOPSIS**

43483 #include &lt;wctype.h&gt;

43484 int iswblank(wint\_t wc);

43485 CX int iswblank\_l(wint\_t wc, locale\_t locale);

43486 **DESCRIPTION**43487 CX For *iswblank()*: The functionality described on this reference page is aligned with the ISO C  
43488 standard. Any conflict between the requirements described here and the ISO C standard is  
43489 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43490 CX The *iswblank()* and *iswblank\_l()* functions shall test whether *wc* is a wide-character code  
43491 CX representing a character of class **blank** in the current locale, or in the locale represented by  
43492 *locale*, respectively; see XBD Chapter 7 (on page 127).43493 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
43494 code corresponding to a valid character in the locale used by the function, or equal to the value  
43495 of the macro WEOF. If the argument has any other value, the behavior is undefined.43496 CX The behavior is undefined if the *locale* argument to *iswblank\_l()* is the special locale object  
43497 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43498 **RETURN VALUE**43499 CX The *iswblank()* and *iswblank\_l()* functions shall return non-zero if *wc* is a blank wide-character  
43500 code; otherwise, they shall return 0.43501 **ERRORS**

43502 No errors are defined.

43503 **EXAMPLES**

43504 None.

43505 **APPLICATION USAGE**43506 To ensure applications portability, especially across natural languages, only these functions and  
43507 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43508 classification.43509 **RATIONALE**

43510 None.

43511 **FUTURE DIRECTIONS**

43512 None.

43513 **SEE ALSO**43514 *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
43515 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43516 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43517 **CHANGE HISTORY**

43518 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

43519 **Issue 7**43520 The *iswblank\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
43521 API Set Part 4.

43522 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0315 [302], XSH/TC1-2008/0316 [283],  
43523 and XSH/TC1-2008/0317 [283] are applied.

43524 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0182 [685] is applied.

43525 **Issue 8**

43526 Austin Group Defect 1770 is applied, changing “*iswblank()* and *iswblank()* functions” to  
43527 “*iswblank()* and *iswblank\_l()* functions”.

43528 **NAME**

43529 iswcntrl, iswcntrl\_l — test for a control wide-character code

43530 **SYNOPSIS**

43531 #include &lt;wctype.h&gt;

43532 int iswcntrl(wint\_t wc);

43533 CX int iswcntrl\_l(wint\_t wc, locale\_t locale);

43534 **DESCRIPTION**43535 CX For *iswcntrl()*: The functionality described on this reference page is aligned with the ISO C  
43536 standard. Any conflict between the requirements described here and the ISO C standard is  
43537 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43538 CX The *iswcntrl()* and *iswcntrl\_l()* functions shall test whether *wc* is a wide-character code  
43539 CX representing a character of class **cntrl** in the current locale, or in the locale represented by *locale*,  
43540 respectively; see XBD Chapter 7 (on page 127).43541 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
43542 code corresponding to a valid character in the locale used by the function, or equal to the value  
43543 of the macro WEOF. If the argument has any other value, the behavior is undefined.43544 CX The behavior is undefined if the *locale* argument to *iswcntrl\_l()* is the special locale object  
43545 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43546 **RETURN VALUE**43547 CX The *iswcntrl()* and *iswcntrl\_l()* functions shall return non-zero if *wc* is a control wide-character  
43548 code; otherwise, they shall return 0.43549 **ERRORS**

43550 No errors are defined.

43551 **EXAMPLES**

43552 None.

43553 **APPLICATION USAGE**43554 To ensure applications portability, especially across natural languages, only these functions and  
43555 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43556 classification.43557 **RATIONALE**

43558 None.

43559 **FUTURE DIRECTIONS**

43560 None.

43561 **SEE ALSO**43562 *iswalnum()*, *iswalphabeta()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
43563 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43564 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43565 **CHANGE HISTORY**

43566 First released in Issue 4.

43567 **Issue 5**43568 The following change has been made in this version for alignment with  
43569 ISO/IEC 9899:1990/Amendment 1:1995 (E):

43570 • The SYNOPSIS has been changed to indicate that this function and associated data types  
43571 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43572 **Issue 6**

43573 The normative text is updated to avoid use of the term “must” for application requirements.

43574 **Issue 7**

43575 The `iswcntrl_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43576 API Set Part 4.

43577 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0318 [302], XSH/TC1-2008/0319 [283],  
43578 and XSH/TC1-2008/0320 [283] are applied.

43579 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0183 [685] is applied.

43580 **NAME**

43581 iswctype, iswctype\_l — test character for a specified class

43582 **SYNOPSIS**

```
43583 #include <wctype.h>
43584 int iswctype(wint_t wc, wctype_t charclass);
43585 CX int iswctype_l(wint_t wc, wctype_t charclass,
43586 locale_t locale);
```

43587 **DESCRIPTION**

43588 CX For *iswctype()*: The functionality described on this reference page is aligned with the ISO C  
 43589 standard. Any conflict between the requirements described here and the ISO C standard is  
 43590 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

43591 CX The *iswctype()* and *iswctype\_l()* functions shall determine whether the wide-character code *wc*  
 43592 CX has the character class *charclass*, returning true or false. The *iswctype()* and *iswctype\_l()*  
 43593 functions are defined on WEOF and wide-character codes corresponding to the valid character  
 43594 CX encodings in the current locale, or in the locale represented by *locale*, respectively. If the *wc*  
 43595 argument is not in the domain of the function, the result is undefined. If the value of *charclass* is  
 43596 invalid (that is, not obtained by a call to *wctype()* or *charclass* is invalidated by a subsequent call  
 43597 to *setlocale()* that has affected category *LC\_CTYPE*) the result is unspecified.

43598 CX The behavior is undefined if the *locale* argument to *iswctype\_l()* is the special locale object  
 43599 *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.

43600 **RETURN VALUE**

43601 CX The *iswctype()* and *iswctype\_l()* functions shall return non-zero (true) if and only if *wc* has the  
 43602 CX property described by *charclass*. If *charclass* is *(wctype\_t)0*, the *iswctype()* and *iswctype\_l()*  
 43603 functions shall return 0.

43604 **ERRORS**

43605 No errors are defined.

43606 **EXAMPLES**43607 **Testing for a Valid Character**

```
43608 #include <wctype.h>
43609 ...
43610 int yes_or_no;
43611 wint_t wc;
43612 wctype_t valid_class;
43613 ...
43614 if ((valid_class=wctype("vowel")) == (wctype_t)0)
43615     /* Invalid character class. */
43616     yes_or_no=iswctype(wc,valid_class);
```

43617 **APPLICATION USAGE**

43618 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",  
 43619 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard  
 43620 character classes. In the table below, the functions in the left column are equivalent to the  
 43621 functions in the right column.

43622	<i>iswalnum(wc)</i>	<i>iswctype(wc, wctype("alnum"))</i>
43623	<i>iswalnum_l(wc, locale)</i>	<i>iswctype_l(wc, wctype("alnum"), locale)</i>
43624	<i>iswalpha(wc)</i>	<i>iswctype(wc, wctype("alpha"))</i>



```

43625     iswalpha_l(wc, locale)  iswctype_l(wc, wctype("alpha"), locale)
43626     iswblank(wc)           iswctype(wc, wctype("blank"))
43627     iswblank_l(wc, locale) iswctype_l(wc, wctype("blank"), locale)
43628     iswcntrl(wc)          iswctype(wc, wctype("cntrl"))
43629     iswcntrl_l(wc, locale) iswctype_l(wc, wctype("cntrl"), locale)
43630     iswdigit(wc)         iswctype(wc, wctype("digit"))
43631     iswdigit_l(wc, locale) iswctype_l(wc, wctype("digit"), locale)
43632     iswgraph(wc)        iswctype(wc, wctype("graph"))
43633     iswgraph_l(wc, locale) iswctype_l(wc, wctype("graph"), locale)
43634     iswlower(wc)       iswctype(wc, wctype("lower"))
43635     iswlower_l(wc, locale) iswctype_l(wc, wctype("lower"), locale)
43636     iswprint(wc)       iswctype(wc, wctype("print"))
43637     iswprint_l(wc, locale) iswctype_l(wc, wctype("print"), locale)
43638     iswpunct(wc)       iswctype(wc, wctype("punct"))
43639     iswpunct_l(wc, locale) iswctype_l(wc, wctype("punct"), locale)
43640     iswspace(wc)      iswctype(wc, wctype("space"))
43641     iswspace_l(wc, locale) iswctype_l(wc, wctype("space"), locale)
43642     iswupper(wc)      iswctype(wc, wctype("upper"))
43643     iswupper_l(wc, locale) iswctype_l(wc, wctype("upper"), locale)
43644     iswxdigit(wc)     iswctype(wc, wctype("xdigit"))
43645     iswxdigit_l(wc, locale) iswctype_l(wc, wctype("xdigit"), locale)

```

**43646 RATIONALE**

43647 None.

**43648 FUTURE DIRECTIONS**

43649 None.

**43650 SEE ALSO**

43651 [iswalnum\(\)](#), [iswalpha\(\)](#), [iswcntrl\(\)](#), [iswdigit\(\)](#), [iswgraph\(\)](#), [iswlower\(\)](#), [iswprint\(\)](#), [iswpunct\(\)](#),  
43652 [iswspace\(\)](#), [iswupper\(\)](#), [iswxdigit\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#), [wctype\(\)](#)

43653 XBD [<locale.h>](#), [<wctype.h>](#)

**43654 CHANGE HISTORY**

43655 First released as World-wide Portability Interfaces in Issue 4.

**43656 Issue 5**

43657 The following change has been made in this version for alignment with  
43658 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 43659 • The SYNOPSIS has been changed to indicate that this function and associated data types  
43660 are now made visible by inclusion of the [<wctype.h>](#) header rather than [<wchar.h>](#).

**43661 Issue 6**

43662 The behavior of `charclass = (wctype_t)0` is now described.

43663 An example is added.

43664 A new function, [iswblank\(\)](#), is added to the list in the APPLICATION USAGE.

**43665 Issue 7**

43666 The [iswctype\\_l\(\)](#) function is added from The Open Group Technical Standard, 2006, Extended  
43667 API Set Part 4.

43668 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0321 [283] and XSH/TC1-2008/0322  
43669 [283] are applied.

43670 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0184 [799] and XSH/TC2-2008/0185  
43671 [799] are applied.

43672 **Issue 8**

43673 Austin Group Defect 1302 is applied, aligning the *iswctype()* function with the  
43674 ISO/IEC 9899:2018 standard.

43675 **NAME**

43676 iswdigit, iswdigit\_l — test for a decimal digit wide-character code

43677 **SYNOPSIS**

43678 #include &lt;wctype.h&gt;

43679 int iswdigit(wint\_t wc);

43680 CX int iswdigit\_l(wint\_t wc, locale\_t locale);

43681 **DESCRIPTION**43682 CX For *iswdigit()*: The functionality described on this reference page is aligned with the ISO C  
43683 standard. Any conflict between the requirements described here and the ISO C standard is  
43684 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43685 CX The *iswdigit()* and *iswdigit\_l()* functions shall test whether *wc* is a wide-character code  
43686 CX representing a character of class **digit** in the current locale, or in the locale represented by *locale*,  
43687 respectively; see XBD Chapter 7 (on page 127).43688 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
43689 code corresponding to a valid character in the locale used by the function, or equal to the value  
43690 of the macro WEOF. If the argument has any other value, the behavior is undefined.43691 CX The behavior is undefined if the *locale* argument to *iswdigit\_l()* is the special locale object  
43692 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43693 **RETURN VALUE**43694 CX The *iswdigit()* and *iswdigit\_l()* functions shall return non-zero if *wc* is a decimal digit wide-  
43695 character code; otherwise, they shall return 0.43696 **ERRORS**

43697 No errors are defined.

43698 **EXAMPLES**

43699 None.

43700 **APPLICATION USAGE**43701 To ensure applications portability, especially across natural languages, only these functions and  
43702 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43703 classification.43704 **RATIONALE**

43705 None.

43706 **FUTURE DIRECTIONS**

43707 None.

43708 **SEE ALSO**43709 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
43710 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43711 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43712 **CHANGE HISTORY**

43713 First released in Issue 4.

43714 **Issue 5**43715 The following change has been made in this version for alignment with  
43716 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 43717                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
43718                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43719 **Issue 6**

43720                   The normative text is updated to avoid use of the term “must” for application requirements.

43721 **Issue 7**

43722                   The `iswdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43723                   API Set Part 4.

43724                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0323 [302], XSH/TC1-2008/0324 [283],  
43725                   and XSH/TC1-2008/0325 [283] are applied.

43726                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0186 [685] is applied.

43727 **NAME**

43728 iswgraph, iswgraph\_l — test for a visible wide-character code

43729 **SYNOPSIS**

43730 #include &lt;wctype.h&gt;

43731 int iswgraph(wint\_t wc);

43732 CX int iswgraph\_l(wint\_t wc, locale\_t locale);

43733 **DESCRIPTION**43734 CX For *iswgraph()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43737 CX The *iswgraph()* and *iswgraph\_l()* functions shall test whether *wc* is a wide-character code representing a character of class **graph** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).43740 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.43743 CX The behavior is undefined if the *locale* argument to *iswgraph\_l()* is the special locale object LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43745 **RETURN VALUE**43746 CX The *iswgraph()* and *iswgraph\_l()* functions shall return non-zero if *wc* is a wide-character code with a visible representation; otherwise, they shall return 0.43748 **ERRORS**

43749 No errors are defined.

43750 **EXAMPLES**

43751 None.

43752 **APPLICATION USAGE**

43753 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

43756 **RATIONALE**

43757 None.

43758 **FUTURE DIRECTIONS**

43759 None.

43760 **SEE ALSO**43761 *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43763 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43764 **CHANGE HISTORY**

43765 First released in Issue 4.

43766 **Issue 5**43767 The following change has been made in this version for alignment with  
43768 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 43769                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
43770                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43771 **Issue 6**

43772                   The normative text is updated to avoid use of the term “must” for application requirements.

43773 **Issue 7**

43774                   The `iswgraph_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43775                   API Set Part 4.

43776                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0326 [302], XSH/TC1-2008/0327 [283],  
43777                   and XSH/TC1-2008/0328 [283] are applied.

43778                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0187 [685] is applied.

43779 **NAME**

43780 iswlower, iswlower\_l — test for a lowercase letter wide-character code

43781 **SYNOPSIS**

43782 #include &lt;wctype.h&gt;

43783 int iswlower(wint\_t wc);

43784 CX int iswlower\_l(wint\_t wc, locale\_t locale);

43785 **DESCRIPTION**43786 CX For *iswlower()*: The functionality described on this reference page is aligned with the ISO C  
43787 standard. Any conflict between the requirements described here and the ISO C standard is  
43788 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43789 CX The *iswlower()* and *iswlower\_l()* functions shall test whether *wc* is a wide-character code  
43790 CX representing a character of class **lower** in the current locale, or in the locale represented by  
43791 *locale*, respectively; see XBD Chapter 7 (on page 127).43792 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
43793 code corresponding to a valid character in the locale used by the function, or equal to the value  
43794 of the macro WEOF. If the argument has any other value, the behavior is undefined.43795 CX The behavior is undefined if the *locale* argument to *iswlower\_l()* is the special locale object  
43796 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43797 **RETURN VALUE**43798 CX The *iswlower()* and *iswlower\_l()* functions shall return non-zero if *wc* is a lowercase letter wide-  
43799 character code; otherwise, they shall return 0.43800 **ERRORS**

43801 No errors are defined.

43802 **EXAMPLES**

43803 None.

43804 **APPLICATION USAGE**43805 To ensure applications portability, especially across natural languages, only these functions and  
43806 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43807 classification.43808 **RATIONALE**

43809 None.

43810 **FUTURE DIRECTIONS**

43811 None.

43812 **SEE ALSO**43813 *iswalnum()*, *iswalphabeta()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswprint()*, *iswpunct()*,  
43814 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()* (on page 2328) 1

43815 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43816 **CHANGE HISTORY**

43817 First released in Issue 4.

43818 **Issue 5**43819 The following change has been made in this version for alignment with  
43820 ISO/IEC 9899:1990/Amendment 1:1995 (E):

43821                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
43822                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43823 **Issue 6**

43824                   The normative text is updated to avoid use of the term “must” for application requirements.

43825 **Issue 7**

43826                   The `iswlower_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43827                   API Set Part 4.

43828                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0329 [302], XSH/TC1-2008/0330 [283],  
43829                   and XSH/TC1-2008/0331 [283] are applied.

43830                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0188 [685] is applied.



43831 **NAME**

43832 iswprint, iswprint\_l — test for a printable wide-character code

43833 **SYNOPSIS**

43834 #include &lt;wctype.h&gt;

43835 int iswprint(wint\_t wc);

43836 CX int iswprint\_l(wint\_t wc, locale\_t locale);

43837 **DESCRIPTION**43838 CX For *iswprint()*: The functionality described on this reference page is aligned with the ISO C  
43839 standard. Any conflict between the requirements described here and the ISO C standard is  
43840 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43841 CX The *iswprint()* and *iswprint\_l()* functions shall test whether *wc* is a wide-character code  
43842 CX representing a character of class **print** in the current locale, or in the locale represented by *locale*,  
43843 respectively; see XBD Chapter 7 (on page 127).43844 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
43845 code corresponding to a valid character in the locale used by the function, or equal to the value  
43846 of the macro WEOF. If the argument has any other value, the behavior is undefined.43847 CX The behavior is undefined if the *locale* argument to *iswprint\_l()* is the special locale object  
43848 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43849 **RETURN VALUE**43850 CX The *iswprint()* and *iswprint\_l()* functions shall return non-zero if *wc* is a printable wide-  
43851 character code; otherwise, they shall return 0.43852 **ERRORS**

43853 No errors are defined.

43854 **EXAMPLES**

43855 None.

43856 **APPLICATION USAGE**43857 To ensure applications portability, especially across natural languages, only these functions and  
43858 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43859 classification.43860 **RATIONALE**

43861 None.

43862 **FUTURE DIRECTIONS**

43863 None.

43864 **SEE ALSO**43865 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswpunct()*,  
43866 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43867 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43868 **CHANGE HISTORY**

43869 First released in Issue 4.

43870 **Issue 5**43871 The following change has been made in this version for alignment with  
43872 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 43873                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
43874                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43875 **Issue 6**

43876                   The normative text is updated to avoid use of the term “must” for application requirements.

43877 **Issue 7**

43878                   The `iswprint_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43879                   API Set Part 4.

43880                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0332 [302], XSH/TC1-2008/0333 [283],  
43881                   and XSH/TC1-2008/0334 [283] are applied.

43882                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0189 [685] is applied.

43883 **NAME**

43884 iswpunct, iswpunct\_l — test for a punctuation wide-character code

43885 **SYNOPSIS**

```
43886 #include <wctype.h>
43887 int iswpunct(wint_t wc);
43888 CX int iswpunct_l(wint_t wc, locale_t locale);
```

43889 **DESCRIPTION**

43890 CX For *iswpunct()*: The functionality described on this reference page is aligned with the ISO C  
 43891 standard. Any conflict between the requirements described here and the ISO C standard is  
 43892 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

43893 CX The *iswpunct()* and *iswpunct\_l()* functions shall test whether *wc* is a wide-character code  
 43894 CX representing a character of class **punct** in the current locale, or in the locale represented by  
 43895 *locale*, respectively; see XBD Chapter 7 (on page 127).

43896 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
 43897 code corresponding to a valid character in the locale used by the function, or equal to the value  
 43898 of the macro WEOF. If the argument has any other value, the behavior is undefined.

43899 CX The behavior is undefined if the *locale* argument to *iswpunct\_l()* is the special locale object  
 43900 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

43901 **RETURN VALUE**

43902 CX The *iswpunct()* and *iswpunct\_l()* functions shall return non-zero if *wc* is a punctuation wide-  
 43903 character code; otherwise, they shall return 0.

43904 **ERRORS**

43905 No errors are defined.

43906 **EXAMPLES**

43907 None.

43908 **APPLICATION USAGE**

43909 To ensure applications portability, especially across natural languages, only these functions and  
 43910 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 43911 classification.

43912 **RATIONALE**

43913 None.

43914 **FUTURE DIRECTIONS**

43915 None.

43916 **SEE ALSO**

43917 *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
 43918 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43919 XBD Chapter 7 (on page 127), [<locale.h>](#), [<wctype.h>](#)

43920 **CHANGE HISTORY**

43921 First released in Issue 4.

43922 **Issue 5**

43923 The following change has been made in this version for alignment with  
 43924 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 43925                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
43926                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43927 **Issue 6**

43928                   The normative text is updated to avoid use of the term “must” for application requirements.

43929 **Issue 7**

43930                   The `iswpunct_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43931                   API Set Part 4.

43932                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0335 [302], XSH/TC1-2008/0336 [283],  
43933                   and XSH/TC1-2008/0337 [283] are applied.

43934                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0190 [685] is applied.

43935 **NAME**

43936 iswspace, iswspace\_l — test for a white-space wide-character code

43937 **SYNOPSIS**

43938 #include &lt;wctype.h&gt;

43939 int iswspace(wint\_t wc);

43940 CX int iswspace\_l(wint\_t wc, locale\_t locale);

43941 **DESCRIPTION**43942 CX For *iswspace()*: The functionality described on this reference page is aligned with the ISO C  
43943 standard. Any conflict between the requirements described here and the ISO C standard is  
43944 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43945 CX The *iswspace()* and *iswspace\_l()* functions shall test whether *wc* is a wide-character code  
43946 CX representing a character of class **space** in the current locale, or in the locale represented by *locale*,  
43947 respectively; see XBD Chapter 7 (on page 127).43948 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
43949 code corresponding to a valid character in the locale used by the function, or equal to the value  
43950 of the macro WEOF. If the argument has any other value, the behavior is undefined.43951 CX The behavior is undefined if the *locale* argument to *iswspace\_l()* is the special locale object  
43952 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.43953 **RETURN VALUE**43954 CX The *iswspace()* and *iswspace\_l()* functions shall return non-zero if *wc* is a white-space wide-  
43955 character code; otherwise, they shall return 0.43956 **ERRORS**

43957 No errors are defined.

43958 **EXAMPLES**

43959 None.

43960 **APPLICATION USAGE**43961 To ensure applications portability, especially across natural languages, only these functions and  
43962 the functions in the reference pages listed in the SEE ALSO section should be used for character  
43963 classification.43964 **RATIONALE**

43965 None.

43966 **FUTURE DIRECTIONS**

43967 None.

43968 **SEE ALSO**43969 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
43970 *iswpunct()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

43971 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

43972 **CHANGE HISTORY**

43973 First released in Issue 4.

43974 **Issue 5**43975 The following change has been made in this version for alignment with  
43976 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 43977                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
43978                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

43979 **Issue 6**

43980                   The normative text is updated to avoid use of the term “must” for application requirements.

43981 **Issue 7**

43982                   The `iswspace_l()` function is added from The Open Group Technical Standard, 2006, Extended  
43983                   API Set Part 4.

43984                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0338 [302], XSH/TC1-2008/0339 [283],  
43985                   and XSH/TC1-2008/0340 [283] are applied.

43986                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0191 [685] is applied.

43987 **NAME**

43988 iswupper, iswupper\_l — test for an uppercase letter wide-character code

43989 **SYNOPSIS**

43990 #include &lt;wctype.h&gt;

43991 int iswupper(wint\_t wc);

43992 CX int iswupper\_l(wint\_t wc, locale\_t locale);

43993 **DESCRIPTION**43994 CX For *iswupper()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.43997 CX The *iswupper()* and *iswupper\_l()* functions shall test whether *wc* is a wide-character code representing a character of class **upper** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 127).44000 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.44003 CX The behavior is undefined if the *locale* argument to *iswupper\_l()* is the special locale object LC\_GLOBAL\_LOCALE or is not a valid locale object handle.44005 **RETURN VALUE**44006 CX The *iswupper()* and *iswupper\_l()* functions shall return non-zero if *wc* is an uppercase letter wide-character code; otherwise, they shall return 0.44008 **ERRORS**

44009 No errors are defined.

44010 **EXAMPLES**

44011 None.

44012 **APPLICATION USAGE**

44013 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

44016 **RATIONALE**

44017 None.

44018 **FUTURE DIRECTIONS**

44019 None.

44020 **SEE ALSO**44021 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswxdigit()*, *setlocale()*, *uselocale()*

44023 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

44024 **CHANGE HISTORY**

44025 First released in Issue 4.

44026 **Issue 5**44027 The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):  
44028

- 44029                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
44030                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

44031 **Issue 6**

44032                   The normative text is updated to avoid use of the term “must” for application requirements.

44033 **Issue 7**

44034                   The `iswupper_l()` function is added from The Open Group Technical Standard, 2006, Extended  
44035                   API Set Part 4.

44036                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0341 [302], XSH/TC1-2008/0342 [283],  
44037                   and XSH/TC1-2008/0343 [283] are applied.

44038                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0192 [685] is applied.



44039 **NAME**

44040 iswxdigit, iswxdigit\_l — test for a hexadecimal digit wide-character code

44041 **SYNOPSIS**

44042 #include &lt;wctype.h&gt;

44043 int iswxdigit(wint\_t wc);

44044 CX int iswxdigit\_l(wint\_t wc, locale\_t locale);

44045 **DESCRIPTION**44046 CX For *iswxdigit()*: The functionality described on this reference page is aligned with the ISO C  
44047 standard. Any conflict between the requirements described here and the ISO C standard is  
44048 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.44049 CX The *iswxdigit()* and *iswxdigit\_l()* functions shall test whether *wc* is a wide-character code  
44050 CX representing a character of class **xdigit** in the current locale, or in the locale represented by  
44051 *locale*, respectively; see XBD Chapter 7 (on page 127).44052 The *wc* argument is a **wint\_t**, the value of which the application shall ensure is a wide-character  
44053 code corresponding to a valid character in the locale used by the function, or equal to the value  
44054 of the macro WEOF. If the argument has any other value, the behavior is undefined.44055 CX The behavior is undefined if the *locale* argument to *iswxdigit\_l()* is the special locale object  
44056 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.44057 **RETURN VALUE**44058 CX The *iswxdigit()* and *iswxdigit\_l()* functions shall return non-zero if *wc* is a hexadecimal digit  
44059 wide-character code; otherwise, they shall return 0.44060 **ERRORS**

44061 No errors are defined.

44062 **EXAMPLES**

44063 None.

44064 **APPLICATION USAGE**44065 To ensure applications portability, especially across natural languages, only these functions and  
44066 the functions in the reference pages listed in the SEE ALSO section should be used for character  
44067 classification.44068 **RATIONALE**

44069 None.

44070 **FUTURE DIRECTIONS**

44071 None.

44072 **SEE ALSO**44073 *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,  
44074 *iswpunct()*, *iswspace()*, *iswupper()*, *setlocale()*, *uselocale()*

44075 XBD Chapter 7 (on page 127), &lt;locale.h&gt;, &lt;wctype.h&gt;

44076 **CHANGE HISTORY**

44077 First released in Issue 4.

44078 **Issue 5**44079 The following change has been made in this version for alignment with  
44080 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 44081                   • The SYNOPSIS has been changed to indicate that this function and associated data types  
44082                   are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

44083 **Issue 6**

44084                   The normative text is updated to avoid use of the term “must” for application requirements.

44085 **Issue 7**

44086                   The `iswxdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended  
44087                   API Set Part 4.

44088                   POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0344 [302], XSH/TC1-2008/0345 [283],  
44089                   and XSH/TC1-2008/0346 [283] are applied.

44090                   POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0193 [685] is applied.

44091 **NAME**

44092 isxdigit, isxdigit\_l — test for a hexadecimal digit

44093 **SYNOPSIS**

```
44094 #include <ctype.h>
44095 int isxdigit(int c);
44096 CX int isxdigit_l(int c, locale_t locale);
```

44097 **DESCRIPTION**

44098 CX For *isxdigit()*: The functionality described on this reference page is aligned with the ISO C  
 44099 standard. Any conflict between the requirements described here and the ISO C standard is  
 44100 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

44101 CX The *isxdigit()* and *isxdigit\_l()* functions shall test whether *c* is a character of class **xdigit** in the  
 44102 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page  
 44103 127).

44104 The *c* argument is an **int**, the value of which the application shall ensure is a character  
 44105 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has  
 44106 any other value, the behavior is undefined.

44107 CX The behavior is undefined if the *locale* argument to *isxdigit\_l()* is the special locale object  
 44108 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

44109 **RETURN VALUE**

44110 CX The *isxdigit()* and *isxdigit\_l()* functions shall return non-zero if *c* is a hexadecimal digit;  
 44111 otherwise, they shall return 0.

44112 **ERRORS**

44113 No errors are defined.

44114 **EXAMPLES**

44115 None.

44116 **APPLICATION USAGE**

44117 To ensure applications portability, especially across natural languages, only these functions and  
 44118 the functions in the reference pages listed in the SEE ALSO section should be used for character  
 44119 classification.

44120 **RATIONALE**

44121 None.

44122 **FUTURE DIRECTIONS**

44123 None.

44124 **SEE ALSO**

44125 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,  
 44126 *isupper()*

44127 XBD Chapter 7 (on page 127), **<ctype.h>**

44128 **CHANGE HISTORY**

44129 First released in Issue 1. Derived from Issue 1 of the SVID.

44130 **Issue 6**

44131 The normative text is updated to avoid use of the term “must” for application requirements.

44132 **Issue 7**

44133

44134

The *isxdigit\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

44135

44136

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0347 [302], XSH/TC1-2008/0348 [283], and XSH/TC1-2008/0349 [283] are applied.

44137 **NAME**

44138           j0, j1, jn — Bessel functions of the first kind

44139 **SYNOPSIS**

```
44140 XSI      #include <math.h>
44141          double j0(double x);
44142          double j1(double x);
44143          double jn(int n, double x);
```

44144 **DESCRIPTION**

44145           The *j0()*, *j1()*, and *jn()* functions shall compute Bessel functions of *x* of the first kind of orders 0,  
44146           1, and *n*, respectively.

44147           An application wishing to check for error situations should set *errno* to zero and call  
44148           *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
44149           *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
44150           zero, an error has occurred.

44151 **RETURN VALUE**

44152           Upon successful completion, these functions shall return the relevant Bessel value of *x* of the  
44153           first kind.

44154           If the *x* argument is finite and too large in magnitude, or the correct result would cause  
44155 MXX       underflow and is not representable, a range error may occur, and the function shall return 0.0,  
44156           or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no  
44157           greater in magnitude than DBL\_MIN.

44158 MXX       If the correct result would cause underflow, and is representable, a range error may occur and  
44159           the correct value shall be returned.

44160 MXX       If *x* is +Inf, +0 shall be returned.

44161 MXX       If *x* is NaN, a NaN shall be returned.

44162 **ERRORS**

44163           These functions may fail if:

44164           Range Error       The value of *x* was too large in magnitude, or an underflow occurred.

44165                               If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
44166                               then *errno* shall be set to [ERANGE]. If the integer expression  
44167                               (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
44168                               floating-point exception shall be raised.

44169           No other errors shall occur.

44170 **EXAMPLES**

44171           None.

44172 **APPLICATION USAGE**

44173           On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
44174           MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

44175 **RATIONALE**

44176           None.

44177 **FUTURE DIRECTIONS**

44178 None.

44179 **SEE ALSO**44180 *feclearexcept()*, *fetetestexcept()*, *isnan()*, *y0()*

44181 XBD Section 4.23 (on page 109), &lt;math.h&gt;

44182 **CHANGE HISTORY**

44183 First released in Issue 1. Derived from Issue 1 of the SVID.

44184 **Issue 5**44185 The DESCRIPTION is updated to indicate how an application should check for an error. This  
44186 text was previously published in the APPLICATION USAGE section.44187 **Issue 6**

44188 The may fail [EDOM] error is removed for the case for NaN.

44189 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling  
44190 with the ISO/IEC 9899:1999 standard.44191 **Issue 7**

44192 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0350 [68] is applied.

44193 **Issue 8**44194 Austin Group Defect 714 is applied, changing the behavior of these functions for special cases to  
44195 be a better match for their mathematical behavior.

44196 **NAME**

44197       jrand48 — generate a uniformly distributed pseudo-random long signed integer

44198 **SYNOPSIS**

```
44199 XSI       #include <stdlib.h>  
44200       long jrand48(unsigned short xsubi[3]);
```

44201 **DESCRIPTION**44202       Refer to *drand48()*.

44203 **NAME**

44204 kill — send a signal to a process or a group of processes

44205 **SYNOPSIS**

```
44206 CX #include <signal.h>
44207 int kill(pid_t pid, int sig);
```

44208 **DESCRIPTION**

44209 The *kill()* function shall send a signal to a process or a group of processes specified by *pid*. The  
 44210 signal to be sent is specified by *sig* and is either one from the list given in **<signal.h>** or 0. If *sig* is  
 44211 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can  
 44212 be used to check the validity of *pid*.

44213 For a process to have permission to send a signal to a process designated by *pid*, unless the  
 44214 sending process has appropriate privileges, the real or effective user ID of the sending process  
 44215 shall match the real or saved set-user-ID of the receiving process.

44216 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

44217 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)  
 44218 whose process group ID is equal to the process group ID of the sender, and for which the process  
 44219 has permission to send a signal.

44220 If *pid* is -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for  
 44221 which the process has permission to send that signal.

44222 If *pid* is negative, but not -1, *sig* shall be sent to all processes (excluding an unspecified set of  
 44223 system processes) whose process group ID is equal to the absolute value of *pid*, and for which  
 44224 the process has permission to send a signal.

44225 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for  
 44226 the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait()* function  
 44227 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending  
 44228 thread before *kill()* returns.

44229 The user ID tests described above shall not be applied when sending SIGCONT to a process that  
 44230 is a member of the same session as the sending process.

44231 An implementation that provides extended security controls may impose further  
 44232 implementation-defined restrictions on the sending of signals, including the null signal. In  
 44233 particular, the system may deny the existence of some or all of the processes specified by *pid*.

44234 The *kill()* function is successful if the process has permission to send *sig* to any of the processes  
 44235 specified by *pid*. If *kill()* fails, no signal shall be sent.

44236 **RETURN VALUE**

44237 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 44238 indicate the error.

44239 **ERRORS**

44240 The *kill()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 44241 | [EINVAL] | The value of the <i>sig</i> argument is an invalid or unsupported signal number.  |
| 44242 | [EPERM]  | The process does not have permission to send the signal to any receiving process. |
| 44243 |          |   |



44244 [ESRCH] No process or process group can be found corresponding to that specified by  
44245 *pid*.

#### 44246 EXAMPLES

44247 None.

#### 44248 APPLICATION USAGE

44249 None.

#### 44250 RATIONALE

44251 The semantics for permission checking for *kill()* differed between System V and most other  
44252 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of  
44253 POSIX.1-2024 agree with System V. Specifically, a set-user-ID process cannot protect itself  
44254 against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice  
44255 allows the user who starts an application to send it signals even if it changes its effective user ID.  
44256 The other semantics give more power to an application that wants to protect itself from the user  
44257 who ran it.

44258 Some implementations provide semantic extensions to the *kill()* function when the absolute  
44259 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a  
44260 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not  
44261 included in this volume of POSIX.1-2024, although a conforming implementation could provide  
44262 such an extension.

44263 The unspecified processes to which a signal cannot be sent may include the scheduler or *init*.

44264 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the  
44265 calling process and that signal is not blocked, that signal would be delivered before *kill()*  
44266 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.  
44267 However, historical implementations that provide only the *signal()* function make only the  
44268 weaker guarantee in this volume of POSIX.1-2024, because they only deliver one signal each  
44269 time a process enters the kernel. Modifications to such implementations to support the  
44270 *sigaction()* function generally require entry to the kernel following return from a signal-catching  
44271 function, in order to restore the signal mask. Such modifications have the effect of satisfying the  
44272 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.  
44273 The standard developers considered making the stronger requirement except when *signal()* is  
44274 used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the  
44275 stronger requirement whenever possible. In practice, the weaker requirement is the same, except  
44276 in the rare case when two signals arrive during a very short window. This reasoning also applies  
44277 to a similar requirement for *sigprocmask()*.

44278 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID  
44279 security checks. This allows a job control shell to continue a job even if processes in the job have  
44280 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of  
44281 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any  
44282 process in the same session regardless of user ID security checks. This is less restrictive than BSD  
44283 in the sense that ancestor processes (in the same session) can now be the recipient. It is more  
44284 restrictive than BSD in the sense that descendant processes that form new sessions are now  
44285 subject to the user ID checks. A similar relaxation of security is not necessary for the other job  
44286 control signals since those signals are typically sent by the terminal driver in recognition of  
44287 special characters being typed; the terminal driver bypasses all security checks.

44288 In secure implementations, a process may be restricted from sending a signal to a process having  
44289 a different security label. In order to prevent the existence or nonexistence of a process from  
44290 being used as a covert channel, such processes should appear nonexistent to the sender; that is,  
44291 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

44292 Historical implementations varied on the result of a *kill()* with *pid* indicating a zombie process.  
 44293 Some indicated success on such a call (subject to permission checking), while others gave an  
 44294 error of [ESRCH]. Since the definition of process lifetime in this volume of POSIX.1-2024 covers  
 44295 zombie processes, the [ESRCH] error as described is inappropriate in this case and  
 44296 implementations that give this error do not conform. This means that an application cannot have  
 44297 a parent process check for termination of a particular child by sending it the null signal with  
 44298 *kill()*, but must instead use *waitpid()* or *waitid()*.

44299 There is some belief that the name *kill()* is misleading, since the function is not always intended  
 44300 to cause process termination. However, the name is common to all historical implementations,  
 44301 and any change would be in conflict with the goal of minimal changes to existing application  
 44302 code.

#### 44303 FUTURE DIRECTIONS

44304 None.

#### 44305 SEE ALSO

44306 *getpid()*, *raise()*, *setsid()*, *sig2str()*, *sigaction()*, *sigqueue()*, *wait()*

44307 XBD [<signal.h>](#), [<sys/types.h>](#)

#### 44308 CHANGE HISTORY

44309 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 44310 Issue 5

44311 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 44312 Issue 6

44313 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

44314 The following new requirements on POSIX implementations derive from alignment with the  
 44315 Single UNIX Specification:

- 44316 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-  
 44317 user-ID of the calling process is checked in place of its effective user ID. This is a FIPS  
 44318 requirement.
- 44319 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 44320 required for conforming implementations of previous POSIX specifications, it was not  
 44321 required for UNIX applications.
- 44322 • The behavior when *pid* is  $-1$  is now specified. It was previously explicitly unspecified in  
 44323 the POSIX.1-1988 standard.

44324 The normative text is updated to avoid use of the term “must” for application requirements.

44325 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/51 is applied, correcting the RATIONALE  
 44326 section.

#### 44327 Issue 7

44328 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0194 [765] is applied.

#### 44329 Issue 8

44330 Austin Group Defect 1138 is applied, adding *sig2str()* to the SEE ALSO section.

44331 **NAME**

44332 kill\_dependency — terminate a dependency chain

44333 **SYNOPSIS**

44334 #include &lt;stdatomic.h&gt;

44335 type kill\_dependency(type \*y);

44336 **DESCRIPTION**

44337 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
44338 conflict between the requirements described here and the ISO C standard is unintentional. This  
44339 volume of POSIX.1-2024 defers to the ISO C standard.

44340 Implementations that define the macro `__STDC_NO_ATOMICS__` need not provide the  
44341 `<stdatomic.h>` header nor support the `kill_dependency()` macro.

44342 The `kill_dependency()` macro shall terminate a dependency chain (see XBD [Section 4.15.1](#), on page  
44343 100). The argument shall not carry a dependency to the return value.

44344 **RETURN VALUE**44345 The `kill_dependency()` macro shall return the value of *y*.44346 **ERRORS**

44347 No errors are defined.

44348 **EXAMPLES**

44349 None.

44350 **APPLICATION USAGE**

44351 None.

44352 **RATIONALE**

44353 None.

44354 **FUTURE DIRECTIONS**

44355 None.

44356 **SEE ALSO**44357 XBD [Section 4.15.1](#), `<stdatomic.h>`44358 **CHANGE HISTORY**

44359 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

44360 **NAME**

44361 killpg — send a signal to a process group

44362 **SYNOPSIS**

```
44363 XSI #include <signal.h>
44364 int killpg(pid_t pgrp, int sig);
```

44365 **DESCRIPTION**44366 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.

44367 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or  
 44368 equal to 1, the behavior of *killpg()* is undefined.

44369 **RETURN VALUE**44370 Refer to *kill()*.44371 **ERRORS**44372 Refer to *kill()*.44373 **EXAMPLES**44374 **Sending a Signal to All Other Members of a Process Group**

44375 The following example shows how the calling process could send a signal to all other members  
 44376 of its process group. To prevent itself from receiving the signal it first makes itself immune to the  
 44377 signal by ignoring it.

```
44378 #include <signal.h>
44379 #include <unistd.h>
44380 ...
44381     if (signal(SIGUSR1, SIG_IGN) == SIG_ERR)
44382         /* Handle error */;
44383     if (killpg(getpgrp(), SIGUSR1) == -1)
44384         /* Handle error */;
```

44385 **APPLICATION USAGE**

44386 None.

44387 **RATIONALE**

44388 None.

44389 **FUTURE DIRECTIONS**

44390 None.

44391 **SEE ALSO**44392 *getpgid()*, *getpid()*, *kill()*, *raise()*44393 XBD [<signal.h>](#)44394 **CHANGE HISTORY**

44395 First released in Issue 4, Version 2.

44396 **Issue 5**

44397 Moved from X/OPEN UNIX extension to BASE.

44398 **Issue 6**

44399 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/52 is applied, adding the example to the  
44400 EXAMPLES section.

44401 **NAME**

44402       l64a — convert a 32-bit integer to a radix-64 ASCII string

44403 **SYNOPSIS**

```
44404 XSI       #include <stdlib.h>  
44405       char *l64a(long value);
```

44406 **DESCRIPTION**

44407       Refer to [a64l\(\)](#).

44408 **NAME**

44409 labs, llabs — return a long integer absolute value

44410 **SYNOPSIS**

44411 #include &lt;stdlib.h&gt;

44412 long labs(long i);

44413 long long llabs(long long i);

44414 **DESCRIPTION**

44415 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
44416 conflict between the requirements described here and the ISO C standard is unintentional. This  
44417 volume of POSIX.1-2024 defers to the ISO C standard.

44418 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*  
44419 function shall compute the absolute value of the **long long** integer operand *i*. If the result  
44420 cannot be represented, the behavior is undefined.

44421 **RETURN VALUE**44422 The *labs()* function shall return the absolute value of the **long** integer operand.44423 The *llabs()* function shall return the absolute value of the **long long** integer operand.44424 **ERRORS**

44425 No errors are defined.

44426 **EXAMPLES**

44427 None.

44428 **APPLICATION USAGE**

44429 Since POSIX.1 requires a two's complement representation of **long** and **long long**, the absolute  
44430 value of the negative integers with the largest magnitude {LONG\_MIN} and {LLONG\_MIN} are  
44431 not representable, thus *labs*(LONG\_MIN) and *llabs*(LLONG\_MIN) are undefined.

44432 **RATIONALE**

44433 None.

44434 **FUTURE DIRECTIONS**

44435 None.

44436 **SEE ALSO**44437 [abs\(\)](#)44438 XBD [<stdlib.h>](#)44439 **CHANGE HISTORY**

44440 First released in Issue 4. Derived from the ISO C standard.

44441 **Issue 6**44442 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.44443 **Issue 7**

44444 SD5-XSH-ERN-152 is applied, correcting the RETURN VALUE section.

44445 **Issue 8**

44446 Austin Group Defect 1108 is applied, changing the APPLICATION USAGE section.

44447 **NAME**

44448 lchown — change the owner and group of a symbolic link

44449 **SYNOPSIS**

44450 #include &lt;unistd.h&gt;

44451 int lchown(const char \*path, uid\_t owner, gid\_t group);

44452 **DESCRIPTION**

44453 The *lchown()* function shall be equivalent to *chown()*, except in the case where the named file is a  
 44454 symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file itself,  
 44455 while *chown()* changes the ownership of the file or directory to which the symbolic link refers.

44456 **RETURN VALUE**

44457 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to  
 44458 indicate an error.

44459 **ERRORS**44460 The *lchown()* function shall fail if:

- 44461 [EACCES] Search permission is denied on a component of the path prefix of *path*.
- 44462 [EINVAL] The owner or group ID is not a value supported by the implementation.
- 44463 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 44464 argument.
- 44465 [ENAMETOOLONG]  
 44466 The length of a component of a pathname is longer than {NAME\_MAX}.
- 44467 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 44468 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
 44469 directory nor a symbolic link to a directory, or the *path* argument contains at  
 44470 least one non-*<slash>* character and ends with one or more trailing *<slash>*  
 44471 characters and the last pathname component names an existing file that is  
 44472 neither a directory nor a symbolic link to a directory.
- 44473 [EPERM] The effective user ID does not match the owner of the file and the process does  
 44474 not have appropriate privileges.
- 44475 [EROFS] The file resides on a read-only file system.
- 44476 The *lchown()* function may fail if:
- 44477 [EIO] An I/O error occurred while reading or writing to the file system.
- 44478 [EINTR] A signal was caught during execution of the function.
- 44479 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 44480 resolution of the *path* argument.
- 44481 [ENAMETOOLONG]  
 44482 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 44483 symbolic link produced an intermediate result with a length that exceeds  
 44484 {PATH\_MAX}.



44485 **EXAMPLES**44486 **Changing the Current Owner of a File**

44487 The following example shows how to change the ownership of the symbolic link named  
44488 **/modules/pass1** to the user ID associated with ```jones``` and the group ID associated with ```cnd```.

44489 The numeric value for the user ID is obtained by using the `getpwnam()` function. The numeric  
44490 value for the group ID is obtained by using the `getgrnam()` function.

```
44491 #include <sys/types.h>
44492 #include <unistd.h>
44493 #include <pwd.h>
44494 #include <grp.h>

44495 struct passwd *pwd;
44496 struct group *grp;
44497 char          *path = "/modules/pass1";
44498 ...
44499 pwd = getpwnam("jones");
44500 grp = getgrnam("cnd");
44501 lchown(path, pwd->pw_uid, grp->gr_gid);
```

44502 **APPLICATION USAGE**

44503 On implementations which support symbolic links as directory entries rather than files, `lchown()`  
44504 may fail.

44505 **RATIONALE**

44506 None.

44507 **FUTURE DIRECTIONS**

44508 None.

44509 **SEE ALSO**

44510 [\*chown\(\)\*](#), [\*symlink\(\)\*](#)

44511 XBD [\*\*<unistd.h>\*\*](#)

44512 **CHANGE HISTORY**

44513 First released in Issue 4, Version 2.

44514 **Issue 5**

44515 Moved from X/OPEN UNIX extension to BASE.

44516 **Issue 6**

44517 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
44518 [ELOOP] error condition is added.

44519 The Open Group Base Resolution bwg2001-013 is applied, adding wording to the  
44520 APPLICATION USAGE.

44521 **Issue 7**

44522 Austin Group Interpretation 1003.1-2001 #143 is applied.

44523 The `lchown()` function is moved from the XSI option to the Base.

44524 The [EOPNOTSUPP] error is removed.

44525 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
44526 pathname exists but is not a directory or a symbolic link to a directory.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0351 [324] is applied.

44528 **NAME**

44529 lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

44530 **SYNOPSIS**

```
44531 XSI #include <stdlib.h>  
44532 void lcong48(unsigned short param[7]);
```

44533 **DESCRIPTION**44534 Refer to *drand48()*.

44535 **NAME**

44536 ldexp, ldexpf, ldexpl — load exponent of a floating-point number

44537 **SYNOPSIS**

```
44538 #include <math.h>
44539 double ldexp(double x, int exp);
44540 float ldexpf(float x, int exp);
44541 long double ldexpl(long double x, int exp);
```

44542 **DESCRIPTION**

44543 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 44544 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44545 volume of POSIX.1-2024 defers to the ISO C standard.

44546 These functions shall compute the quantity  $x * 2^{exp}$ .

44547 An application wishing to check for error situations should set *errno* to zero and call  
 44548 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 44549 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 44550 zero, an error has occurred.

44551 **RETURN VALUE**44552 Upon successful completion, these functions shall return  $x$  multiplied by 2, raised to the power  
44553 *exp*.

44554 If these functions would cause overflow, a range error shall occur and *ldexp*(*x*), *ldexpf*(*x*), and  
 44555 *ldexpl*(*x*) shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (according to the sign of  
 44556 *x*), respectively.

44557 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
 44558 MXX and *ldexp*(*x*), *ldexpf*(*x*), and *ldexpl*(*x*) shall return 0.0, or (if IEC 60559 Floating-Point is not  
 44559 supported) an implementation-defined value no greater in magnitude than DBL\_MIN,  
 44560 FLT\_MIN, and LDBL\_MIN, respectively.

44561 MX If *x* is NaN, a NaN shall be returned.44562 If *x* is  $\pm 0$  or  $\pm$ Inf, *x* shall be returned.44563 If *exp* is 0, *x* shall be returned.

44564 MXX If the correct value would cause underflow, and is representable, a range error may occur and  
 44565 the correct value shall be returned.

44566 **ERRORS**

44567 These functions shall fail if:

44568 Range Error The result overflows.

44569 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 44570 then *errno* shall be set to [ERANGE]. If the integer expression  
 44571 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 44572 floating-point exception shall be raised.

44573 These functions may fail if:

44574 Range Error The result underflows.

44575 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 44576 then *errno* shall be set to [ERANGE]. If the integer expression  
 44577 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow

44578 floating-point exception shall be raised.

44579 **EXAMPLES**  
44580 None.

44581 **APPLICATION USAGE**  
44582 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
44583 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

44584 **RATIONALE**  
44585 None.

44586 **FUTURE DIRECTIONS**  
44587 None.

44588 **SEE ALSO**  
44589 [feclearexcept\(\)](#), [fetetestexcept\(\)](#), [frexp\(\)](#), [isnan\(\)](#)  
44590 XBD Section 4.23 (on page 109), <[math.h](#)>

44591 **CHANGE HISTORY**  
44592 First released in Issue 1. Derived from Issue 1 of the SVID.

44593 **Issue 5**  
44594 The DESCRIPTION is updated to indicate how an application should check for an error. This  
44595 text was previously published in the APPLICATION USAGE section.

44596 **Issue 6**  
44597 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999  
44598 standard.  
44599 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
44600 revised to align with the ISO/IEC 9899:1999 standard.  
44601 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
44602 marked.

44603 **Issue 7**  
44604 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0352 [68] and XSH/TC1-2008/0353  
44605 [68] are applied.

44606 **NAME**

44607 ldiv, lldiv — compute quotient and remainder of a long division

44608 **SYNOPSIS**

44609 #include &lt;stdlib.h&gt;

44610 ldiv\_t ldiv(long numer, long denom);

44611 lldiv\_t lldiv(long long numer, long long denom);

44612 **DESCRIPTION**

44613 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 44614 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44615 volume of POSIX.1-2024 defers to the ISO C standard.

44616 These functions shall compute the quotient and remainder of the division of the numerator  
 44617 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**  
 44618 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude  
 44619 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is  
 44620 undefined; otherwise,  $quot * denom + rem$  shall equal *numer*.

44621 **RETURN VALUE**

44622 The *ldiv()* function shall return a structure of type **ldiv\_t**, comprising both the quotient and the  
 44623 remainder. The structure shall include the following members, in any order:

```
44624 long    quot;    /* Quotient */
44625 long    rem;     /* Remainder */
```

44626 The *lldiv()* function shall return a structure of type **lldiv\_t**, comprising both the quotient and the  
 44627 remainder. The structure shall include the following members, in any order:

```
44628 long long quot;  /* Quotient */
44629 long long rem;   /* Remainder */
```

44630 **ERRORS**

44631 No errors are defined.

44632 **EXAMPLES**

44633 None.

44634 **APPLICATION USAGE**

44635 None.

44636 **RATIONALE**

44637 None.

44638 **FUTURE DIRECTIONS**

44639 None.

44640 **SEE ALSO**44641 [div\(\)](#)44642 XBD [<stdlib.h>](#)44643 **CHANGE HISTORY**

44644 First released in Issue 4. Derived from the ISO C standard.

44645 **Issue 6**44646 The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.

44647 **NAME**

44648 le16toh, le32toh, le64toh — convert values between host and specified byte order

44649 **SYNOPSIS**

44650 #include &lt;endian.h&gt;

44651 uint16\_t le16toh(uint16\_t *little\_endian\_16bits*);44652 uint32\_t le32toh(uint32\_t *little\_endian\_32bits*);44653 uint64\_t le64toh(uint64\_t *little\_endian\_64bits*);44654 **DESCRIPTION**44655 Refer to *be16toh()*.

44656 **NAME**

44657 lfind — find entry in a linear search table

44658 **SYNOPSIS**

```
44659 XSI #include <search.h>
44660 void *lfind(const void *key, const void *base, size_t *nel,
44661           size_t width, int (*compar)(const void *, const void *));
```

44662 **DESCRIPTION**

44663 Refer to [lsearch\(\)](#).



44664 **NAME**

44665 lgamma, lgammaf, lgammal, signgam — log gamma function

44666 **SYNOPSIS**

```
44667 #include <math.h>
44668 double lgamma(double x);
44669 float lgammaf(float x);
44670 long double lgammal(long double x);
44671 XSI extern int signgam;
```

44672 **DESCRIPTION**

44673 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 44674 conflict between the requirements described here and the ISO C standard is unintentional. This  
 44675 volume of POSIX.1-2024 defers to the ISO C standard.

44676 These functions shall compute  $\log_e |\Gamma(x)|$  where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t} t^{x-1} dt$ . The argument  $x$   
 44677 need not be a non-positive integer ( $\Gamma(x)$  is defined over the reals, except the non-positive  
 44678 integers).

44679 XSI The sign of  $\Gamma(x)$  shall be returned in the external integer *signgam*. If  $x$  is NaN,  $-\text{Inf}$ , or a negative  
 44680 integer, the value of *signgam* is unspecified.

44681 If concurrent calls are made to these functions, the value of *signgam* is indeterminate.

44682 An application wishing to check for error situations should set *errno* to zero and call  
 44683 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 44684 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 44685 zero, an error has occurred.

44686 **RETURN VALUE**

44687 Upon successful completion, these functions shall return the logarithmic gamma of  $x$ .

44688 If  $x$  is a non-positive integer, a pole error shall occur and *lgamma()*, *lgammaf()*, and *lgammal()*  
 44689 shall return +HUGE\_VAL, +HUGE\_VALF, and +HUGE\_VALL, respectively.

44690 If the correct value would cause overflow, a range error shall occur and *lgamma()*, *lgammaf()*,  
 44691 and *lgammal()* shall return  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  $\pm\text{HUGE\_VALL}$  (having the same  
 44692 sign as the correct value), respectively.

44693 MX If  $x$  is NaN, a NaN shall be returned.

44694 If  $x$  is 1 or 2, +0 shall be returned.

44695 If  $x$  is  $\pm\text{Inf}$ , +Inf shall be returned.

44696 **ERRORS**

44697 These functions shall fail if:

44698 Pole Error The  $x$  argument is a negative integer or zero.

44699 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 44700 then *errno* shall be set to [ERANGE]. If the integer expression  
 44701 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 44702 floating-point exception shall be raised.

44703 Range Error The result overflows.  
 44704  
 44705 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 44706 then *errno* shall be set to [ERANGE]. If the integer expression  
 44707 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 floating-point exception shall be raised.

**EXAMPLES**

44708  
 44709 None.

**APPLICATION USAGE**

44711 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 44712 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

44713 If the value of *signgam* will be obtained after a call to *lgamma()*, *lgammaf()*, or *lgammal()*, in order  
 44714 to ensure that the value will not be altered by another call in a different thread, applications  
 44715 should either restrict calls to these functions to be from a single thread or use a lock such as a  
 44716 mutex or spin lock to protect a critical section starting before the function call and ending after  
 44717 the value of *signgam* has been obtained.

**RATIONALE**

44718 Earlier versions of this standard did not require *lgamma()*, *lgammaf()*, and *lgammal()* to be  
 44719 thread-safe because *signgam* was a global variable. They are now required to be thread-safe to  
 44720 align with the ISO C standard (which, since the introduction of threads in 2011, requires that  
 44721 they avoid data races), with the exception that they need not avoid data races when storing a  
 44722 value in the *signgam* variable. Since *signgam* is not specified by the ISO C standard, this  
 44723 exception is not a conflict with that standard.  
 44724

**FUTURE DIRECTIONS**

44725  
 44726 None.

**SEE ALSO**

44728 [exp\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

44729 XBD Section 4.23 (on page 109), [<math.h>](#)

**CHANGE HISTORY**

44730  
 44731 First released in Issue 3.

**Issue 5**

44732 The DESCRIPTION is updated to indicate how an application should check for an error. This  
 44733 text was previously published in the APPLICATION USAGE section.  
 44734

44735 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

**Issue 6**

44736 The *lgamma()* function is no longer marked as an extension.  
 44737

44738 The *lgammaf()* and *lgammal()* functions are added for alignment with the ISO/IEC 9899:1999  
 44739 standard.

44740 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 44741 revised to align with the ISO/IEC 9899:1999 standard.

44742 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 44743 marked.

44744 Functionality relating to the XSI option is marked.

44745 **Issue 7**

44746 Austin Group Interpretation 1003.1-2001 #156 is applied.

44747 The DESCRIPTION is clarified regarding the value of *signgam* when  $x$  is Nan,  $-\text{Inf}$ , or a negative  
44748 integer.

44749 **Issue 8**

44750 Austin Group Defect 1002 is applied, reinstating the requirement for the sign of  $\Gamma(x)$  to be  
44751 returned in *signgam*, which had been accidentally removed in Issue 7.

44752 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
44753 standard.

44754 **NAME**

44755 link, linkat — hard link one file to another file

44756 **SYNOPSIS**

44757 #include &lt;unistd.h&gt;

44758 int link(const char \*path1, const char \*path2);

44759 OH #include &lt;fcntl.h&gt;

44760 int linkat(int fd1, const char \*path1, int fd2,

44761 const char \*path2, int flag);

44762 **DESCRIPTION**44763 The *link()* function shall create a new hard link (directory entry) for the existing file, *path1*.

44764 The *path1* argument points to a pathname naming an existing file. The *path2* argument points to  
 44765 a pathname naming the new directory entry to be created. The *link()* function shall atomically  
 44766 create a new hard link for the existing file and the link count of the file shall be incremented by  
 44767 one.

44768 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the  
 44769 implementation supports using *link()* on directories.

44770 If *path1* names a symbolic link, it is implementation-defined whether *link()* follows the symbolic  
 44771 link, or creates a new hard link to the symbolic link itself.

44772 Upon successful completion, *link()* shall mark for update the last file status change timestamp of  
 44773 the file. Also, the last data modification and last file status change timestamps of the directory  
 44774 that contains the new entry shall be marked for update.

44775 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.

44776 The implementation may require that the calling process has permission to access the existing  
 44777 file.

44778 The *linkat()* function shall be equivalent to the *link()* function except that symbolic links shall be  
 44779 handled as specified by the value of *flag* (see below) and except in the case where either *path1* or  
 44780 *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the  
 44781 directory associated with the file descriptor *fd1* instead of the current working directory and  
 44782 similarly for *path2* and the file descriptor *fd2*. If the access mode of the open file description  
 44783 associated with the file descriptor is not O\_SEARCH, the function shall check whether directory  
 44784 searches are permitted using the current permissions of the directory underlying the file  
 44785 descriptor. If the access mode is O\_SEARCH, the function shall not perform the check.

44786 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 44787 in <fcntl.h>:

44788 AT\_SYMLINK\_FOLLOW

44789 If *path1* names a symbolic link, a new hard link for the target of the symbolic link is created.

44790 If *linkat()* is passed the special value AT\_FDCWD in the *fd1* or *fd2* parameter, the current  
 44791 working directory shall be used for the respective *path* argument. If both *fd1* and *fd2* have value  
 44792 AT\_FDCWD, the behavior shall be identical to a call to *link()*, except that symbolic links shall be  
 44793 handled as specified by the value of *flag*.

44794 If the AT\_SYMLINK\_FOLLOW flag is clear in the *flag* argument and the *path1* argument names a  
 44795 symbolic link, a new hard link is created for the symbolic link *path1* and not its target.

44796 **RETURN VALUE**

44797       Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
44798       return  $-1$  and set *errno* to indicate the error.

44799 **ERRORS**

44800       These functions shall fail if:

44801       [EACCES]       A component of either path prefix denies search permission, or the requested  
44802       link requires writing in a directory that denies write permission, or the calling  
44803       process does not have permission to access the existing file and this is required  
44804       by the implementation.

44805       [EEXIST]       The *path2* argument resolves to an existing directory entry or refers to a  
44806       symbolic link.

44807       [EILSEQ]       The last pathname component of *path2* is not a portable filename, and cannot  
44808       be created in the target directory.

44809       [ELOOP]       A loop exists in symbolic links encountered during resolution of the *path1* or  
44810       *path2* argument.

44811       [EMLINK]       The number of hard links to the file named by *path1* would exceed  
44812       {LINK\_MAX}.

44813       [ENAMETOOLONG]

44814       The length of a component of a pathname is longer than {NAME\_MAX}.

44815       [ENOENT]       A component of either path prefix does not exist; the file named by *path1* does  
44816       not exist; or *path1* or *path2* points to an empty string.

44817       [ENOENT] or [ENOTDIR]

44818       The *path1* argument names an existing non-directory file, and the *path2*  
44819       argument contains at least one non-`<slash>` character and ends with one or  
44820       more trailing `<slash>` characters. If *path2* without the trailing `<slash>`  
44821       characters would name an existing file, an [ENOENT] error shall not occur.

44822       [ENOSPC]       The directory to contain the link cannot be extended.

44823       [ENOTDIR]

44824       A component of either path prefix names an existing file that is neither a  
44825       directory nor a symbolic link to a directory, or the *path1* argument contains at  
44826       least one non-`<slash>` character and ends with one or more trailing `<slash>`  
44827       characters and the last pathname component names an existing file that is  
44828       neither a directory nor a symbolic link to a directory, or the *path1* argument  
44829       names an existing non-directory file and the *path2* argument names a  
44830       nonexistent file, contains at least one non-`<slash>` character, and ends with  
44830       one or more trailing `<slash>` characters.

44831       [EPERM]       The file named by *path1* is a directory and either the calling process does not  
44832       have appropriate privileges or the implementation prohibits using *link()* on  
44833       directories.

44834       [EROFS]       The requested link requires writing in a directory on a read-only file system.

44835       [EXDEV]       The file named by *path1* and the directory in which the directory entry named  
44836       by *path2* is to be created are on different file systems and the implementation  
44837       does not support hard links between file systems.

44838       The *linkat()* function shall fail if:

44839 [EACCES] The access mode of the open file description associated with *fd1* or *fd2* is not  
 44840 O\_SEARCH and the permissions of the directory underlying *fd1* or *fd2*,  
 44841 respectively, do not permit directory searches.

44842 [EBADF] The *path1* or *path2* argument does not specify an absolute path and the *fd1* or  
 44843 *fd2* argument, respectively, is neither AT\_FDCWD nor a valid file descriptor  
 44844 open for reading or searching.

44845 [ENOTDIR] The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*,  
 44846 respectively, is a file descriptor associated with a non-directory file.

44847 These functions may fail if:

44848 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 44849 resolution of the *path1* or *path2* argument.

44850 [ENAMETOOLONG]  
 44851 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 44852 symbolic link produced an intermediate result with a length that exceeds  
 44853 {PATH\_MAX}.

44854 The *linkat()* function may fail if:

44855 [EINVAL] The value of the *flag* argument is not valid.

## 44856 EXAMPLES

### 44857 Creating a Hard Link to a File

44858 The following example shows how to create an additional hard link to a file named  
 44859 **/home/cnd/mod1** by creating a new directory entry named **/modules/pass1**.

```
44860 #include <unistd.h>
44861 char *path1 = "/home/cnd/mod1";
44862 char *path2 = "/modules/pass1";
44863 int status;
44864 ...
44865 status = link (path1, path2);
```

### 44866 Creating a Hard Link to a File Within a Program

44867 In the following program example, the *link()* function hard links the **/etc/passwd** file (defined as  
 44868 **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the  
 44869 current password file. Then, after removing the current password file (defined as  
 44870 **PASSWDFILE**), the new password file is saved as the current password file using the *link()*  
 44871 function again.

```
44872 #include <unistd.h>
44873 #define LOCKFILE "/etc/ptmp"
44874 #define PASSWDFILE "/etc/passwd"
44875 #define SAVEFILE "/etc/opasswd"
44876 ...
44877 /* Save current password file */
44878 link (PASSWDFILE, SAVEFILE);
44879 /* Remove current password file. */
44880 unlink (PASSWDFILE);
```

```
44881      /* Save new password file as current password file. */
44882      link (LOCKFILE,PASSWDFILE);
```

#### 44883 APPLICATION USAGE

44884 Some implementations do allow hard links between file systems.

44885 If *path1* refers to a symbolic link, application developers should use *linkat()* with appropriate  
44886 flags to select whether or not the symbolic link should be resolved.

#### 44887 RATIONALE

44888 Creating additional hard links to a directory is restricted to the superuser in most historical  
44889 implementations because this capability may produce loops in the file hierarchy or otherwise  
44890 corrupt the file system. This volume of POSIX.1-2024 continues that philosophy by prohibiting  
44891 *link()* and *unlink()* from doing this. Other functions could do it if the implementor designed  
44892 such an extension.

44893 Some historical implementations allow hard linking of files on different file systems. Wording  
44894 was added to explicitly allow this optional behavior.

44895 The exception for cross-file system hard links is intended to apply only to links that are  
44896 programmatically indistinguishable from traditional hard links.

44897 The purpose of the *linkat()* function is to link files in directories other than the current working  
44898 directory without exposure to race conditions. Any part of the path of a file could be changed in  
44899 parallel to a call to *link()*, resulting in unspecified behavior. By opening a file descriptor for the  
44900 directory of both the existing file and the target location and using the *linkat()* function it can be  
44901 guaranteed that the both filenames are in the desired directories.

44902 Earlier versions of this standard specified only the *link()* function, and required it to behave like  
44903 *linkat()* with the `AT_SYMLINK_FOLLOW` flag. However, historical practice from SVR4 and  
44904 Linux kernels had *link()* behaving like *linkat()* with no flags, and many systems that attempted  
44905 to provide a conforming *link()* function did so in a way that was rarely used, and when it was  
44906 used did not conform to the standard (e.g., by not being atomic, or by dereferencing the  
44907 symbolic link incorrectly). Since applications could not rely on *link()* following symbolic links in  
44908 practice, the *linkat()* function was added taking a flag to specify the desired behavior for the  
44909 application.

44910 Implementations are encouraged to have *link()* and *linkat()* report an [EILSEQ] error if the file  
44911 named by *path2* did not previously exist, and the last component of that pathname contains any  
44912 bytes that have the encoded value of a <newline> character.

#### 44913 FUTURE DIRECTIONS

44914 None.

#### 44915 SEE ALSO

44916 [\*rename\(\)\*](#), [\*symlink\(\)\*](#), [\*unlink\(\)\*](#)

44917 XBD [`<fcntl.h>`](#), [`<unistd.h>`](#)

#### 44918 CHANGE HISTORY

44919 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 44920 Issue 6

44921 The following new requirements on POSIX implementations derive from alignment with the  
44922 Single UNIX Specification:

- 44923 • The [ELOOP] mandatory error condition is added.

- 44924           • A second [ENAMETOOLONG] is added as an optional error condition.
- 44925           The following changes were made to align with the IEEE P1003.1a draft standard:
- 44926           • An explanation is added of the action when *path2* refers to a symbolic link.
- 44927           • The [ELOOP] optional error condition is added.
- 44928   **Issue 7**
- 44929           Austin Group Interpretation 1003.1-2001 #143 is applied.
- 44930           SD5-XSH-ERN-93 is applied, adding RATIONALE.
- 44931           The *linkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- 44932
- 44933           Functionality relating to XSI STREAMS is marked obsolescent.
- 44934           Changes are made related to support for finegrained timestamps.
- 44935           The [EOPNOTSUPP] error is removed.
- 44936           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0354 [326], XSH/TC1-2008/0355 [461],
- 44937           XSH/TC1-2008/0356 [326], XSH/TC1-2008/0357 [324], XSH/TC1-2008/0358 [147,429],
- 44938           XSH/TC1-2008/0359 [277], XSH/TC1-2008/0360 [278], and XSH/TC1-2008/0361 [278] are
- 44939           applied.
- 44940           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0195 [873], XSH/TC2-2008/0196 [591],
- 44941           XSH/TC2-2008/0197 [817], XSH/TC2-2008/0198 [822], and XSH/TC2-2008/0199 [817] are
- 44942           applied.
- 44943   **Issue 8**
- 44944           Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of
- 44945           filenames containing any bytes that have the encoded value of a <newline> character.
- 44946           Austin Group Defect 293 is applied, adding the [EILSEQ] error.
- 44947           Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 44948           Austin Group Defect 1380 is applied, changing text using the term “link” in line with its
- 44949           updated definition and removing a paragraph from the RATIONALE section.



44950 **NAME**

44951       lio\_listio — list directed I/O

44952 **SYNOPSIS**

44953       #include &lt;aio.h&gt;

44954       int lio\_listio(int mode, struct aiocb \*restrict const list[restrict],  
44955                   int nent, struct sigevent \*restrict sig);44956 **DESCRIPTION**44957       The *lio\_listio()* function shall initiate a list of I/O requests with a single function call.44958       The *mode* argument takes one of the values LIO\_WAIT or LIO\_NOWAIT declared in <aio.h> and  
44959       determines whether the function returns when the I/O operations have been completed, or as  
44960       soon as the operations have been queued. If the *mode* argument is LIO\_WAIT, the function shall  
44961       wait until all I/O is complete and the *sig* argument shall be ignored.44962       If the *mode* argument is LIO\_NOWAIT, the function shall return immediately, and asynchronous  
44963       notification shall occur, according to the *sig* argument, when all the I/O operations complete. If  
44964       *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous  
44965       notification occurs as specified in [Section 2.4.1](#) (on page 513) when all the requests in *list* have  
44966       completed.44967       The I/O requests enumerated by *list* are submitted in an unspecified order.44968       The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.  
44969       The array may contain NULL elements, which shall be ignored.44970       If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list*  
44971       become illegal addresses before all asynchronous I/O completed and, if necessary, the  
44972       notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio\_buf*  
44973       member of the **aiocb** structure pointed to by the elements of the array *list* become illegal  
44974       addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed,  
44975       the behavior is undefined.44976       The *aio\_lio\_opcode* field of each **aiocb** structure specifies the operation to be performed. The  
44977       supported operations are LIO\_READ, LIO\_WRITE, and LIO\_NOP; these symbols are defined in  
44978       <aio.h>. The LIO\_NOP operation causes the list entry to be ignored. If the *aio\_lio\_opcode*  
44979       element is equal to LIO\_READ, then an I/O operation is submitted as if by a call to *aio\_read()*  
44980       with the *aio\_cbp* equal to the address of the **aiocb** structure. If the *aio\_lio\_opcode* element is equal to  
44981       LIO\_WRITE, then an I/O operation is submitted as if by a call to *aio\_write()* with the *aio\_cbp*  
44982       equal to the address of the **aiocb** structure.44983       The *aio\_fildes* member specifies the file descriptor on which the operation is to be performed.44984       The *aio\_buf* member specifies the address of the buffer to or from which the data is transferred.44985       The *aio\_nbytes* member specifies the number of bytes of data to be transferred.44986       The members of the **aiocb** structure further describe the I/O operation to be performed, in a  
44987       manner identical to that of the corresponding **aiocb** structure when used by the *aio\_read()* and  
44988       *aio\_write()* functions.44989       The *nent* argument specifies how many elements are members of the list; that is, the length of the  
44990       array.44991       The behavior of this function is altered according to the definitions of synchronized I/O data  
44992       integrity completion and synchronized I/O file integrity completion if synchronized I/O is  
44993       enabled on the file associated with *aio\_fildes*.

44994 For regular files, no data transfer shall occur past the offset maximum established in the open  
44995 file description associated with *aiocbp*→*aio\_fildes*.

44996 If *sig*→*sigev\_notify* is SIGEV\_THREAD and *sig*→*sigev\_notify\_attributes* is a non-null pointer and  
44997 the block pointed to by this pointer becomes an illegal address prior to all asynchronous I/O  
44998 being completed, then the behavior is undefined.

#### 44999 RETURN VALUE

45000 If the *mode* argument has the value LIO\_NOWAIT, the *lio\_listio()* function shall return the value  
45001 zero if the I/O operations are successfully queued; otherwise, the function shall return the value  
45002 -1 and set *errno* to indicate the error.

45003 If the *mode* argument has the value LIO\_WAIT, the *lio\_listio()* function shall return the value zero  
45004 when all the indicated I/O has completed successfully. Otherwise, *lio\_listio()* shall return a value  
45005 of -1 and set *errno* to indicate the error.

45006 In either case, the return value only indicates the success or failure of the *lio\_listio()* call itself,  
45007 not the status of the individual I/O requests. In some cases one or more of the I/O requests  
45008 contained in the list may fail. Failure of an individual request does not prevent completion of  
45009 any other individual request. To determine the outcome of each I/O request, the application  
45010 shall examine the error status associated with each **aiocb** control block. The error statuses so  
45011 returned are identical to those returned as the result of an *aio\_read()* or *aio\_write()* function.

#### 45012 ERRORS

45013 The *lio\_listio()* function shall fail if:

45014 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The  
45015 application may check the error status for each **aiocb** to determine the  
45016 individual request(s) that failed.

45017 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit  
45018 {AIO\_MAX} to be exceeded.

45019 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than  
45020 {AIO\_LISTIO\_MAX}.

45021 [EINTR] A signal was delivered while waiting for all I/O requests to complete during  
45022 an LIO\_WAIT operation. Note that, since each I/O operation invoked by  
45023 *lio\_listio()* may possibly provoke a signal when it completes, this error return  
45024 may be caused by the completion of one (or more) of the very I/O operations  
45025 being awaited. Outstanding I/O requests are not canceled, and the application  
45026 shall examine each list element to determine whether the request was  
45027 initiated, canceled, or completed.

45028 [EIO] One or more of the individual I/O operations failed. The application may  
45029 check the error status for each **aiocb** structure to determine the individual  
45030 request(s) that failed.

45031 If the *lio\_listio()* function succeeds or fails with errors of [EAGAIN], [EINTR], or [EIO], then  
45032 some of the I/O specified by the list may have been initiated. If the *lio\_listio()* function fails with  
45033 an error code other than [EAGAIN], [EINTR], or [EIO], no operations from the list shall have  
45034 been initiated. The I/O operation indicated by each list element can encounter errors specific to  
45035 the individual read or write function being performed. In this event, the error status for each  
45036 **aiocb** control block contains the associated error code. The error codes that can be set are the  
45037 same as would be set if the I/O operation had been initiated by an *aio\_read()* or *aio\_write()*  
45038 function, with the following additional error codes possible:

45039 [EAGAIN] The requested I/O operation was not queued due to resource limitations.

45040 [EINPROGRESS] The requested I/O is in progress.

#### 45041 EXAMPLES

45042 None.

#### 45043 APPLICATION USAGE

45044 None.

#### 45045 RATIONALE

45046 Although it may appear that there are inconsistencies in the specified circumstances for error  
 45047 codes, the [EIO] error condition applies when any circumstance relating to an individual  
 45048 operation makes that operation fail. This might be due to a badly formulated request (for  
 45049 example, the *aio\_lio\_opcode* field is invalid, and *aio\_error()* returns [EINVAL]) or might arise from  
 45050 application behavior (for example, the file descriptor is closed before the operation is initiated,  
 45051 and *aio\_error()* returns [EBADF]).

45052 The limitation on the set of error codes returned when operations from the list shall have been  
 45053 initiated enables applications to know when operations have been started and whether  
 45054 *aio\_error()* is valid for a specific operation.

#### 45055 FUTURE DIRECTIONS

45056 None.

#### 45057 SEE ALSO

45058 *aio\_read()*, *aio\_write()*, *aio\_error()*, *aio\_return()*, *aio\_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,  
 45059 *read()*

45060 XBD <[aio.h](#)>

#### 45061 CHANGE HISTORY

45062 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45063 Large File Summit extensions are added.

#### 45064 Issue 6

45065 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 45066 implementation does not support the Asynchronous Input and Output option.

45067 The *lio\_listio()* function is marked as part of the Asynchronous Input and Output option.

45068 The following new requirements on POSIX implementations derive from alignment with the  
 45069 Single UNIX Specification:

- 45070 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs  
 45071 past the offset maximum established in the open file description associated with  
 45072 *aiocbp->aio\_fildes*. This change is to support large files.
- 45073 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support  
 45074 large files.

45075 The normative text is updated to avoid use of the term “must” for application requirements.

45076 The **restrict** keyword is added to the *lio\_listio()* prototype for alignment with the  
 45077 ISO/IEC 9899:1999 standard.

45078 **Issue 6**

45079 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/53 is applied, adding new text for  
45080 symmetry with the *aio\_read()* and *aio\_write()* functions to the DESCRIPTION.

45081 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/54 is applied, adding text to the  
45082 DESCRIPTION making it explicit that the user is required to keep the structure pointed to by  
45083 *sig->sigev\_notify\_attributes* valid until the last asynchronous operation finished and the  
45084 notification has been sent.

45085 **Issue 7**

45086 The *lio\_listio()* function is moved from the Asynchronous Input and Output option to the Base.

45087 **Issue 8**

45088 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

45089 **NAME**

45090 listen — listen for socket connections and limit the queue of incoming connections

45091 **SYNOPSIS**

```
45092 #include <sys/socket.h>
45093 int listen(int socket, int backlog);
```

45094 **DESCRIPTION**

45095 The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as  
 45096 accepting connections.

45097 The *backlog* argument provides a hint to the implementation which the implementation shall use  
 45098 to limit the number of outstanding connections in the socket's listen queue. Implementations  
 45099 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*  
 45100 argument value shall result in a larger or equal length of the listen queue. Implementations shall  
 45101 support values of *backlog* up to SOMAXCONN, defined in **<sys/socket.h>**.

45102 The implementation may include incomplete connections in its listen queue. The limits on the  
 45103 number of incomplete connections and completed connections queued may be different.

45104 The implementation may have an upper limit on the length of the listen queue—either global or  
 45105 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

45106 If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it  
 45107 had been called with a *backlog* argument value of 0.

45108 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of  
 45109 the listen queue may be set to an implementation-defined minimum value.

45110 The socket in use may require the process to have appropriate privileges to use the *listen()*  
 45111 function.

45112 **RETURN VALUE**

45113 Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set  
 45114 to indicate the error.

45115 **ERRORS**

45116 The *listen()* function shall fail if:

45117 [EBADF] The *socket* argument is not a valid file descriptor.

45118 [EDESTADDRREQ]

45119 The socket is not bound to a local address, and the protocol does not support  
 45120 listening on an unbound socket.

45121 [EINVAL] The *socket* is already connected.

45122 [ENOTSOCK] The *socket* argument does not refer to a socket.

45123 [EOPNOTSUPP] The socket protocol does not support *listen()*.

45124 The *listen()* function may fail if:

45125 [EACCES] The calling process does not have appropriate privileges.

45126 [EINVAL] The *socket* has been shut down.

45127 [ENOBUFS] Insufficient resources are available in the system to complete the call.

45128 **EXAMPLES**

45129       None.

45130 **APPLICATION USAGE**

45131       None.

45132 **RATIONALE**

45133       None.

45134 **FUTURE DIRECTIONS**

45135       None.

45136 **SEE ALSO**

45137       [\*accept\(\)\*](#), [\*connect\(\)\*](#), [\*socket\(\)\*](#)

45138       XBD <[\*sys/socket.h\*](#)>

45139 **CHANGE HISTORY**

45140       First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

45141       The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*  
45142       argument.

45143 **NAME**

45144 llabs — return a long integer absolute value

45145 **SYNOPSIS**

45146 #include <stdlib.h>

45147 long long llabs(long long i);

45148 **DESCRIPTION**

45149 Refer to *labs()*.

45150 **NAME**

45151        `lldiv` — compute quotient and remainder of a long division

45152 **SYNOPSIS**

45153        `#include <stdlib.h>`

45154        `lldiv_t lldiv(long long numer, long long denom);`

45155 **DESCRIPTION**

45156        Refer to *ldiv()*.



45157 **NAME**

45158 llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

45159 **SYNOPSIS**

```
45160 #include <math.h>
45161 long long llrint(double x);
45162 long long llrintf(float x);
45163 long long llrintl(long double x);
```

45164 **DESCRIPTION**

45165 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45166 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45167 volume of POSIX.1-2024 defers to the ISO C standard.

45168 These functions shall round their argument to the nearest integer value, rounding according to  
 45169 the current rounding direction.

45170 An application wishing to check for error situations should set *errno* to zero and call  
 45171 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 45172 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 45173 zero, an error has occurred.

45174 **RETURN VALUE**

45175 Upon successful completion, these functions shall return the rounded integer value.

45176 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.45177 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.45178 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

45179 If the correct value is positive and too large to represent as a **long long**, an unspecified value  
 45180 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 45181 CX shall occur; otherwise, a domain error may occur.

45182 If the correct value is negative and too large to represent as a **long long**, an unspecified value  
 45183 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 45184 CX shall occur; otherwise, a domain error may occur.

45185 **ERRORS**

45186 These functions shall fail if:

45187 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 45188 integer.

45189 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45190 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 45191 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 45192 shall be raised.

45193 These functions may fail if:

45194 **Domain Error** The correct value is not representable as an integer.

45195 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45196 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 45197 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 45198 shall be raised.

45199 **EXAMPLES**

45200 None.

45201 **APPLICATION USAGE**

45202 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
45203 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

45204 **RATIONALE**

45205 These functions provide floating-to-integer conversions. They round according to the current  
45206 rounding direction. If the rounded value is outside the range of the return type, the numeric  
45207 result is unspecified and the invalid floating-point exception is raised. When they raise no other  
45208 floating-point exception and the result differs from the argument, they raise the inexact floating-  
45209 point exception.

45210 **FUTURE DIRECTIONS**

45211 None.

45212 **SEE ALSO**45213 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [lrint\(\)](#)45214 XBD Section 4.23 (on page 109), [<math.h>](#)45215 **CHANGE HISTORY**

45216 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

45217 **Issue 7**

45218 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #53 is applied.

45219 **NAME**

45220 llround, llroundf, llroundl — round to nearest integer value

45221 **SYNOPSIS**

```
45222 #include <math.h>
45223 long long llround(double x);
45224 long long llroundf(float x);
45225 long long llroundl(long double x);
```

45226 **DESCRIPTION**

45227 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45228 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45229 volume of POSIX.1-2024 defers to the ISO C standard.

45230 These functions shall round their argument to the nearest integer value, rounding halfway cases  
 45231 away from zero, regardless of the current rounding direction.

45232 An application wishing to check for error situations should set *errno* to zero and call  
 45233 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 45234 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 45235 zero, an error has occurred.

45236 **RETURN VALUE**

45237 Upon successful completion, these functions shall return the rounded integer value.

45238 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.45239 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.45240 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

45241 If the correct value is positive and too large to represent as a **long long**, an unspecified value  
 45242 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 45243 CX shall occur; otherwise, a domain error may occur.

45244 If the correct value is negative and too large to represent as a **long long**, an unspecified value  
 45245 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error  
 45246 CX shall occur; otherwise, a domain error may occur.

45247 **ERRORS**

45248 These functions shall fail if:

45249 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 45250 integer.

45251 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45252 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 45253 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 45254 shall be raised.

45255 These functions may fail if:

45256 **Domain Error** The correct value is not representable as an integer.

45257 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45258 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 45259 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 45260 shall be raised.

45261 **EXAMPLES**

45262           None.

45263 **APPLICATION USAGE**

45264           On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
45265           MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

45266 **RATIONALE**

45267           These functions differ from the *llrint()* functions in that the default rounding direction for the  
45268           *llround()* functions round halfway cases away from zero and need not raise the inexact floating-  
45269           point exception for non-integer arguments that round to within the range of the return type.

45270 **FUTURE DIRECTIONS**

45271           None.

45272 **SEE ALSO**

45273           *feclearexcept()*, *fetestexcept()*, *lround()*

45274           XBD Section 4.23 (on page 109), <[math.h](#)>

45275 **CHANGE HISTORY**

45276           First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

45277 **Issue 7**

45278           ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

45279 **NAME**

45280 localeconv — return locale-specific information

45281 **SYNOPSIS**

45282 #include &lt;locale.h&gt;

45283 struct lconv \*localeconv(void);

45284 **DESCRIPTION**

45285 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
45286 conflict between the requirements described here and the ISO C standard is unintentional. This  
45287 volume of POSIX.1-2024 defers to the ISO C standard.

45288 The *localeconv()* function shall set the components of an object with the type **struct lconv** with  
45289 the values appropriate for the formatting of numeric quantities (monetary and otherwise)  
45290 according to the rules of the current locale.

45291 The members of the structure with type **char \*** are pointers to strings, any of which (except  
45292 **decimal\_point**) can point to " ", to indicate that the value is not available in the current locale or  
45293 is of zero length. The members with type **char** are non-negative numbers, any of which can be  
45294 {CHAR\_MAX} to indicate that the value is not available in the current locale.

45295 The members include the following:

45296 **char \*decimal\_point**

45297 The radix character used to format non-monetary quantities.

45298 **char \*thousands\_sep**

45299 The character used to separate groups of digits before the decimal-point character in  
45300 formatted non-monetary quantities.

45301 **char \*grouping**

45302 A string whose elements taken as one-byte integer values indicate the size of each group of  
45303 digits in formatted non-monetary quantities.

45304 **char \*int\_curr\_symbol**

45305 The international currency symbol applicable to the current locale. The first three characters  
45306 contain the alphabetic international currency symbol in accordance with those specified in  
45307 the ISO 4217:2015 standard. The fourth character (immediately preceding the null byte) is  
45308 the character used to separate the international currency symbol from the monetary  
45309 quantity.

45310 **char \*currency\_symbol**

45311 The local currency symbol applicable to the current locale.

45312 **char \*mon\_decimal\_point**

45313 The radix character used to format monetary quantities.

45314 **char \*mon\_thousands\_sep**

45315 The separator for groups of digits before the decimal-point in formatted monetary  
45316 quantities.

45317 **char \*mon\_grouping**

45318 A string whose elements taken as one-byte integer values indicate the size of each group of  
45319 digits in formatted monetary quantities.

45320 **char \*positive\_sign**

45321 The string used to indicate a non-negative valued formatted monetary quantity.

- 45322        **char \*negative\_sign**  
45323            The string used to indicate a negative valued formatted monetary quantity.
- 45324        **char int\_frac\_digits**  
45325            The number of fractional digits (those after the decimal-point) to be displayed in an  
45326            internationally formatted monetary quantity.
- 45327        **char frac\_digits**  
45328            The number of fractional digits (those after the decimal-point) to be displayed in a  
45329            formatted monetary quantity.
- 45330        **char p\_cs\_precedes**  
45331            Set to 1 if the **currency\_symbol** precedes the value for a non-negative formatted monetary  
45332            quantity. Set to 0 if the symbol succeeds the value.
- 45333        **char p\_sep\_by\_space**  
45334            Set to a value indicating the separation of the **currency\_symbol**, the sign string, and the  
45335            value for a non-negative formatted monetary quantity.
- 45336        **char n\_cs\_precedes**  
45337            Set to 1 if the **currency\_symbol** precedes the value for a negative formatted monetary  
45338            quantity. Set to 0 if the symbol succeeds the value.
- 45339        **char n\_sep\_by\_space**  
45340            Set to a value indicating the separation of the **currency\_symbol**, the sign string, and the  
45341            value for a negative formatted monetary quantity.
- 45342        **char p\_sign\_posn**  
45343            Set to a value indicating the positioning of the **positive\_sign** for a non-negative formatted  
45344            monetary quantity.
- 45345        **char n\_sign\_posn**  
45346            Set to a value indicating the positioning of the **negative\_sign** for a negative formatted  
45347            monetary quantity.
- 45348        **char int\_p\_cs\_precedes**  
45349            Set to 1 or 0 if the **int\_curr\_symbol** respectively precedes or succeeds the value for a non-  
45350            negative internationally formatted monetary quantity.
- 45351        **char int\_n\_cs\_precedes**  
45352            Set to 1 or 0 if the **int\_curr\_symbol** respectively precedes or succeeds the value for a  
45353            negative internationally formatted monetary quantity.
- 45354        **char int\_p\_sep\_by\_space**  
45355            Set to a value indicating the separation of the **int\_curr\_symbol**, the sign string, and the  
45356            value for a non-negative internationally formatted monetary quantity.
- 45357        **char int\_n\_sep\_by\_space**  
45358            Set to a value indicating the separation of the **int\_curr\_symbol**, the sign string, and the  
45359            value for a negative internationally formatted monetary quantity.
- 45360        **char int\_p\_sign\_posn**  
45361            Set to a value indicating the positioning of the **positive\_sign** for a non-negative  
45362            internationally formatted monetary quantity.
- 45363        **char int\_n\_sign\_posn**  
45364            Set to a value indicating the positioning of the **negative\_sign** for a negative internationally  
45365            formatted monetary quantity.

- 45366 The elements of **grouping** and **mon\_grouping** are interpreted according to the following:
- 45367 {CHAR\_MAX} No further grouping is to be performed.
- 45368 0 The previous element is to be repeatedly used for the remainder of the digits.
- 45369 *other* The integer value is the number of digits that comprise the current group. The  
45370 next element is examined to determine the size of the next group of digits  
45371 before the current group.
- 45372 The values of **p\_sep\_by\_space**, **n\_sep\_by\_space**, **int\_p\_sep\_by\_space**, and **int\_n\_sep\_by\_space**  
45373 are interpreted according to the following:
- 45374 0 No space separates the currency symbol and value.
- 45375 1 If the currency symbol and sign string are adjacent, a space separates them from the value;  
45376 otherwise, a space separates the currency symbol from the value.
- 45377 2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a  
45378 space separates the sign string from the value.
- 45379 For **int\_p\_sep\_by\_space** and **int\_n\_sep\_by\_space**, the fourth character of **int\_curr\_symbol** is  
45380 used instead of a space.
- 45381 The values of **p\_sign\_posn**, **n\_sign\_posn**, **int\_p\_sign\_posn**, and **int\_n\_sign\_posn** are  
45382 interpreted according to the following:
- 45383 0 Parentheses surround the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 45384 1 The sign string precedes the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 45385 2 The sign string succeeds the quantity and **currency\_symbol** or **int\_curr\_symbol**.
- 45386 3 The sign string immediately precedes the **currency\_symbol** or **int\_curr\_symbol**.
- 45387 4 The sign string immediately succeeds the **currency\_symbol** or **int\_curr\_symbol**.
- 45388 The implementation shall behave as if no function in this volume of POSIX.1-2024 calls  
45389 *localeconv()*.
- 45390 The *localeconv()* function need not be thread-safe; however, *localeconv()* shall avoid data races  
45391 with all other functions.

#### 45392 RETURN VALUE

- 45393 The *localeconv()* function shall return a pointer to the filled-in object. The application shall not  
45394 CX modify the structure to which the return value points, nor any storage areas pointed to by  
45395 pointers within the structure. The returned pointer, and pointers within the structure, might be  
45396 CX invalidated or the structure or the storage areas might be overwritten by a subsequent call to  
45397 CX *localeconv()*. In addition, the returned pointer, and pointers within the structure, might be  
45398 CX invalidated or the structure or the storage areas might be overwritten by subsequent calls to  
45399 CX *setlocale()* with the categories LC\_ALL, LC\_MONETARY, or LC\_NUMERIC, or by calls to  
45400 *uselocale()* which change the categories LC\_MONETARY or LC\_NUMERIC. The returned  
45401 pointer, pointers within the structure, the structure, and the storage areas might also be  
45402 invalidated if the calling thread is terminated.

#### 45403 ERRORS

- 45404 No errors are defined.

45405 **EXAMPLES**

45406 None.

45407 **APPLICATION USAGE**

45408 The following table illustrates the rules which may be used by four countries to format  
 45409 monetary quantities.

Country	Positive Format	Negative Format	International Format
Italy	€1.230	-€1.230	EUR.1.230
Netherlands	€ 1.234,56	€ -1.234,56	EUR 1.234,56
Norway	kr1.234,56	kr1.234,56-	NOK 1.234,56
Switzerland	SFrs.1,234.56	SFrs.1,234.56C	CHF 1,234.56

45415 For these four countries, the respective values for the monetary members of the structure  
 45416 returned by *localeconv()* are:

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	"EUR. "	"EUR "	"NOK "	"CHF "
currency_symbol	"€. "	"€"	"kr"	"SFrs. "
mon_decimal_point	" "	","	","	."
mon_thousands_sep	"."	"."	"."	","
mon_grouping	"\3"	"\3"	"\3"	"\3"
positive_sign	""	""	""	""
negative_sign	"-"	"-"	"-"	"C"
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2
int_p_cs_precedes	1	1	1	1
int_n_cs_precedes	1	1	1	1
int_p_sep_by_space	0	0	0	0
int_n_sep_by_space	0	0	0	0
int_p_sign_posn	1	1	1	1
int_n_sign_posn	1	4	4	2

45439 **RATIONALE**

45440 None.

45441 **FUTURE DIRECTIONS**

45442 None.

45443 **SEE ALSO**

45444 *fprintf()*, *fscanf()*, *isalpha()*, *nl\_langinfo()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*,  
 45445 *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, *uselocale()*

45446 XBD &lt;langinfo.h&gt;, &lt;locale.h&gt;



45447 **CHANGE HISTORY**

45448 First released in Issue 4. Derived from the ANSI C standard.

45449 **Issue 6**

45450 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

45451 The RETURN VALUE section is rewritten to avoid use of the term “must”.

45452 This reference page is updated for alignment with the ISO/IEC 9899:1999 standard.

45453 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

45454 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to **int\_curr\_symbol** and updating the descriptions of **p\_sep\_by\_space** and **n\_sep\_by\_space**. These  
45455 changes are for alignment with the ISO C standard.  
45456

45457 **Issue 7**

45458 Austin Group Interpretation 1003.1-2001 #156 is applied.

45459 The definitions of **int\_curr\_symbol** and **currency\_symbol** are updated.

45460 The examples in the APPLICATION USAGE section are updated.

45461 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0362 [75] is applied.

45462 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0200 [656] is applied.

45463 **Issue 8**

45464 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
45465 standard.

45466 **NAME**

45467 localtime, localtime\_r — convert a time value to a broken-down local time

45468 **SYNOPSIS**

```
45469 #include <time.h>
45470 struct tm *localtime(const time_t *timer);
45471 CX struct tm *localtime_r(const time_t *restrict timer,
45472 struct tm *restrict result);
```

45473 **DESCRIPTION**

45474 CX For *localtime()*: The functionality described on this reference page is aligned with the ISO C  
 45475 standard. Any conflict between the requirements described here and the ISO C standard is  
 45476 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

45477 The *localtime()* function shall convert the time in seconds since the Epoch pointed to by *timer*  
 45478 into a broken-down time, expressed as a local time. The function corrects for the timezone and  
 45479 CX any seasonal time adjustments. Local timezone information shall be set as though *localtime()*  
 45480 calls *tzset()*.

45481 The relationship between a time in seconds since the Epoch used as an argument to *localtime()*  
 45482 and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in  
 45483 the expression given in the definition of seconds since the Epoch (see XBD [Section 4.19](#), on page  
 45484 107) corrected for timezone and any seasonal time adjustments, where the names in the structure  
 45485 and in the expression correspond.

45486 The same relationship shall apply for *localtime\_r()*.

45487 The *localtime()* function need not be thread-safe; however, *localtime()* shall avoid data races with  
 45488 all functions other than itself, *asctime()*, *ctime()*, and *gmtime()*.

45489 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static  
 45490 objects: a broken-down time structure and an array of type **char**. Execution of any of the  
 45491 functions that return a pointer to one of these object types may overwrite the information in any  
 45492 object of the same type pointed to by the value returned from any previous call to any of them.

45493 CX The *localtime\_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*  
 45494 into a broken-down time stored in the structure to which *result* points. The *localtime\_r()* function  
 45495 shall also return a pointer to that same structure.

45496 Unlike *localtime()*, the *localtime\_r()* function is not required to set *tzname*. If *localtime\_r()* sets  
 45497 *tzname*, it shall also set *daylight* and *timezone*. If *localtime\_r()* does not set *tzname*, it shall not set  
 45498 *daylight* and shall not set *timezone*. If the **tm** structure member *tm\_zone* is accessed after the value  
 45499 of *TZ* is subsequently modified, the behaviour is undefined.

45500 **RETURN VALUE**

45501 Upon successful completion, the *localtime()* function shall return a pointer to the broken-down  
 45502 CX time structure. If an error is detected, *localtime()* shall return a null pointer and set *errno* to  
 45503 indicate the error.

45504 Upon successful completion, *localtime\_r()* shall return a pointer to the structure pointed to by the  
 45505 argument *result*. If an error is detected, *localtime\_r()* shall return a null pointer and set *errno* to  
 45506 indicate the error.

45507 **ERRORS**45508 CX The `localtime()` and `localtime_r()` functions shall fail if:45509 CX **[EOVERFLOW]** The result cannot be represented.45510 **EXAMPLES**45511 **Getting the Local Date and Time**

45512 The following example uses the `time()` function to calculate the time elapsed, in seconds, since  
 45513 January 1, 1970 0:00 UTC (the Epoch), `localtime()` to convert that value to a broken-down time,  
 45514 and `asctime()` to convert the broken-down time values into a printable string.

```
45515 #include <stdio.h>
45516 #include <time.h>
45517 int main(void)
45518 {
45519     time_t result;
45520
45521     result = time(NULL);
45522     printf("%s%ju secs since the Epoch\n",
45523           asctime(localtime(&result)),
45524           (uintmax_t)result);
45525     return(0);
45526 }
```

45526 This example writes the current time to `stdout` in a form like this:

```
45527 Wed Jun 26 10:32:15 1996
45528 835810335 secs since the Epoch
```

45529 **Getting the Modification Time for a File**

45530 The following example prints the last data modification timestamp in the local timezone for a  
 45531 given file.

```
45532 #include <stdio.h>
45533 #include <time.h>
45534 #include <sys/stat.h>
45535 int
45536 print_file_time(const char *pathname)
45537 {
45538     struct stat statbuf;
45539     struct tm *tm;
45540     char timestr[BUFSIZ];
45541
45542     if(stat(pathname, &statbuf) == -1)
45543         return -1;
45544     if((tm = localtime(&statbuf.st_mtime)) == NULL)
45545         return -1;
45546     if(strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", tm) == 0)
45547         return -1;
45548     printf("%s: %s.%09ld\n", pathname, timestr, statbuf.st_mtim.tv_nsec);
45549     return 0;
45550 }
```

45550 **Timing an Event**

45551 The following example gets the current time, converts it to a string using *localtime()* and  
 45552 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 45553 an event being timed.

```
45554 #include <time.h>
45555 #include <stdio.h>
45556 ...
45557 time_t now;
45558 int minutes_to_event;
45559 ...
45560 time(&now);
45561 printf("The time is ");
45562 fputs(asctime(localtime(&now)), stdout);
45563 printf("There are still %d minutes to the event.\n",
45564       minutes_to_event);
45565 ...
```

45566 **APPLICATION USAGE**

45567 The *localtime\_r()* function is thread-safe and returns values in a user-supplied buffer instead of  
 45568 possibly using a static data area that may be overwritten by each call.

45569 **RATIONALE**

45570 None.

45571 **FUTURE DIRECTIONS**

45572 None.

45573 **SEE ALSO**

45574 [asctime\(\)](#), [clock\(\)](#), [ctime\(\)](#), [difftime\(\)](#), [futimens\(\)](#), [getdate\(\)](#), [gmtime\(\)](#), [mktime\(\)](#), [strptime\(\)](#),  
 45575 [strptime\(\)](#), [time\(\)](#), [tzset\(\)](#)

45576 XBD Section 4.19 (on page 107), [<time.h>](#)

45577 **CHANGE HISTORY**

45578 First released in Issue 1. Derived from Issue 1 of the SVID.

45579 **Issue 5**

45580 A note indicating that the *localtime()* function need not be reentrant is added to the  
 45581 DESCRIPTION.

45582 The *localtime\_r()* function is included for alignment with the POSIX Threads Extension.

45583 **Issue 6**

45584 The *localtime\_r()* function is marked as part of the Thread-Safe Functions option.

45585 Extensions beyond the ISO C standard are marked.

45586 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
 45587 its avoidance of possibly using a static data area.

45588 The **restrict** keyword is added to the *localtime\_r()* prototype for alignment with the  
 45589 ISO/IEC 9899:1999 standard.

45590 Examples are added.

45591 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/32 is applied, adding the  
 45592 [EOverflow] error.

- 45593 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/55 is applied, updating the error handling  
45594 for *localtime\_r()*.
- 45595 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/56 is applied, adding a requirement that if  
45596 *localtime\_r()* does not set the *tzname* variable, it shall not set the *daylight* or *timezone* variables. On  
45597 systems supporting XSI, the *daylight*, *timezone*, and *tzname* variables should all be set to provide  
45598 information for the same timezone. This updates the description of *localtime\_r()* to mention  
45599 *daylight* and *timezone* as well as *tzname*. The SEE ALSO section is updated.
- 45600 **Issue 7**
- 45601 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 45602 The *localtime\_r()* function is moved from the Thread-Safe Functions option to the Base.
- 45603 Changes are made to the EXAMPLES section related to support for finegrained timestamps.
- 45604 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0363 [291] is applied.
- 45605 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0201 [664] is applied.
- 45606 **Issue 8**
- 45607 Austin Group Defect 1125 is applied, changing “Local timezone information is used” to “Local  
45608 timezone information shall be set”.
- 45609 Austin Group Defect 1302 is applied, aligning the *localtime()* function with the  
45610 ISO/IEC 9899:2018 standard.
- 45611 Austin Group Defect 1376 is applied, removing CX shading from some text derived from the  
45612 ISO C standard and updating it to match the ISO C standard.
- 45613 Austin Group Defect 1533 is applied, adding *tm\_gmtoff* and *tm\_zone* to the **tm** structure.
- 45614 Austin Group Defect 1570 is applied, removing extra spacing in "==".

45615 **NAME**

45616 lockf — record locking on files

45617 **SYNOPSIS**

```
45618 XSI #include <unistd.h>
45619 int lockf(int fildev, int function, off_t size);
```

45620 **DESCRIPTION**

45621 The *lockf()* function shall lock sections of a file with advisory-mode process-owned file locks.  
 45622 Calls to *lockf()* from threads in other processes which attempt to lock the locked file section shall  
 45623 either return an error value or block until the section becomes unlocked. All the locks for a  
 45624 process are removed when the process terminates. Record locking with *lockf()* shall be  
 45625 supported for regular files and may be supported for other files.

45626 The *fildev* argument is an open file descriptor. To establish a lock with this function, the file  
 45627 descriptor shall be opened with write-only permission (O\_WRONLY) or with read/write  
 45628 permission (O\_RDWR).

45629 The *function* argument is a control value which specifies the action to be taken. The permissible  
 45630 values for *function* are defined in <unistd.h> as follows:

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

45636 F\_TEST shall detect if a lock by another process is present on the specified section.

45637 F\_LOCK and F\_TLOCK shall both lock a section of a file if the section is available.

45638 F\_ULOCK shall remove locks from a section of the file.

45639 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be  
 45640 locked or unlocked starts at the current offset in the file and extends forward for a positive size  
 45641 or backward for a negative size (the preceding bytes up to but not including the current offset).  
 45642 If *size* is 0, the section from the current offset through the largest possible file offset shall be  
 45643 locked (that is, from the current offset through the present or any future end-of-file). An area  
 45644 need not be allocated to the file to be locked because locks may exist past the end-of-file.

45645 The sections locked with F\_LOCK or F\_TLOCK may, in whole or in part, contain or be contained  
 45646 by a previously locked section for the same process. When this occurs, or if adjacent locked  
 45647 sections would occur, the sections shall be combined into a single locked section. If the request  
 45648 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

45649 F\_LOCK and F\_TLOCK requests differ only by the action taken if the section is not available.  
 45650 F\_LOCK shall block the calling thread until the section is available. F\_TLOCK shall cause the  
 45651 function to fail if the section is already locked by another process.

45652 Process-owned file locks shall be released on first close by the locking process of any file  
 45653 descriptor for the file.

45654 F\_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the  
 45655 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or  
 45656 to the end-of-file if *size* is (off\_t)0. When all of a locked section is not released (that is, when the  
 45657 beginning or end of the area to be unlocked falls within a locked section), the remaining portions  
 45658 of that section shall remain locked by the process. Releasing the center portion of a locked

45659 section shall cause the remaining locked beginning and end portions to become two separate  
 45660 locked sections. If the request would cause the number of locks in the system to exceed a system-  
 45661 imposed limit, the request shall fail.

45662 A potential for deadlock occurs if the threads of a process controlling a locked section are  
 45663 blocked by accessing a locked section of another process. If the system detects that deadlock  
 45664 would occur, *lockf()* shall fail with an [EDEADLK] error.

45665 The interaction between *fcntl()* and *lockf()* locks is unspecified.

45666 Blocking on a section shall be interrupted by any signal.

45667 An F\_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested  
 45668 section is the maximum value for an object of type **off\_t**, when the process has an existing lock  
 45669 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a  
 45670 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an  
 45671 F\_ULOCK request shall attempt to unlock only the requested section.

45672 Attempting to lock a section of a file that is associated with a buffered stream produces  
 45673 unspecified results.

#### 45674 RETURN VALUE

45675 Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return  $-1$ , set *errno* to  
 45676 indicate an error, and existing locks shall not be changed.

#### 45677 ERRORS

45678 The *lockf()* function shall fail if:

45679 [EBADF] The *fildev* argument is not a valid open file descriptor; or *function* is F\_LOCK  
 45680 or F\_TLOCK and *fildev* is not a valid file descriptor open for writing.

45681 [EACCES] or [EAGAIN]  
 45682 The *function* argument is F\_TLOCK or F\_TEST and the section is already  
 45683 locked by another process.

45684 [EDEADLK] The *function* argument is F\_LOCK and a deadlock is detected.

45685 [EINTR] A signal was caught during execution of the function.

45686 [EINVAL] The *function* argument is not one of F\_LOCK, F\_TLOCK, F\_TEST, or  
 45687 F\_ULOCK; or *size* plus the current file offset is less than 0.

45688 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested  
 45689 section cannot be represented correctly in an object of type **off\_t**.

45690 The *lockf()* function may fail if:

45691 [EAGAIN] The *function* argument is F\_LOCK or F\_TLOCK and the file is mapped with  
 45692 *mmap()*.

45693 [EDEADLK] or [ENOLCK]  
 45694 The *function* argument is F\_LOCK, F\_TLOCK, or F\_ULOCK, and the request  
 45695 would cause the number of locks to exceed a system-imposed limit.

45696 [EOPNOTSUPP] or [EINVAL]  
 45697 The implementation does not support the locking of files of the type indicated  
 45698 by the *fildev* argument.

45699 **EXAMPLES**45700 **Locking a Portion of a File**

45701 In the following example, a file named `/home/cnd/mod1` is being modified. Other processes that  
45702 use locking are prevented from changing it during this process. Only the first 10 000 bytes are  
45703 locked, and the lock call fails if another process has any part of this area locked already.

```
45704 #include <fcntl.h>  
45705 #include <unistd.h>  
  
45706 int fildes;  
45707 int status;  
45708 ...  
45709 fildes = open("/home/cnd/mod1", O_RDWR);  
45710 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

45711 **APPLICATION USAGE**

45712 Record-locking should not be used in combination with buffered standard I/O streams (see  
45713 [Section 2.5](#), on page 521). Instead, non-buffered I/O should be used. Unexpected results may  
45714 occur in processes that do buffering in the user address space. The process may later read/write  
45715 data which is/was locked. Functions that operate on standard I/O streams are the most  
45716 common source of such buffering.

45717 The `alarm()` function may be used to provide a timeout facility in applications requiring it.

45718 **RATIONALE**

45719 None.

45720 **FUTURE DIRECTIONS**

45721 None.

45722 **SEE ALSO**

45723 [`alarm\(\)`](#), [`chmod\(\)`](#), [`close\(\)`](#), [`creat\(\)`](#), [`fcntl\(\)`](#), [`fopen\(\)`](#), [`mmap\(\)`](#), [`open\(\)`](#), [`read\(\)`](#), [`write\(\)`](#)

45724 XBD [`<unistd.h>`](#)

45725 **CHANGE HISTORY**

45726 First released in Issue 4, Version 2.

45727 **Issue 5**

45728 Moved from X/OPEN UNIX extension to BASE.

45729 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified  
45730 and moved from optional to mandatory status.

45731 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a  
45732 file that is associated with a buffered stream.

45733 **Issue 6**

45734 The normative text is updated to avoid use of the term “must” for application requirements.

45735 **Issue 7**

45736 Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION.

45737 **Issue 8**

45738 Austin Group Defect 768 is applied, adding OFD-owned file locks.

45739 Austin Group Defect 1672 is applied, changing the APPLICATION USAGE section.



45740 **NAME**

45741 log, logf, logl — natural logarithm function

45742 **SYNOPSIS**

```
45743 #include <math.h>
45744 double log(double x);
45745 float logf(float x);
45746 long double logl(long double x);
```

45747 **DESCRIPTION**

45748 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45749 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45750 volume of POSIX.1-2024 defers to the ISO C standard.

45751 These functions shall compute the natural logarithm of their argument  $x$ ,  $\log_e(x)$ .

45752 An application wishing to check for error situations should set *errno* to zero and call  
 45753 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 45754 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 45755 zero, an error has occurred.

45756 **RETURN VALUE**

45757 Upon successful completion, these functions shall return the natural logarithm of  $x$ .

45758 If  $x$  is  $\pm 0$ , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return `-HUGE_VAL`,  
 45759 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

45760 MX For finite values of  $x$  that are less than 0, or if  $x$  is `-Inf`, a domain error shall occur, and either a  
 45761 NaN (if supported), or an implementation-defined value shall be returned.

45762 MX If  $x$  is NaN, a NaN shall be returned.

45763 If  $x$  is 1, `+0` shall be returned.

45764 If  $x$  is `+Inf`,  $x$  shall be returned.

45765 **ERRORS**

45766 These functions shall fail if:

45767 MX Domain Error The finite value of  $x$  is negative, or  $x$  is `-Inf`.

45768 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45769 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 45770 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 45771 shall be raised.

45772 Pole Error The value of  $x$  is zero.

45773 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45774 then *errno* shall be set to [ERANGE]. If the integer expression  
 45775 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 45776 floating-point exception shall be raised.

45777 **EXAMPLES**

45778 None.

45779 **APPLICATION USAGE**

45780 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
45781 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

45782 **RATIONALE**

45783 None.

45784 **FUTURE DIRECTIONS**

45785 None.

45786 **SEE ALSO**45787 [exp\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log10\(\)](#), [log1p\(\)](#)45788 XBD Section 4.23 (on page 109), [<math.h>](#)45789 **CHANGE HISTORY**

45790 First released in Issue 1. Derived from Issue 1 of the SVID.

45791 **Issue 5**

45792 The DESCRIPTION is updated to indicate how an application should check for an error. This  
45793 text was previously published in the APPLICATION USAGE section.

45794 **Issue 6**

45795 The normative text is updated to avoid use of the term “must” for application requirements.

45796 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

45797 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
45798 revised to align with the ISO/IEC 9899:1999 standard.

45799 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
45800 marked.

45801 **NAME**45802 `log10`, `log10f`, `log10l` — base 10 logarithm function45803 **SYNOPSIS**

```
45804 #include <math.h>
45805 double log10(double x);
45806 float log10f(float x);
45807 long double log10l(long double x);
```

45808 **DESCRIPTION**

45809 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45810 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45811 volume of POSIX.1-2024 defers to the ISO C standard.

45812 These functions shall compute the base 10 logarithm of their argument  $x$ ,  $\log_{10}(x)$ .

45813 An application wishing to check for error situations should set *errno* to zero and call  
 45814 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 45815 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 45816 zero, an error has occurred.

45817 **RETURN VALUE**

45818 Upon successful completion, these functions shall return the base 10 logarithm of  $x$ .

45819 If  $x$  is  $\pm 0$ , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return `-HUGE_VAL`,  
 45820 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

45821 MX For finite values of  $x$  that are less than 0, or if  $x$  is `-Inf`, a domain error shall occur, and either a  
 45822 NaN (if supported), or an implementation-defined value shall be returned.

45823 MX If  $x$  is NaN, a NaN shall be returned.

45824 If  $x$  is 1, `+0` shall be returned.

45825 If  $x$  is `+Inf`, `+Inf` shall be returned.

45826 **ERRORS**

45827 These functions shall fail if:

45828 MX Domain Error The finite value of  $x$  is negative, or  $x$  is `-Inf`.

45829 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45830 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 45831 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 45832 shall be raised.

45833 Pole Error The value of  $x$  is zero.

45834 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45835 then *errno* shall be set to [ERANGE]. If the integer expression  
 45836 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 45837 floating-point exception shall be raised.

45838 **EXAMPLES**

45839 None.

45840 **APPLICATION USAGE**

45841 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
45842 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

45843 **RATIONALE**

45844 None.

45845 **FUTURE DIRECTIONS**

45846 None.

45847 **SEE ALSO**45848 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log\(\)](#), [pow\(\)](#)45849 XBD Section 4.23 (on page 109), [<math.h>](#)45850 **CHANGE HISTORY**

45851 First released in Issue 1. Derived from Issue 1 of the SVID.

45852 **Issue 5**

45853 The DESCRIPTION is updated to indicate how an application should check for an error. This  
45854 text was previously published in the APPLICATION USAGE section.

45855 **Issue 6**

45856 The normative text is updated to avoid use of the term “must” for application requirements.

45857 The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999  
45858 standard.

45859 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
45860 revised to align with the ISO/IEC 9899:1999 standard.

45861 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
45862 marked.

45863 **NAME**45864 `log1p`, `log1pf`, `log1pl` — compute a natural logarithm45865 **SYNOPSIS**

```
45866 #include <math.h>
45867 double log1p(double x);
45868 float log1pf(float x);
45869 long double log1pl(long double x);
```

45870 **DESCRIPTION**

45871 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45872 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45873 volume of POSIX.1-2024 defers to the ISO C standard.

45874 These functions shall compute  $\log_e(1.0 + x)$ .

45875 An application wishing to check for error situations should set *errno* to zero and call  
 45876 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 45877 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 45878 zero, an error has occurred.

45879 **RETURN VALUE**

45880 Upon successful completion, these functions shall return the natural logarithm of  $1.0 + x$ .

45881 If  $x$  is  $-1$ , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return `-HUGE_VAL`,  
 45882 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

45883 MX For finite values of  $x$  that are less than  $-1$ , or if  $x$  is  $-\text{Inf}$ , a domain error shall occur, and either a  
 45884 NaN (if supported), or an implementation-defined value shall be returned.

45885 MX If  $x$  is NaN, a NaN shall be returned.

45886 If  $x$  is  $\pm 0$ , or  $+\text{Inf}$ ,  $x$  shall be returned.

45887 MXX If  $x$  is subnormal,  $x$  should be returned.

45888 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *log1p()*, *log1pf()*, and *log1pl()*  
 45889 shall return an implementation-defined value no greater in magnitude than `DBL_MIN`,  
 45890 `FLT_MIN`, and `LDBL_MIN`, respectively.

45891 **ERRORS**

45892 These functions shall fail if:

45893 MX Domain Error The finite value of  $x$  is less than  $-1$ , or  $x$  is  $-\text{Inf}$ .

45894 If the integer expression (*math\_errhandling* & `MATH_ERRNO`) is non-zero,  
 45895 then *errno* shall be set to `[EDOM]`. If the integer expression (*math\_errhandling*  
 45896 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception  
 45897 shall be raised.

45898 Pole Error The value of  $x$  is  $-1$ .

45899 If the integer expression (*math\_errhandling* & `MATH_ERRNO`) is non-zero,  
 45900 then *errno* shall be set to `[ERANGE]`. If the integer expression  
 45901 (*math\_errhandling* & `MATH_ERREXCEPT`) is non-zero, then the divide-by-zero  
 45902 floating-point exception shall be raised.

45903 These functions may fail if:

45904 MX **Range Error** The value of  $x$  is subnormal.

45905 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45906 then *errno* shall be set to [ERANGE]. If the integer expression  
 45907 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 45908 floating-point exception shall be raised.

#### 45909 **EXAMPLES**

45910 None.

#### 45911 **APPLICATION USAGE**

45912 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 45913 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

#### 45914 **RATIONALE**

45915 None.

#### 45916 **FUTURE DIRECTIONS**

45917 None.

#### 45918 **SEE ALSO**

45919 [\*feclearexcept\(\)\*](#), [\*fetestexcept\(\)\*](#), [\*log\(\)\*](#)

45920 XBD [Section 4.23](#) (on page 109), [<math.h>](#)

#### 45921 **CHANGE HISTORY**

45922 First released in Issue 4, Version 2.

#### 45923 **Issue 5**

45924 Moved from X/OPEN UNIX extension to BASE.

#### 45925 **Issue 6**

45926 The normative text is updated to avoid use of the term “must” for application requirements.

45927 The *log1p()* function is no longer marked as an extension.

45928 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999  
 45929 standard.

45930 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 45931 revised to align with the ISO/IEC 9899:1999 standard.

45932 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 45933 marked.

#### 45934 **Issue 7**

45935 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0364 [68] is applied.

#### 45936 **Issue 8**

45937 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when  $x$  is  
 45938 subnormal to avoid the need for two shading changes.

45939 **NAME**45940 `log2, log2f, log2l` — compute base 2 logarithm functions45941 **SYNOPSIS**

```
45942 #include <math.h>
45943 double log2(double x);
45944 float log2f(float x);
45945 long double log2l(long double x);
```

45946 **DESCRIPTION**

45947 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45948 conflict between the requirements described here and the ISO C standard is unintentional. This  
 45949 volume of POSIX.1-2024 defers to the ISO C standard.

45950 These functions shall compute the base 2 logarithm of their argument  $x$ ,  $\log_2(x)$ .

45951 An application wishing to check for error situations should set *errno* to zero and call  
 45952 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 45953 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 45954 zero, an error has occurred.

45955 **RETURN VALUE**

45956 Upon successful completion, these functions shall return the base 2 logarithm of  $x$ .

45957 If  $x$  is  $\pm 0$ , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return `-HUGE_VAL`,  
 45958 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

45959 MX For finite values of  $x$  that are less than 0, or if  $x$  is `-Inf`, a domain error shall occur, and either a  
 45960 NaN (if supported), or an implementation-defined value shall be returned.

45961 MX If  $x$  is NaN, a NaN shall be returned.

45962 If  $x$  is 1, `+0` shall be returned.

45963 If  $x$  is `+Inf`,  $x$  shall be returned.

45964 **ERRORS**

45965 These functions shall fail if:

45966 MX Domain Error The finite value of  $x$  is less than zero, or  $x$  is `-Inf`.

45967 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45968 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 45969 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 45970 shall be raised.

45971 Pole Error The value of  $x$  is zero.

45972 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 45973 then *errno* shall be set to [ERANGE]. If the integer expression  
 45974 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
 45975 floating-point exception shall be raised.

45976 **EXAMPLES**

45977       None.

45978 **APPLICATION USAGE**

45979       On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
45980       MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

45981 **RATIONALE**

45982       None.

45983 **FUTURE DIRECTIONS**

45984       None.

45985 **SEE ALSO**

45986       *feclearexcept()*, *fetestexcept()*, *log()*

45987       XBD Section 4.23 (on page 109), <[math.h](#)>

45988 **CHANGE HISTORY**

45989       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.



45990 **NAME**45991 `logb`, `logbf`, `logbl` — radix-independent exponent45992 **SYNOPSIS**

```
45993 #include <math.h>
45994 double logb(double x);
45995 float logbf(float x);
45996 long double logbl(long double x);
```

45997 **DESCRIPTION**

45998 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 45999 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46000 volume of POSIX.1-2024 defers to the ISO C standard.

46001 These functions shall compute the exponent of  $x$ , which is the integral part of  $\log_r |x|$ , as a  
 46002 signed floating-point value, for non-zero  $x$ , where  $r$  is the radix of the machine's floating-point  
 46003 arithmetic, which is the value of `FLT_RADIX` defined in the `<float.h>` header.

46004 If  $x$  is subnormal it is treated as though it were normalized; thus for finite positive  $x$ :

46005 
$$1 \leq x * FLT\_RADIX^{-\text{logb}(x)} < FLT\_RADIX$$

46006 An application wishing to check for error situations should set `errno` to zero and call  
 46007 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 46008 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 46009 zero, an error has occurred.

46010 **RETURN VALUE**

46011 Upon successful completion, these functions shall return the exponent of  $x$ .

46012 MX The returned value shall be exact and shall be independent of the current rounding direction  
 46013 mode.

46014 If  $x$  is  $\pm 0$ , `logb()`, `logbf()`, and `logbl()` shall return `-HUGE_VAL`, `-HUGE_VALF`, and  
 46015 `-HUGE_VALL`, respectively.

46016 MX On systems that support the IEC 60559 Floating-Point option, a pole error shall occur;  
 46017 CX otherwise, a pole error may occur.

46018 MX If  $x$  is NaN, a NaN shall be returned.

46019 MX If  $x$  is  $\pm\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

46020 **ERRORS**

46021 These functions shall fail if:

46022 MX **Pole Error** The value of  $x$  is  $\pm 0$ .  
 46023 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 46024 then `errno` shall be set to `[ERANGE]`. If the integer expression  
 46025 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the divide-by-zero  
 46026 floating-point exception shall be raised.

46027 These functions may fail if:

46028 **Pole Error** The value of  $x$  is 0.  
 46029 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,  
 46030 then `errno` shall be set to `[ERANGE]`. If the integer expression  
 46031 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the divide-by-zero  
 46032 floating-point exception shall be raised.

46033 **EXAMPLES**

46034 None.

46035 **APPLICATION USAGE**

46036 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
46037 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46038 **RATIONALE**

46039 None.

46040 **FUTURE DIRECTIONS**

46041 None.

46042 **SEE ALSO**46043 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [ilogb\(\)](#), [scalbln\(\)](#)46044 XBD Section 4.23 (on page 109), [<float.h>](#), [<math.h>](#)46045 **CHANGE HISTORY**

46046 First released in Issue 4, Version 2.

46047 **Issue 5**

46048 Moved from X/OPEN UNIX extension to BASE.

46049 **Issue 6**46050 The *logb()* function is no longer marked as an extension.46051 The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

46052 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
46053 revised to align with the ISO/IEC 9899:1999 standard.

46054 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
46055 marked.

46056 **Issue 7**

46057 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

46058 **Issue 8**

46059 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
46060 standard.

46061 **NAME**

46062       logf, logl — natural logarithm function

46063 **SYNOPSIS**

46064       #include &lt;math.h&gt;

46065       float logf(float *x*);46066       long double logl(long double *x*);46067 **DESCRIPTION**46068       Refer to *log()*.

46069 **NAME**

46070 longjmp — non-local goto

46071 **SYNOPSIS**

46072 #include &lt;setjmp.h&gt;

46073 \_Noreturn void longjmp(jmp\_buf env, int val);

46074 **DESCRIPTION**

46075 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46076 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46077 volume of POSIX.1-2024 defers to the ISO C standard.

46078 The *longjmp()* function shall restore the environment saved by the most recent invocation of  
 46079 *setjmp()* in the same process, with the corresponding **jmp\_buf** argument. If the most recent  
 46080 invocation of *setjmp()* with the corresponding **jmp\_buf** occurred in another thread, or if there is  
 46081 no such invocation, or if the function containing the invocation of *setjmp()* has terminated  
 46082 execution in the interim, or if the invocation of *setjmp()* was within the scope of an identifier  
 46083 with variably modified type and execution has left that scope in the interim, the behavior is  
 46084 CX undefined. It is unspecified whether *longjmp()* restores the signal mask, leaves the signal mask  
 46085 unchanged, or restores it to its value at the time *setjmp()* was called.

46086 All accessible objects have values, and all other components of the abstract machine have state  
 46087 (for example, floating-point status flags and open files), as of the time *longjmp()* was called,  
 46088 except that the values of objects of automatic storage duration are unspecified if they meet all  
 46089 the following conditions:

- 46090 • They are local to the function containing the corresponding *setjmp()* invocation.
- 46091 • They do not have volatile-qualified type.
- 46092 • They are changed between the *setjmp()* invocation and *longjmp()* call.

46093 CX Although *longjmp()* is an async-signal-safe function, if it is invoked from a signal handler which  
 46094 interrupted a non-async-signal-safe function or equivalent (such as the processing equivalent to  
 46095 *exit()* performed after a return from the initial call to *main()*), the behavior of any subsequent  
 46096 call to a non-async-signal-safe function or equivalent is undefined.

46097 The effect of a call to *longjmp()* where initialization of the **jmp\_buf** structure was not performed  
 46098 in the calling thread is undefined.

46099 **RETURN VALUE**

46100 After *longjmp()* is completed, thread execution shall continue as if the corresponding invocation  
 46101 of *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause  
 46102 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

46103 **ERRORS**

46104 No errors are defined.

46105 **EXAMPLES**

46106 None.

46107 **APPLICATION USAGE**

46108 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*  
 46109 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the  
 46110 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under  
 46111 application control).

46112 It is recommended that applications do not call *longjmp()* or *siglongjmp()* from signal handlers.  
 46113 To avoid undefined behavior when calling these functions from a signal handler, the application

- 46114 needs to ensure one of the following two things:
- 46115 1. After the call to *longjmp()* or *siglongjmp()* the process only calls async-signal-safe  
46116 functions and does not return from the initial call to *main()*.
  - 46117 2. Any signal whose handler calls *longjmp()* or *siglongjmp()* is blocked during *every* call to a  
46118 non-async-signal-safe function, and no such calls are made after returning from the initial  
46119 call to *main()*.
- 46120 **RATIONALE**
- 46121 None.
- 46122 **FUTURE DIRECTIONS**
- 46123 None.
- 46124 **SEE ALSO**
- 46125 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*
- 46126 XBD <[setjmp.h](#)>
- 46127 **CHANGE HISTORY**
- 46128 First released in Issue 1. Derived from Issue 1 of the SVID.
- 46129 **Issue 5**
- 46130 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.
- 46131 **Issue 6**
- 46132 Extensions beyond the ISO C standard are marked.
- 46133 The following new requirements on POSIX implementations derive from alignment with the  
46134 Single UNIX Specification:
- 46135 • The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask  
46136 unspecified.
- 46137 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.
- 46138 **Issue 7**
- 46139 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0365 [394] is applied.
- 46140 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0202 [516] is applied.
- 46141 **Issue 8**
- 46142 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
46143 standard.

46144 **NAME**

46145 lrand48 — generate uniformly distributed pseudo-random non-negative long integers

46146 **SYNOPSIS**

```
46147 XSI #include <stdlib.h>  
46148 long lrand48(void);
```

46149 **DESCRIPTION**

46150 Refer to *drand48()*.

46151 **NAME**

46152 lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

46153 **SYNOPSIS**

```
46154 #include <math.h>
46155 long lrint(double x);
46156 long lrintf(float x);
46157 long lrintl(long double x);
```

46158 **DESCRIPTION**

46159 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46160 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46161 volume of POSIX.1-2024 defers to the ISO C standard.

46162 These functions shall round their argument to the nearest integer value, rounding according to  
 46163 the current rounding direction.

46164 An application wishing to check for error situations should set *errno* to zero and call  
 46165 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 46166 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 46167 zero, an error has occurred.

46168 **RETURN VALUE**

46169 Upon successful completion, these functions shall return the rounded integer value.

46170 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.46171 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.46172 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

46173 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be  
 46174 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall  
 46175 CX occur; otherwise, a domain error may occur.

46176 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be  
 46177 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall  
 46178 CX occur; otherwise, a domain error may occur.

46179 **ERRORS**

46180 These functions shall fail if:

46181 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 46182 integer.

46183 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 46184 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 46185 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 46186 shall be raised.

46187 These functions may fail if:

46188 **Domain Error** The correct value is not representable as an integer.

46189 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 46190 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 46191 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 46192 shall be raised.

46193 **EXAMPLES**

46194       None.

46195 **APPLICATION USAGE**46196       On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
46197       MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.46198 **RATIONALE**46199       These functions provide floating-to-integer conversions. They round according to the current  
46200       rounding direction. If the rounded value is outside the range of the return type, the numeric  
46201       result is unspecified and the invalid floating-point exception is raised. When they raise no other  
46202       floating-point exception and the result differs from the argument, they raise the inexact floating-  
46203       point exception.46204 **FUTURE DIRECTIONS**

46205       None.

46206 **SEE ALSO**46207       [feclearexcept\(\)](#), [fetestexcept\(\)](#), [llrint\(\)](#)46208       XBD Section 4.23 (on page 109), [<math.h>](#)46209 **CHANGE HISTORY**

46210       First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

46211 **Issue 7**

46212       ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.



46213 **NAME**

46214 lround, lroundf, lroundl — round to nearest integer value

46215 **SYNOPSIS**

```
46216 #include <math.h>
46217 long lround(double x);
46218 long lroundf(float x);
46219 long lroundl(long double x);
```

46220 **DESCRIPTION**

46221 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46222 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46223 volume of POSIX.1-2024 defers to the ISO C standard.

46224 These functions shall round their argument to the nearest integer value, rounding halfway cases  
 46225 away from zero, regardless of the current rounding direction.

46226 An application wishing to check for error situations should set *errno* to zero and call  
 46227 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 46228 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 46229 zero, an error has occurred.

46230 **RETURN VALUE**

46231 Upon successful completion, these functions shall return the rounded integer value.

46232 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

46233 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

46234 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

46235 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be  
 46236 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;  
 46237 CX otherwise, a domain error may occur.

46238 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be  
 46239 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;  
 46240 CX otherwise, a domain error may occur.

46241 **ERRORS**

46242 These functions shall fail if:

46243 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an  
 46244 integer.

46245 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 46246 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 46247 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 46248 shall be raised.

46249 These functions may fail if:

46250 **Domain Error** The correct value is not representable as an integer.

46251 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 46252 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 46253 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 46254 shall be raised.

46255 **EXAMPLES**

46256 None.

46257 **APPLICATION USAGE**

46258 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
46259 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46260 **RATIONALE**

46261 These functions differ from the *lrint()* functions in the default rounding direction, with the  
46262 *lround()* functions rounding halfway cases away from zero and needing not to raise the inexact  
46263 floating-point exception for non-integer arguments that round to within the range of the return  
46264 type.

46265 **FUTURE DIRECTIONS**

46266 None.

46267 **SEE ALSO**

46268 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [llround\(\)](#)

46269 XBD Section 4.23 (on page 109), [<math.h>](#)

46270 **CHANGE HISTORY**

46271 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

46272 **Issue 7**

46273 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.

46274 **NAME**

46275 lsearch, lfind — linear search and update

46276 **SYNOPSIS**

```
46277 XSI #include <search.h>
46278 void *lsearch(const void *key, void *base, size_t *nel, size_t width,
46279             int (*compar)(const void *, const void *));
46280 void *lfind(const void *key, const void *base, size_t *nel,
46281            size_t width, int (*compar)(const void *, const void *));
```

46282 **DESCRIPTION**

46283 The *lsearch()* function shall linearly search the table and return a pointer into the table for the  
 46284 matching entry. If the entry does not occur, it shall be added at the end of the table. The *key*  
 46285 argument points to the entry to be sought in the table. The *base* argument points to the first  
 46286 element in the table. The *width* argument is the size of an element in bytes. The *nel* argument  
 46287 points to an integer containing the current number of elements in the table. The integer to which  
 46288 *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to  
 46289 a comparison function which the application shall supply (for example, *strcmp()*). It is called  
 46290 with two arguments that point to the elements being compared. The application shall ensure  
 46291 that the function returns 0 if the elements are equal, and non-zero otherwise.

46292 The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not  
 46293 added to the table. Instead, a null pointer is returned.

46294 **RETURN VALUE**

46295 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it.  
 46296 Otherwise, *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly  
 46297 added element.

46298 Both functions shall return a null pointer in case of error.

46299 **ERRORS**

46300 No errors are defined.

46301 **EXAMPLES**46302 **Storing Strings in a Table**

46303 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to  
 46304 ELSIZE and stores them in a table, eliminating duplicates.

```
46305 #include <stdio.h>
46306 #include <string.h>
46307 #include <search.h>
46308 #define TABSIZE 50
46309 #define ELSIZE 120
46310 ...
46311     char line[ELSIZE], tab[TABSIZE][ELSIZE];
46312     size_t nel = 0;
46313     ...
46314     while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
46315         (void) lsearch(line, tab, &nel,
46316                       ELSIZE, (int (*)(const void *, const void *)) strcmp);
46317     ...
```

**46318 Finding a Matching Entry**

46319 The following example finds any line that reads "This is a test.".

```
46320 #include <search.h>
46321 #include <string.h>
46322 ...
46323 char line[ELSIZE], tab[TABSIZE][ELSIZE];
46324 size_t nel = 0;
46325 char *findline;
46326 void *entry;

46327 findline = "This is a test.\n";

46328 entry = lfind(findline, tab, &nel, ELSIZE, (
46329     int (*)(const void *, const void *)) strcmp);
```

**46330 APPLICATION USAGE**

46331 The comparison function need not compare every byte, so arbitrary data may be contained in  
46332 the elements in addition to the values being compared.

46333 Undefined results can occur if there is not enough room in the table to add a new item.

**46334 RATIONALE**

46335 None.

**46336 FUTURE DIRECTIONS**

46337 None.

**46338 SEE ALSO**

46339 *hcreate(), tdelete()*

46340 XBD [<search.h>](#)

**46341 CHANGE HISTORY**

46342 First released in Issue 1. Derived from Issue 1 of the SVID.

**46343 Issue 6**

46344 The normative text is updated to avoid use of the term ``must'' for application requirements.

46345 **NAME**

46346 lseek — move the read/write file offset

46347 **SYNOPSIS**

46348 #include &lt;unistd.h&gt;

46349 off\_t lseek(int *filides*, off\_t *offset*, int *whence*);46350 **DESCRIPTION**46351 The *lseek()* function shall set the file offset for the open file description associated with the file  
46352 descriptor *filides*, as follows:

- 46353 • If *whence* is SEEK\_SET, the file offset shall be set to *offset* bytes.
- 46354 • If *whence* is SEEK\_CUR, the file offset shall be set to its current location plus *offset*.
- 46355 • If *whence* is SEEK\_END, the file offset shall be set to the size of the file plus *offset*.
- 46356 • If *whence* is SEEK\_HOLE, the file offset shall be set to the smallest location of a byte within  
46357 a hole and not less than *offset*, except that if *offset* falls beyond the last byte not within a  
46358 hole, then the file offset may be set to the file size instead. It shall be an error if *offset* is  
46359 greater than or equal to the size of the file.
- 46360 • If *whence* is SEEK\_DATA, the file offset shall be set to the smallest location of a byte not  
46361 within a hole and not less than *offset*. It shall be an error if no such byte exists.

46362 The symbolic constants SEEK\_SET, SEEK\_CUR, SEEK\_END, SEEK\_HOLE, and SEEK\_DATA are  
46363 defined in <unistd.h>.46364 A hole is a contiguous region of bytes within a file, all having the value of zero. Not all bytes  
46365 with the value zero need belong to a hole; however, all seekable files shall have a virtual hole  
46366 starting at the current size of the file, whether or not the file is sparse.46367 The behavior of *lseek()* on devices which are incapable of seeking is implementation-defined.  
46368 The value of the file offset associated with such a device is undefined.46369 The *lseek()* function shall allow the file offset to be set beyond the end of the existing data in the  
46370 file. If data is later written at this point, subsequent reads of data in the gap shall return bytes  
46371 with the value 0 until data is actually written into the gap.46372 The *lseek()* function shall not, by itself, extend the size of a file.46373 SHM If *filides* refers to a shared memory object, the result of the *lseek()* function is unspecified.46374 TYM If *filides* refers to a typed memory object, the result of the *lseek()* function is unspecified.46375 **RETURN VALUE**46376 Upon successful completion, the resulting offset, as measured in bytes from the beginning of the  
46377 file, shall be returned. Otherwise, -1 shall be returned, *errno* shall be set to indicate the error, and  
46378 the file offset shall remain unchanged.46379 **ERRORS**46380 The *lseek()* function shall fail if:

- 46381 [EBADF] The *filides* argument is not an open file descriptor.
- 46382 [EINVAL] The *whence* argument is not a proper value, or the resulting file offset would  
46383 be negative for a regular file, block special file, or directory.
- 46384 [ENXIO] The *whence* argument is SEEK\_HOLE or SEEK\_DATA, and *offset* is greater  
46385 than or equal to the file size; or the *whence* argument is SEEK\_DATA and the  
46386 offset falls beyond the last byte not within a hole.

46387 [EOVERFLOW] The resulting file offset would be a value which cannot be represented  
46388 correctly in an object of type `off_t`.

46389 [ESPIPE] The *fildes* argument is associated with a pipe, FIFO, or socket.

#### 46390 EXAMPLES

46391 None.

#### 46392 APPLICATION USAGE

46393 None.

#### 46394 RATIONALE

46395 The ISO C standard includes the functions *fgetpos()* and *fsetpos()*, which work on very large files  
46396 by use of a special positioning type.

46397 Although *lseek()* may position the file offset beyond the end of the file, this function does not  
46398 itself extend the size of the file. While the only function in POSIX.1-2024 that may directly extend  
46399 the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally derived from  
46400 the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing calls on *write()*).

46401 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-  
46402 defined and device-dependent (for example, memory may have few invalid values). A negative  
46403 file offset may be valid for some devices in some implementations.

46404 The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset.  
46405 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return  
46406 to determine whether a return value of `(off_t)-1` is a negative offset or an indication of an error  
46407 condition. The standard developers did not wish to require this action on the part of a  
46408 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting  
46409 file offset would be negative for a regular file, block special file, or directory.

46410 Not all file systems support holes, and even where sparse files are supported, not all contiguous  
46411 blocks of zero bytes are required to be recognized as a hole. However, since all files are required  
46412 to have a virtual hole starting at the current file size, application writers can use SEEK\_HOLE  
46413 and SEEK\_DATA to optimize algorithms that can run faster when it is known that a block of  
46414 bytes is all zeros, because a non-sparse file will correctly report the entire file as a single non-  
46415 hole. A trivial recursive implementation for these two constants would be as follows, however,  
46416 for file systems that support sparse files, implementations are encouraged to do better.

```
46417 off_t lseek(int fildes, off_t offset, int whence)
46418 {
46419     off_t cur, end;
46420     switch (whence)
46421     {
46422     case SEEK_HOLE:
46423     case SEEK_DATA:
46424         cur = lseek(fildes, 0, SEEK_CUR);
46425         if (cur < 0)
46426             return cur;
46427         end = lseek(fildes, 0, SEEK_END);
46428         if (end < 0)
46429             return end;
46430         if (offset < end)
46431             return whence == SEEK_HOLE ?
46432                 end : lseek(fildes, offset, SEEK_SET);
46433         return lseek(fildes, cur, SEEK_SET);

```

```

46434         errno = ENXIO;
46435         return -1;
46436     default:
46437         ... /* Existing implementation */
46438     }
46439 }

```

46440 Note that although the above looks like user-space code, *lseek()* cannot be implemented with  
 46441 recursive calls in user space because this would not conform to the atomicity requirements in  
 46442 [Section 2.9.7](#) (on page 547).

#### 46443 FUTURE DIRECTIONS

46444 None.

#### 46445 SEE ALSO

46446 [open\(\)](#)

46447 XBD [<sys/types.h>](#), [<unistd.h>](#)

#### 46448 CHANGE HISTORY

46449 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 46450 Issue 5

46451 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

46452 Large File Summit extensions are added.

#### 46453 Issue 6

46454 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

46455 The following new requirements on POSIX implementations derive from alignment with the  
 46456 Single UNIX Specification:

- 46457 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 46458 required for conforming implementations of previous POSIX specifications, it was not  
 46459 required for UNIX applications.

- 46460 • The [EOVERFLOW] error condition is added. This change is to support large files.

46461 An additional [ESPIPE] error condition is added for sockets.

46462 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 46463 *lseek()* results are unspecified for typed memory objects.

#### 46464 Issue 7

46465 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0366 [421] is applied.

#### 46466 Issue 8

46467 Austin Group Defect 415 is applied, adding SEEK\_HOLE and SEEK\_DATA.

46468 **NAME**

46469        lstat — get file status

46470 **SYNOPSIS**

46471        #include <sys/stat.h>

46472        int lstat(const char \*restrict *path*, struct stat \*restrict *buf*);

46473 **DESCRIPTION**

46474        Refer to *fstatat()*.



46475 **NAME**

46476 malloc — a memory allocator

46477 **SYNOPSIS**

46478 #include &lt;stdlib.h&gt;

46479 void \*malloc(size\_t size);

46480 **DESCRIPTION**

46481 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 46482 conflict between the requirements described here and the ISO C standard is unintentional. This  
 46483 volume of POSIX.1-2024 defers to the ISO C standard.

46484 The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by  
 46485 *size* and whose value is unspecified.

46486 The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The  
 46487 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to  
 46488 a pointer to any type of object with a fundamental alignment requirement and then used to  
 46489 access such an object in the space allocated (until the space is explicitly freed or reallocated).  
 46490 Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer  
 46491 returned points to the start (lowest byte address) of the allocated space. If the space cannot be  
 46492 allocated, a null pointer shall be returned. If the size of the space requested is 0, the behavior is  
 46493 implementation-defined: either a null pointer shall be returned, or the behavior shall be as if the  
 46494 size were some non-zero value, except that the behavior is undefined if the returned pointer is  
 46495 used to access an object.

46496 For purposes of determining the existence of a data race, *malloc()* shall behave as though it  
 46497 accessed only memory locations accessible through its argument and not other static duration  
 46498 storage. The function may, however, visibly modify the storage that it allocates. Calls to  
 46499 ADV *aligned\_alloc()*, *calloc()*, *free()*, *malloc()*, *posix\_memalign()*,  
 46500 CX *reallocarray()*, and *realloc()* that allocate or deallocate a particular region of memory shall occur  
 46501 in a single total order (see Section 4.15.1, on page 100), and each such deallocation call shall  
 46502 synchronize with the next allocation (if any) in this order.

46503 **RETURN VALUE**

46504 Upon successful completion, *malloc()* shall return a pointer to the allocated space; if *size* is 0, the  
 46505 application shall ensure that the pointer is not used to access an object.

46506 CX Otherwise, it shall return a null pointer and set *errno* to indicate the error.

46507 **ERRORS**

46508 The *malloc()* function shall fail if:

46509 CX [ENOMEM] Insufficient storage space is available.

46510 The *malloc()* function may fail if:

46511 CX [EINVAL] *size* is 0 and the implementation does not support 0 sized allocations.

46512 **EXAMPLES**

46513 None.

46514 **APPLICATION USAGE**

46515 None.

46516 **RATIONALE**

46517 Some implementations set *errno* to [EAGAIN] to signal memory allocation failures that might  
46518 succeed if retried and [ENOMEM] for failures that are unlikely to ever succeed, for example due  
46519 to configured limits. Section 2.3 (on page 507) permits this behavior; when multiple error  
46520 conditions are simultaneously true there is no precedence between them.

46521 **FUTURE DIRECTIONS**

46522 None.

46523 **SEE ALSO**46524 *aligned\_alloc()*, *calloc()*, *free()*, *getrlimit()*, *posix\_memalign()*, *realloc()*

46525 XBD &lt;stdlib.h&gt;

46526 **CHANGE HISTORY**

46527 First released in Issue 1. Derived from Issue 1 of the SVID.

46528 **Issue 6**

46529 Extensions beyond the ISO C standard are marked.

46530 The following new requirements on POSIX implementations derive from alignment with the  
46531 Single UNIX Specification:

- 46532 • In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
- 46533 • The [ENOMEM] error condition is added.

46534 **Issue 7**

46535 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0203 [526] is applied.

46536 **Issue 8**

46537 Austin Group Defect 374 is applied, changing the RETURN VALUE and ERRORS sections in  
46538 relation to 0 sized allocations.

46539 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
46540 standard.

46541 Austin Group Defects 1387 and 1489 are applied, changing the RATIONALE section.

46542 **NAME**

46543           mblen — get number of bytes in a character

46544 **SYNOPSIS**

46545           #include &lt;stdlib.h&gt;

46546           int mblen(const char \*s, size\_t n);

46547 **DESCRIPTION**

46548 CX       Except for requirements relating to data races, the functionality described on this reference page  
 46549 is aligned with the ISO C standard. Any other conflict between the requirements described here  
 46550 and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C  
 46551 standard for all *mblen()* functionality except in relation to data races.

46552       If *s* is not a null pointer, *mblen()* shall determine the number of bytes constituting the character  
 46553 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:

46554       mbtowc((wchar\_t \*)0, (const char \*)0, 0);

46555       mbtowc((wchar\_t \*)0, s, n);

46556       The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 46557 *mblen()*.

46558       The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 46559 state-dependent encoding, this function shall be placed into its initial state at program startup  
 46560 and can be returned to that state by a call for which its character pointer argument, *s*, is a null  
 46561 pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the  
 46562 function to be altered as necessary. A call with *s* as a null pointer shall cause this function to  
 46563 return a non-zero value if encodings have state dependency, and 0 otherwise. If the  
 46564 implementation employs special bytes to change the shift state, these bytes shall not produce  
 46565 separate wide-character codes, but shall be grouped with an adjacent character. Changing the  
 46566 *LC\_CTYPE* category causes the shift state of this function to be unspecified.

46567 CX       The *mblen()* function need not be thread-safe; however, it shall avoid data races with all other  
 46568 functions.

46569 **RETURN VALUE**

46570       If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,  
 46571 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall  
 46572 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 46573 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a  
 46574 valid character) and may set *errno* to indicate the error. In no case shall the value returned be  
 46575 greater than *n* or the value of the {MB\_CUR\_MAX} macro.

46576 **ERRORS**46577       The *mblen()* function may fail if:

46578 CX       [EILSEQ]       An invalid character sequence is detected. In the POSIX locale an [EILSEQ]  
 46579 error cannot occur since all byte values are valid characters.

**46580 EXAMPLES**

46581 None.

**46582 APPLICATION USAGE**

46583 None.

**46584 RATIONALE**

46585 When the ISO C standard introduced threads in C11, it required *mblen()* to avoid data races  
46586 (with itself as well as with other functions), whereas POSIX.1-2008 did not require it to be  
46587 thread-safe, and in many implementations it did not avoid data races with itself and still does  
46588 not. The ISO C committee intend to change the requirements in a future version of the ISO C  
46589 standard, but since POSIX.1 currently refers to C17 it is necessary for it not to defer to the ISO C  
46590 standard regarding data races in order to continue to allow this function not to avoid data races  
46591 with itself.

**46592 FUTURE DIRECTIONS**

46593 It is expected that a change in a future version of the ISO C standard will allow a future version  
46594 of this standard to remove the data race exception from the statement that it defers to the ISO C  
46595 standard.

**46596 SEE ALSO**

46597 [\*mbtowc\(\)\*](#), [\*mbstowcs\(\)\*](#), [\*wctomb\(\)\*](#), [\*wcstombs\(\)\*](#)

46598 XBD <[stdlib.h](#)>

**46599 CHANGE HISTORY**

46600 First released in Issue 4. Aligned with the ISO C standard.

**46601 Issue 7**

46602 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0367 [109] is applied.

46603 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

**46604 Issue 8**

46605 Austin Group Defects 708 and 1302 are applied, aligning this function with the  
46606 ISO/IEC 9899:2018 standard, except in relation to data races.

46607 **NAME**

46608 mbrlen — get number of bytes in a character (restartable)

46609 **SYNOPSIS**

46610 #include &lt;wchar.h&gt;

46611 size\_t mbrlen(const char \*restrict *s*, size\_t *n*,  
46612 mbstate\_t \*restrict *ps*);46613 **DESCRIPTION**46614 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
46615 conflict between the requirements described here and the ISO C standard is unintentional. This  
46616 volume of POSIX.1-2024 defers to the ISO C standard.46617 If *s* is not a null pointer, *mbrlen()* shall determine the number of bytes constituting the character  
46618 pointed to by *s*. It shall be equivalent to:46619 mbstate\_t internal;  
46620 mbrtowc(NULL, *s*, *n*, *ps* != NULL ? *ps* : &internal);46621 If *ps* is a null pointer, the *mbrlen()* function shall use its own internal **mbstate\_t** object, which is  
46622 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
46623 pointed to by *ps* shall be used to completely describe the current conversion state of the  
46624 associated character sequence. The implementation shall behave as if no function defined in this  
46625 volume of POSIX.1-2024 calls *mbrlen()*.46626 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.46627 If called with a null *ps* argument, the *mbrlen()* function need not be thread-safe; however, such  
46628 calls shall avoid data races with calls to *mbrlen()* with a non-null argument and with calls to all  
46629 other functions.46630 The *mbrlen()* function shall not change the setting of *errno* if successful.46631 **RETURN VALUE**46632 The *mbrlen()* function shall return the first of the following that applies:46633 0 If the next *n* or fewer bytes complete the character that corresponds to the null  
46634 wide character.46635 *positive* If the next *n* or fewer bytes complete a valid character; the value returned shall  
46636 be the number of bytes that complete the character.46637 **(size\_t)−2** If the next *n* bytes contribute to an incomplete but potentially valid character,  
46638 and all *n* bytes have been processed. When *n* has at least the value of the  
46639 {MB\_CUR\_MAX} macro, this case can only occur if *s* points at a sequence of  
46640 redundant shift sequences (for implementations with state-dependent  
46641 encodings).46642 **(size\_t)−1** If an encoding error occurs, in which case the next *n* or fewer bytes do not  
46643 contribute to a complete and valid character. In this case, [EILSEQ] shall be  
46644 stored in *errno* and the conversion state is undefined.46645 **ERRORS**46646 The *mbrlen()* function shall fail if:46647 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]  
46648 error cannot occur since all byte values are valid characters.

- 46649 The *mbrlen()* function may fail if:
- 46650 [EINVAL] *ps* points to an object that contains an invalid conversion state.
- 46651 **EXAMPLES**
- 46652 None.
- 46653 **APPLICATION USAGE**
- 46654 None.
- 46655 **RATIONALE**
- 46656 None.
- 46657 **FUTURE DIRECTIONS**
- 46658 None.
- 46659 **SEE ALSO**
- 46660 *mbsinit()*, *mbrtowc()*
- 46661 XBD <[wchar.h](#)>
- 46662 **CHANGE HISTORY**
- 46663 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
- 46664 (E).
- 46665 **Issue 6**
- 46666 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.
- 46667 **Issue 7**
- 46668 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 46669 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0368 [109,105] is applied.
- 46670 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.
- 46671 **Issue 8**
- 46672 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018
- 46673 standard.

46674 **NAME**46675 `mbrtoc16`, `mbrtoc32` — convert a character to a Unicode character code (restartable)46676 **SYNOPSIS**46677 `#include <uchar.h>`46678 `size_t mbrtoc16(char16_t *restrict pc16, const char *restrict s,`  
46679 `size_t n, mbstate_t *restrict ps);`46680 `size_t mbrtoc32(char32_t *restrict pc32, const char *restrict s,`  
46681 `size_t n, mbstate_t *restrict ps);`46682 **DESCRIPTION**46683 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
46684 conflict between the requirements described here and the ISO C standard is unintentional. This  
46685 volume of POSIX.1-2024 defers to the ISO C standard.46686 If *s* is a null pointer, the `mbrtoc16()` function shall be equivalent to the call:46687 `mbrtoc16(NULL, "", 1, ps)`46688 In this case, the values of the parameters *pc16* and *n* are ignored.46689 If *s* is not a null pointer, the `mbrtoc16()` function shall inspect at most *n* bytes beginning with the  
46690 byte pointed to by *s* to determine the number of bytes needed to complete the next character  
46691 (including any shift sequences). If the function determines that the next character is complete  
46692 and valid, it shall determine the values of the corresponding wide characters and then, if *pc16* is  
46693 not a null pointer, shall store the value of the first (or only) such character in the object pointed  
46694 to by *pc16*. Subsequent calls shall store successive wide characters without consuming any  
46695 additional input until all the characters have been stored. If the corresponding wide character is  
46696 the null wide character, the resulting state described shall be the initial conversion state.46697 If *ps* is a null pointer, the `mbrtoc16()` function shall use its own internal `mbstate_t` object, which  
46698 shall be initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t`  
46699 object pointed to by *ps* shall be used to completely describe the current conversion state of the  
46700 associated character sequence.46701 The behavior of this function is affected by the `LC_CTYPE` category of the current locale.46702 The `mbrtoc16()` function shall not change the setting of `errno` if successful.46703 The `mbrtoc32()` function shall behave the same way as `mbrtoc16()` except that the first parameter  
46704 shall point to an object of type `char32_t` instead of `char16_t`. References to *pc16* in the above  
46705 description shall apply as if they were *pc32* when they are being read as describing `mbrtoc32()`.46706 If called with a null *ps* argument, the `mbrtoc16()` function need not be thread-safe; however, such  
46707 calls shall avoid data races with calls to `mbrtoc16()` with a non-null argument and with calls to  
46708 all other functions.46709 If called with a null *ps* argument, the `mbrtoc32()` function need not be thread-safe; however, such  
46710 calls shall avoid data races with calls to `mbrtoc32()` with a non-null argument and with calls to  
46711 all other functions.46712 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
46713 `mbrtoc16()` or `mbrtoc32()` with a null pointer for *ps*.46714 **RETURN VALUE**

46715 These functions shall return the first of the following that applies:

46716	0	If the next <i>n</i> or fewer bytes complete the character that corresponds to the null wide character (which is the value stored).
46717		
46718	between 1 and <i>n</i> inclusive	
46719		If the next <i>n</i> or fewer bytes complete a valid character (which is the value stored); the value returned shall be the number of bytes that complete the character.
46720		
46721		
46722	( <b>size_t</b> )−3	If the next character resulting from a previous call has been stored, in which case no bytes from the input shall be consumed by the call.
46723		
46724	( <b>size_t</b> )−2	If the next <i>n</i> bytes contribute to an incomplete but potentially valid character, and all <i>n</i> bytes have been processed (no value is stored). When <i>n</i> has at least the value of the {MB_CUR_MAX} macro, this case can only occur if <i>s</i> points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).
46725		
46726		
46727		
46728		
46729	( <b>size_t</b> )−1	If an encoding error occurs, in which case the next <i>n</i> or fewer bytes do not contribute to a complete and valid character (no value is stored). In this case, [EILSEQ] shall be stored in <i>errno</i> and the conversion state is undefined.
46730		
46731		

**46732 ERRORS**

46733 These functions shall fail if:

46734	CX	[EILSEQ]	An invalid character sequence is detected. In the POSIX locale an [EILSEQ] error cannot occur since all byte values are valid characters.
46735			

46736 These functions may fail if:

46737	CX	[EINVAL]	<i>ps</i> points to an object that contains an invalid conversion state.
-------	----	----------	--

**46738 EXAMPLES**

46739 None.

**46740 APPLICATION USAGE**

46741 None.

**46742 RATIONALE**

46743 None.

**46744 FUTURE DIRECTIONS**

46745 None.

**46746 SEE ALSO**

46747 [c16rtomb\(\)](#)

46748 XBD [<uchar.h>](#)

**46749 CHANGE HISTORY**

46750 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.



46751 **NAME**

46752 mbrtowc — convert a character to a wide-character code (restartable)

46753 **SYNOPSIS**

46754 #include &lt;wchar.h&gt;

46755 size\_t mbrtowc(wchar\_t \*restrict *pwc*, const char \*restrict *s*,  
46756 size\_t *n*, mbstate\_t \*restrict *ps*);46757 **DESCRIPTION**46758 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
46759 conflict between the requirements described here and the ISO C standard is unintentional. This  
46760 volume of POSIX.1-2024 defers to the ISO C standard.46761 If *s* is a null pointer, the *mbrtowc()* function shall be equivalent to the call:46762 mbrtowc(NULL, "", 1, *ps*)46763 In this case, the values of the arguments *pwc* and *n* are ignored.46764 If *s* is not a null pointer, the *mbrtowc()* function shall inspect at most *n* bytes beginning at the  
46765 byte pointed to by *s* to determine the number of bytes needed to complete the next character  
46766 (including any shift sequences). If the function determines that the next character is completed,  
46767 it shall determine the value of the corresponding wide character and then, if *pwc* is not a null  
46768 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide  
46769 character is the null wide character, the resulting state described shall be the initial conversion  
46770 state.46771 If *ps* is a null pointer, the *mbrtowc()* function shall use its own internal **mbstate\_t** object, which  
46772 shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t**  
46773 object pointed to by *ps* shall be used to completely describe the current conversion state of the  
46774 associated character sequence. The implementation shall behave as if no function defined in this  
46775 volume of POSIX.1-2024 calls *mbrtowc()*.46776 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale.46777 If called with a null *ps* argument, the *mbrtowc()* function need not be thread-safe; however, such  
46778 calls shall avoid data races with calls to *mbrtowc()* with a non-null argument and with calls to all  
46779 other functions.46780 The *mbrtowc()* function shall not change the setting of *errno* if successful.46781 **RETURN VALUE**46782 The *mbrtowc()* function shall return the first of the following that applies:46783 0 If the next *n* or fewer bytes complete the character that corresponds to the null  
46784 wide character (which is the value stored).46785 between 1 and *n* inclusive46786 If the next *n* or fewer bytes complete a valid character (which is the value  
46787 stored); the value returned shall be the number of bytes that complete the  
46788 character.46789 (**size\_t**)−2 If the next *n* bytes contribute to an incomplete but potentially valid character,  
46790 and all *n* bytes have been processed (no value is stored). When *n* has at least  
46791 the value of the {*MB\_CUR\_MAX*} macro, this case can only occur if *s* points at  
46792 a sequence of redundant shift sequences (for implementations with state-  
46793 dependent encodings).

46794 (size\_t)-1 If an encoding error occurs, in which case the next *n* or fewer bytes do not  
 46795 contribute to a complete and valid character (no value is stored). In this case,  
 46796 [EILSEQ] shall be stored in *errno* and the conversion state is undefined.

#### 46797 ERRORS

46798 The *mbrtowc()* function shall fail if:

46799 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]  
 46800 error cannot occur since all byte values are valid characters.

46801 The *mbrtowc()* function may fail if:

46802 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

#### 46803 EXAMPLES

46804 None.

#### 46805 APPLICATION USAGE

46806 None.

#### 46807 RATIONALE

46808 None.

#### 46809 FUTURE DIRECTIONS

46810 None.

#### 46811 SEE ALSO

46812 *mbsinit()*, *mbsrtowcs()*

46813 XBD <[wchar.h](#)>

#### 46814 CHANGE HISTORY

46815 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 46816 (E).

#### 46817 Issue 6

46818 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

46819 The following new requirements on POSIX implementations derive from alignment with the  
 46820 Single UNIX Specification:

- 46821 • The [EINVAL] error condition is added.

46822 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

#### 46823 Issue 7

46824 Austin Group Interpretation 1003.1-2001 #170 is applied.

46825 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0369 [109,105] is applied.

46826 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

#### 46827 Issue 8

46828 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
 46829 standard.

46830 **NAME**

46831 mbsinit — determine conversion object status

46832 **SYNOPSIS**

46833 #include &lt;wchar.h&gt;

46834 int mbsinit(const mbstate\_t \*ps);

46835 **DESCRIPTION**

46836 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
46837 conflict between the requirements described here and the ISO C standard is unintentional. This  
46838 volume of POSIX.1-2024 defers to the ISO C standard.

46839 If *ps* is not a null pointer, the *mbsinit()* function shall determine whether the object pointed to by  
46840 *ps* describes an initial conversion state.

46841 **RETURN VALUE**

46842 The *mbsinit()* function shall return non-zero if *ps* is a null pointer, or if the pointed-to object  
46843 describes an initial conversion state; otherwise, it shall return zero.

46844 If an **mbstate\_t** object is altered by any of the functions described as “restartable”, and is then  
46845 used with a different character sequence, or in the other conversion direction, or with a different  
46846 *LC\_CTYPE* category setting than on earlier function calls, the behavior is undefined.

46847 **ERRORS**

46848 No errors are defined.

46849 **EXAMPLES**

46850 None.

46851 **APPLICATION USAGE**

46852 The **mbstate\_t** object is used to describe the current conversion state from a particular character  
46853 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the  
46854 *LC\_CTYPE* category of the current locale.

46855 The initial conversion state corresponds, for a conversion in either direction, to the beginning of  
46856 a new character sequence in the initial shift state. A zero valued **mbstate\_t** object is at least one  
46857 way to describe an initial conversion state. A zero valued **mbstate\_t** object can be used to initiate  
46858 conversion involving any character sequence, in any *LC\_CTYPE* category setting.

46859 **RATIONALE**

46860 None.

46861 **FUTURE DIRECTIONS**

46862 None.

46863 **SEE ALSO**46864 [mbrlen\(\)](#), [mbrtowc\(\)](#), [mbsrtowcs\(\)](#), [wcrctomb\(\)](#), [wcsrtombs\(\)](#)46865 XBD [<wchar.h>](#)46866 **CHANGE HISTORY**

46867 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
46868 (E).

46869 **NAME**46870 `mbsnrto wcs, mbsrtowcs` — convert a character string to a wide-character string (restartable)46871 **SYNOPSIS**46872 `#include <wchar.h>`

```
46873 CX size_t mbsnrto wcs(wchar_t *restrict dst, const char **restrict src,
46874 size_t nmc, size_t len, mbstate_t *restrict ps);
46875 size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
46876 size_t len, mbstate_t *restrict ps);
```

46877 **DESCRIPTION**

46878 CX For `mbsrtowcs()`: The functionality described on this reference page is aligned with the ISO C  
 46879 standard. Any conflict between the requirements described here and the ISO C standard is  
 46880 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

46881 The `mbsrtowcs()` function shall convert a sequence of characters, beginning in the conversion  
 46882 state described by the object pointed to by `ps`, from the array indirectly pointed to by `src` into a  
 46883 sequence of corresponding wide characters. If `dst` is not a null pointer, the converted characters  
 46884 shall be stored into the array pointed to by `dst`. Conversion continues up to and including a  
 46885 terminating null character, which shall also be stored. Conversion shall stop early in either of the  
 46886 following cases:

- 46887 • A sequence of bytes is encountered that does not form a valid character.
- 46888 • `len` codes have been stored into the array pointed to by `dst` (and `dst` is not a null pointer).

46889 Each conversion shall take place as if by a call to the `mbrtowc()` function.

46890 If `dst` is not a null pointer, the pointer object pointed to by `src` shall be assigned either a null  
 46891 pointer (if conversion stopped due to reaching a terminating null character) or the address just  
 46892 past the last character converted (if any). If conversion stopped due to reaching a terminating  
 46893 null character, and if `dst` is not a null pointer, the resulting state described shall be the initial  
 46894 conversion state.

46895 If `ps` is a null pointer, the `mbsrtowcs()` function shall use its own internal `mbstate_t` object, which  
 46896 is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object  
 46897 pointed to by `ps` shall be used to completely describe the current conversion state of the  
 46898 associated character sequence.

46899 CX The `mbsnrto wcs()` function shall be equivalent to the `mbsrtowcs()` function, except that the  
 46900 conversion of characters indirectly pointed to by `src` is limited to at most `nmc` bytes (the size of  
 46901 the input buffer), and under conditions where `mbsrtowcs()` would assign the address just past  
 46902 the last character converted (if any) to the pointer object pointed to by `src`, `mbsnrto wcs()` shall  
 46903 instead assign the address just past the last byte processed (if any) to that pointer object. If the  
 46904 input buffer ends with an incomplete character, conversion shall stop at the end of the input  
 46905 buffer; a subsequent call to `mbsnrto wcs()` with an input buffer that starts with the remainder of  
 46906 the incomplete character shall correctly complete the conversion of that character.

46907 The behavior of these functions shall be affected by the `LC_CTYPE` category of the current locale.

46908 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 46909 these functions.

46910 CX If called with a null `ps` argument, the `mbsnrto wcs()` function need not be thread-safe; however,  
 46911 such calls shall avoid data races with calls to `mbsnrto wcs()` with a non-null argument and with  
 46912 calls to all other functions.

46913 If called with a null `ps` argument, the `mbsrtowcs()` function need not be thread-safe; however,

46914 such calls shall avoid data races with calls to *mbsrtowcs()* with a non-null argument and with  
 46915 calls to all other functions.

46916 The *mbsrtowcs()* function shall not change the setting of *errno* if successful.

#### 46917 RETURN VALUE

46918 If the input conversion encounters a sequence of bytes that do not form a valid character, an  
 46919 encoding error occurs. In this case, these functions shall store the value of the macro [EILSEQ] in  
 46920 *errno* and shall return (*size\_t*)-1; the conversion state is undefined. Otherwise, these functions  
 46921 shall return the number of characters successfully converted, not including the terminating null  
 46922 (if any).

#### 46923 ERRORS

46924 These functions shall fail if:

46925 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]  
 46926 error cannot occur since all byte values are valid characters.

46927 These functions may fail if:

46928 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

#### 46929 EXAMPLES

46930 None.

#### 46931 APPLICATION USAGE

46932 None.

#### 46933 RATIONALE

46934 None.

#### 46935 FUTURE DIRECTIONS

46936 None.

#### 46937 SEE ALSO

46938 *iconv()*, *mbrtowc()*, *mbsinit()*

46939 XBD <[wchar.h](#)>

#### 46940 CHANGE HISTORY

46941 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 46942 (E).

#### 46943 Issue 6

46944 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

46945 The [EINVAL] error condition is marked CX.

#### 46946 Issue 7

46947 Austin Group Interpretation 1003.1-2001 #170 is applied.

46948 The *mbsnrtoowcs()* function is added from The Open Group Technical Standard, 2006, Extended  
 46949 API Set Part 1.

46950 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0370 [109,105] is applied.

46951 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0205 [601], XSH/TC2-2008/0206 [663],  
 46952 and XSH/TC2-2008/0207 [601] are applied.

46953 **Issue 8**

46954 Austin Group Defect 616 is applied, requiring that when the *mbsnr towcs()* input buffer ends  
46955 with an incomplete character, conversion stops at the end of the input buffer (not at the end of  
46956 the previous character, if any).

46957 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
46958 standard.

46959 **NAME**46960 `mbstowcs` — convert a character string to a wide-character string46961 **SYNOPSIS**46962 `#include <stdlib.h>`46963 `size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,`  
46964 `size_t n);`46965 **DESCRIPTION**46966 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
46967 conflict between the requirements described here and the ISO C standard is unintentional. This  
46968 volume of POSIX.1-2024 defers to the ISO C standard.46969 The `mbstowcs()` function shall convert a sequence of characters that begins in the initial shift  
46970 state from the array pointed to by `s` into a sequence of corresponding wide-character codes and  
46971 shall store not more than `n` wide-character codes into the array pointed to by `pwcs`. No  
46972 characters that follow a null byte (which is converted into a wide-character code with value 0)  
46973 shall be examined or converted. Each character shall be converted as if by a call to `mbtowc()`,  
46974 except that the shift state of `mbtowc()` is not affected.46975 No more than `n` elements shall be modified in the array pointed to by `pwcs`. If copying takes  
46976 place between objects that overlap, the behavior is undefined.46977 **XSI** The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.  
46978 If `pwcs` is a null pointer, `mbstowcs()` shall return the length required to convert the entire array  
46979 regardless of the value of `n`, but no values are stored.46980 **RETURN VALUE**46981 **CX** If an invalid character is encountered, `mbstowcs()` shall return `(size_t)-1` and shall set `errno` to  
46982 indicate the error.46983 **XSI** Otherwise, `mbstowcs()` shall return the number of the array elements modified (or required if  
46984 `pwcs` is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if  
46985 the value returned is `n`.46986 **ERRORS**46987 The `mbstowcs()` function shall fail if:46988 **CX** `[EILSEQ]` An invalid character sequence is detected. In the POSIX locale an `[EILSEQ]`  
46989 error cannot occur since all byte values are valid characters.46990 **EXAMPLES**

46991 None.

46992 **APPLICATION USAGE**

46993 None.

46994 **RATIONALE**

46995 None.

46996 **FUTURE DIRECTIONS**

46997 None.

46998 **SEE ALSO**46999 [`mblen\(\)`](#), [`mbtowc\(\)`](#), [`wctomb\(\)`](#), [`wcstombs\(\)`](#)47000 XBD [`<stdlib.h>`](#)

47001 **CHANGE HISTORY**

47002 First released in Issue 4. Aligned with the ISO C standard.

47003 **Issue 6**

47004 The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

47005 Extensions beyond the ISO C standard are marked.

47006 **Issue 7**

47007 Austin Group Interpretation 1003.1-2001 #170 is applied.

47008 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0371 [195] is applied.

47009 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0208 [663,674] is applied.



47010 **NAME**

47011 mbtowc — convert a character to a wide-character code

47012 **SYNOPSIS**

47013 #include &lt;stdlib.h&gt;

47014 int mbtowc(wchar\_t \*restrict *pwc*, const char \*restrict *s*, size\_t *n*);47015 **DESCRIPTION**

47016 CX Except for requirements relating to data races, the functionality described on this reference page  
 47017 is aligned with the ISO C standard. Any other conflict between the requirements described here  
 47018 and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C  
 47019 standard for all *mbtowc()* functionality except in relation to data races.

47020 If *s* is not a null pointer, *mbtowc()* shall determine the number of bytes that constitute the  
 47021 character pointed to by *s*. It shall then determine the wide-character code for the value of type  
 47022 **wchar\_t** that corresponds to that character. (The value of the wide-character code corresponding  
 47023 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store  
 47024 the wide-character code in the object pointed to by *pwc*.

47025 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 47026 state-dependent encoding, this function shall be placed into its initial state at program startup  
 47027 and can be returned to that state by a call for which its character pointer argument, *s*, is a null  
 47028 pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the  
 47029 function to be altered as necessary. A call with *s* as a null pointer shall cause this function to  
 47030 return a non-zero value if encodings have state dependency, and 0 otherwise. If the  
 47031 implementation employs special bytes to change the shift state, these bytes shall not produce  
 47032 separate wide-character codes, but shall be grouped with an adjacent character. Changing the  
 47033 *LC\_CTYPE* category causes the shift state of this function to be unspecified. At most *n* bytes of  
 47034 the array pointed to by *s* shall be examined.

47035 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 47036 *mbtowc()*.

47037 CX The *mbtowc()* function need not be thread-safe; however, it shall avoid data races with all other  
 47038 functions.

47039 **RETURN VALUE**

47040 If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,  
 47041 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*  
 47042 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the  
 47043 CX converted character (if the next *n* or fewer bytes form a valid character), or return -1 and shall  
 47044 set *errno* to indicate the error (if they do not form a valid character).

47045 In no case shall the value returned be greater than *n* or the value of the {MB\_CUR\_MAX} macro.

47046 **ERRORS**

47047 The *mbtowc()* function shall fail if:

47048 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]  
 47049 error cannot occur since all byte values are valid characters.

**47050 EXAMPLES**

47051 None.

**47052 APPLICATION USAGE**

47053 None.

**47054 RATIONALE**

47055 When the ISO C standard introduced threads in C11, it required *mbtowc()* to avoid data races  
47056 (with itself as well as with other functions), whereas POSIX.1-2008 did not require it to be  
47057 thread-safe, and in many implementations it did not avoid data races with itself and still does  
47058 not. The ISO C committee intend to change the requirements in a future version of the ISO C  
47059 standard, but since POSIX.1 currently refers to C17 it is necessary for it not to defer to the ISO C  
47060 standard regarding data races in order to continue to allow this function not to avoid data races  
47061 with itself.

**47062 FUTURE DIRECTIONS**

47063 It is expected that a change in a future version of the ISO C standard will allow a future version  
47064 of this standard to remove the data race exception from the statement that it defers to the ISO C  
47065 standard.

**47066 SEE ALSO**

47067 [mblen\(\)](#), [mbstowcs\(\)](#), [wctomb\(\)](#), [wcstombs\(\)](#)

47068 XBD <[stdlib.h](#)>

**47069 CHANGE HISTORY**

47070 First released in Issue 4. Aligned with the ISO C standard.

**47071 Issue 6**

47072 The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

47073 Extensions beyond the ISO C standard are marked.

**47074 Issue 7**

47075 Austin Group Interpretation 1003.1-2001 #170 is applied.

47076 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0372 [109] and XSH/TC1-2008/0373  
47077 [195] are applied.

47078 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0209 [663,674] is applied.

**47079 Issue 8**

47080 Austin Group Defects 708 and 1302 are applied, aligning this function with the  
47081 ISO/IEC 9899:2018 standard, except in relation to data races.

47082 **NAME**

47083 memccpy — copy bytes in memory

47084 **SYNOPSIS**

```
47085 XSI #include <string.h>
47086 void *memccpy(void *restrict s1, const void *restrict s2,
47087               int c, size_t n);
```

47088 **DESCRIPTION**

47089 The *memccpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first  
47090 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,  
47091 whichever comes first. If copying takes place between objects that overlap, the behavior is  
47092 undefined.

47093 The *memccpy()* function shall not change the setting of *errno* on valid input.

47094 **RETURN VALUE**

47095 The *memccpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null  
47096 pointer if *c* was not found in the first *n* bytes of *s2*.

47097 **ERRORS**

47098 No errors are defined.

47099 **EXAMPLES**

47100 None.

47101 **APPLICATION USAGE**

47102 The *memccpy()* function does not check for the overflow of the receiving memory area.

47103 **RATIONALE**

47104 None.

47105 **FUTURE DIRECTIONS**

47106 None.

47107 **SEE ALSO**

47108 XBD [<string.h>](#)

47109 **CHANGE HISTORY**

47110 First released in Issue 1. Derived from Issue 1 of the SVID.

47111 **Issue 6**

47112 The **restrict** keyword is added to the *memccpy()* prototype for alignment with the  
47113 ISO/IEC 9899:1999 standard.

47114 **Issue 8**

47115 Austin Group Defect 448 is applied, adding a requirement that *memccpy()* does not change the  
47116 setting of *errno* on valid input.

47117 **NAME**

47118 memchr — find byte in memory

47119 **SYNOPSIS**

47120 #include &lt;string.h&gt;

47121 void \*memchr(const void \*s, int c, size\_t n);

47122 **DESCRIPTION**

47123 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
47124 conflict between the requirements described here and the ISO C standard is unintentional. This  
47125 volume of POSIX.1-2024 defers to the ISO C standard.

47126 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in  
47127 the initial *n* bytes (each interpreted as **unsigned char**) pointed to by *s*.

47128 The implementation shall behave as if it reads the bytes sequentially and stops as soon as a  
47129 matching byte is found.

47130 CX The *memchr()* function shall not change the setting of *errno* on valid input.

47131 **RETURN VALUE**

47132 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte is  
47133 not found.

47134 **ERRORS**

47135 No errors are defined.

47136 **EXAMPLES**

47137 None.

47138 **APPLICATION USAGE**

47139 None.

47140 **RATIONALE**

47141 None.

47142 **FUTURE DIRECTIONS**

47143 None.

47144 **SEE ALSO**47145 XBD <[string.h](#)>47146 **CHANGE HISTORY**

47147 First released in Issue 1. Derived from Issue 1 of the SVID.

47148 **Issue 7**

47149 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0374 [110] is applied.

47150 **Issue 8**

47151 Austin Group Defect 448 is applied, adding a requirement that *memchr()* does not change the  
47152 setting of *errno* on valid input.

47153 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
47154 standard.

47155 **NAME**

47156       memcmp — compare bytes in memory

47157 **SYNOPSIS**

47158       #include &lt;string.h&gt;

47159       int memcmp(const void \*s1, const void \*s2, size\_t n);

47160 **DESCRIPTION**47161 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
47162       conflict between the requirements described here and the ISO C standard is unintentional. This  
47163       volume of POSIX.1-2024 defers to the ISO C standard.47164       The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the  
47165       object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.47166       The sign of a non-zero return value shall be determined by the sign of the difference between the  
47167       values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects  
47168       being compared.47169 CX       The *memcmp()* function shall not change the setting of *errno* on valid input.47170 **RETURN VALUE**47171       The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object  
47172       pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.47173 **ERRORS**

47174       No errors are defined.

47175 **EXAMPLES**

47176       None.

47177 **APPLICATION USAGE**

47178       None.

47179 **RATIONALE**

47180       None.

47181 **FUTURE DIRECTIONS**

47182       None.

47183 **SEE ALSO**47184       XBD <[string.h](#)>47185 **CHANGE HISTORY**

47186       First released in Issue 1. Derived from Issue 1 of the SVID.

47187 **Issue 8**47188       Austin Group Defect 448 is applied, adding a requirement that *memcmp()* does not change the  
47189       setting of *errno* on valid input.

47190 **NAME**

47191       memcpy — copy bytes in memory

47192 **SYNOPSIS**

47193       #include &lt;string.h&gt;

47194       void \*memcpy(void \*restrict *s1*, const void \*restrict *s2*, size\_t *n*);47195 **DESCRIPTION**

47196 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
47197 conflict between the requirements described here and the ISO C standard is unintentional. This  
47198 volume of POSIX.1-2024 defers to the ISO C standard.

47199       The *memcpy()* function shall copy *n* bytes from the object pointed to by *s2* into the object pointed  
47200 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

47201 CX       The *memcpy()* function shall not change the setting of *errno* on valid input.

47202 **RETURN VALUE**47203       The *memcpy()* function shall return *s1*; no return value is reserved to indicate an error.47204 **ERRORS**

47205       No errors are defined.

47206 **EXAMPLES**

47207       None.

47208 **APPLICATION USAGE**47209       The *memcpy()* function does not check for the overflow of the receiving memory area.47210 **RATIONALE**

47211       None.

47212 **FUTURE DIRECTIONS**

47213       None.

47214 **SEE ALSO**47215       XBD <[string.h](#)>47216 **CHANGE HISTORY**

47217       First released in Issue 1. Derived from Issue 1 of the SVID.

47218 **Issue 6**47219       The *memcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.47220 **Issue 8**47221       Austin Group Defect 448 is applied, adding a requirement that *memcpy()* does not change the  
47222 setting of *errno* on valid input.

47223 **NAME**

47224 memmem — find a byte subsequence in a byte sequence

47225 **SYNOPSIS**

```
47226 CX #include <string.h>
47227 void *memmem(const void *haystack, size_t haystacklen,
47228             const void *needle, size_t needlelen);
```

47229 **DESCRIPTION**

47230 The *memmem()* function shall locate the first occurrence of byte sequence *needle* of length  
47231 *needlelen* in byte sequence *haystack* of length *haystacklen*.

47232 **RETURN VALUE**

47233 Upon successful completion, *memmem()* shall return a pointer to the the first byte of the located  
47234 byte sequence in *haystack*, or a null pointer if the byte sequence is not found.

47235 If *needlelen* is zero, the function shall return *haystack*.

47236 If *haystacklen* is less than *needlelen*, the function shall return a null pointer.

47237 **ERRORS**

47238 No errors are defined.

47239 **EXAMPLES**

47240 None.

47241 **APPLICATION USAGE**

47242 None.

47243 **RATIONALE**

47244 This function is similar to *strstr()*, except that NUL bytes may be included in either *needle* or  
47245 *haystack*.

47246 **FUTURE DIRECTIONS**

47247 None.

47248 **SEE ALSO**

47249 [\*memchr\(\)\*](#), [\*strstr\(\)\*](#)

47250 XBD [\*\*<string.h>\*\*](#)

47251 **CHANGE HISTORY**

47252 First released in Issue 8.

47253 **NAME**

47254 memmove — copy bytes in memory with overlapping areas

47255 **SYNOPSIS**

47256 #include &lt;string.h&gt;

47257 void \*memmove(void \*s1, const void \*s2, size\_t n);

47258 **DESCRIPTION**

47259 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
47260 conflict between the requirements described here and the ISO C standard is unintentional. This  
47261 volume of POSIX.1-2024 defers to the ISO C standard.

47262 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object  
47263 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first  
47264 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and  
47265 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

47266 CX The *memmove()* function shall not change the setting of *errno* on valid input.

47267 **RETURN VALUE**47268 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.47269 **ERRORS**

47270 No errors are defined.

47271 **EXAMPLES**

47272 None.

47273 **APPLICATION USAGE**

47274 None.

47275 **RATIONALE**

47276 None.

47277 **FUTURE DIRECTIONS**

47278 None.

47279 **SEE ALSO**47280 XBD <[string.h](#)>47281 **CHANGE HISTORY**

47282 First released in Issue 4. Derived from the ANSI C standard.

47283 **Issue 8**

47284 Austin Group Defect 448 is applied, adding a requirement that *memmove()* does not change the  
47285 setting of *errno* on valid input.



47286 **NAME**

47287       memset — set bytes in memory

47288 **SYNOPSIS**

47289       #include &lt;string.h&gt;

47290       void \*memset(void \*s, int c, size\_t n);

47291 **DESCRIPTION**47292 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
47293 conflict between the requirements described here and the ISO C standard is unintentional. This  
47294 volume of POSIX.1-2024 defers to the ISO C standard.47295       The `memset()` function shall copy `c` (converted to an **unsigned char**) into each of the first `n` bytes  
47296 of the object pointed to by `s`.47297 CX       The `memset()` function shall not change the setting of `errno` on valid input.47298 **RETURN VALUE**47299       The `memset()` function shall return `s`; no return value is reserved to indicate an error.47300 **ERRORS**

47301       No errors are defined.

47302 **EXAMPLES**

47303       None.

47304 **APPLICATION USAGE**

47305       None.

47306 **RATIONALE**

47307       None.

47308 **FUTURE DIRECTIONS**

47309       None.

47310 **SEE ALSO**47311       XBD <[string.h](#)>47312 **CHANGE HISTORY**

47313       First released in Issue 1. Derived from Issue 1 of the SVID.

47314 **Issue 8**47315       Austin Group Defect 448 is applied, adding a requirement that `memset()` does not change the  
47316 setting of `errno` on valid input.

47317 **NAME**

47318 mkdir, mkdirat — make a directory

47319 **SYNOPSIS**

47320 #include &lt;sys/stat.h&gt;

47321 int mkdir(const char \*path, mode\_t mode);

47322 OH #include &lt;fcntl.h&gt;

47323 int mkdirat(int fd, const char \*path, mode\_t mode);

47324 **DESCRIPTION**

47325 XSI The *mkdir()* function shall create a new directory with name *path*. The file permission bits and  
 47326 *S\_ISVTX* bit of the new directory shall be initialized from *mode*. The file permission bits of the  
 47327 *mode* argument shall be modified by the file creation mask of the process.

47328 XSI When bits in *mode* other than the file permission bits and *S\_ISVTX* are set, the meaning of these  
 47329 additional bits is implementation-defined.

47330 The directory's user ID shall be set to the process' effective user ID. The directory's group ID  
 47331 shall be set to the group ID of the parent directory or to the effective group ID of the process.  
 47332 Implementations shall provide a way to initialize the directory's group ID to the group ID of the  
 47333 parent directory. Implementations may, but need not, provide an implementation-defined way  
 47334 to initialize the directory's group ID to the effective group ID of the calling process.

47335 The newly created directory shall be an empty directory.

47336 If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

47337 Upon successful completion, *mkdir()* shall mark for update the last data access, last data  
 47338 modification, and last file status change timestamps of the directory. Also, the last data  
 47339 modification and last file status change timestamps of the directory that contains the new entry  
 47340 shall be marked for update.

47341 The *mkdirat()* function shall be equivalent to the *mkdir()* function except in the case where *path*  
 47342 specifies a relative path. In this case the newly created directory is created relative to the  
 47343 directory associated with the file descriptor *fd* instead of the current working directory. If  
 47344 the access mode of the open file description associated with the file descriptor is not O\_SEARCH,  
 47345 the function shall check whether directory searches are permitted using the current permissions  
 47346 of the directory underlying the file descriptor. If the access mode is O\_SEARCH, the function  
 47347 shall not perform the check.

47348 If *mkdirat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 47349 directory shall be used and the behavior shall be identical to a call to *mkdir()*.

47350 **RETURN VALUE**

47351 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 47352 return -1 and set *errno* to indicate the error. If -1 is returned, no directory shall be created.

47353 **ERRORS**

47354 These functions shall fail if:

47355 [EACCES] Search permission is denied on a component of the path prefix, or write  
 47356 permission is denied on the parent directory of the directory to be created.

47357 [EEXIST] The named file exists.

47358 [EILSEQ] The last pathname component of *path* is not a portable filename, and cannot be  
 47359 created in the target directory.

47360	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
47361		
47362	[EMLINK]	The link count of the parent directory would exceed {LINK_MAX}.
47363	[ENAMETOOLONG]	
47364		The length of a component of a pathname is longer than {NAME_MAX}.
47365	[ENOENT]	A component of the path prefix of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
47366		
47367	[ENOSPC]	The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.
47368		
47369	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.
47370		
47371	[EROFS]	The parent directory resides on a read-only file system.
47372		In addition, the <i>mkdirat()</i> function shall fail if:
47373	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
47374		
47375		
47376	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
47377		
47378	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
47379		
47380		These functions may fail if:
47381	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
47382		
47383	[ENAMETOOLONG]	
47384		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
47385		
47386		

## 47387 EXAMPLES

### 47388 Creating a Directory

47389 The following example shows how to create a directory named */home/cnd/mod1*, with  
 47390 read/write/search permissions for owner and group, and with read/search permissions for  
 47391 others.

```
47392 #include <sys/types.h>
47393 #include <sys/stat.h>
47394 int status;
47395 ...
47396 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

## 47397 APPLICATION USAGE

47398 None.

47399 **RATIONALE**

47400 The *mkdir()* function originated in 4.2 BSD and was added to System V in Release 3.0.

47401 4.3 BSD detects [ENAMETOOLONG].

47402 The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the  
 47403 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2  
 47404 required that implementations provide a way to have the group ID be set to the group ID of the  
 47405 containing directory, but did not prohibit implementations also supporting a way to set the  
 47406 group ID to the effective group ID of the creating process. Conforming applications should not  
 47407 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
 47408 group ID after the directory is created, or determine under what conditions the implementation  
 47409 will set the desired group ID.

47410 The purpose of the *mkdirat()* function is to create a directory in directories other than the current  
 47411 working directory without exposure to race conditions. Any part of the path of a file could be  
 47412 changed in parallel to the call to *mkdir()*, resulting in unspecified behavior. By opening a file  
 47413 descriptor for the target directory and using the *mkdirat()* function it can be guaranteed that the  
 47414 newly created directory is located relative to the desired directory.

47415 Implementations are encouraged to have *mkdir()* and *mkdirat()* report an [EILSEQ] error if the  
 47416 last component of *path* contains any bytes that have the encoded value of a <newline> character.

47417 **FUTURE DIRECTIONS**

47418 None.

47419 **SEE ALSO**

47420 *chmod()*, *mkdtemp()*, *mknod()*, *umask()*

47421 XBD <fcntl.h>, <sys/stat.h>, <sys/types.h>

47422 **CHANGE HISTORY**

47423 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

47424 **Issue 6**

47425 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

47426 The following new requirements on POSIX implementations derive from alignment with the  
 47427 Single UNIX Specification:

47428 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 47429 required for conforming implementations of previous POSIX specifications, it was not  
 47430 required for UNIX applications.

47431 • The [ELOOP] mandatory error condition is added.

47432 • A second [ENAMETOOLONG] is added as an optional error condition.

47433 The following changes were made to align with the IEEE P1003.1a draft standard:

47434 • The [ELOOP] optional error condition is added.

47435 **Issue 7**

47436 Austin Group Interpretation 1003.1-2001 #143 is applied.

47437 The *mkdirat()* function is added from The Open Group Technical Standard, 2006, Extended API  
 47438 Set Part 2.

47439 Changes are made related to support for finegrained timestamps.

47440 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0375 [461], XSH/TC1-2008/0376 [324],

47441 XSH/TC1-2008/0377 [277], XSH/TC1-2008/0378 [278], and XSH/TC1-2008/0379 [278] are  
47442 applied.

47443 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0210 [873], XSH/TC2-2008/0211 [591],  
47444 XSH/TC2-2008/0212 [817], XSH/TC2-2008/0213 [817], and XSH/TC2-2008/0214 [591] are  
47445 applied.

47446 **Issue 8**

47447 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
47448 filenames containing any bytes that have the encoded value of a <newline> character.

47449 Austin Group Defects 293 and 1734 are applied, adding the [EILSEQ] error.

47450 Austin Group Defect 1522 is applied, adding requirements relating to the S\_ISVTX bit.

47451 Austin Group Defect 1729 is applied, changing the description of the [ENOENT] error.

47452 **NAME**

47453 mkdtemp, mkostemp, mkstemp — create a unique directory or file

47454 **SYNOPSIS**

```
47455 CX #include <stdlib.h>
47456 char *mkdtemp(char *template);
47457 int mkostemp(char *template, int flag);
47458 int mkstemp(char *template);
```

47459 **DESCRIPTION**

47460 The *mkdtemp()* function shall create a directory with a unique name derived from *template*. The  
 47461 application shall ensure that the string provided in *template* is a pathname ending with at least  
 47462 six trailing 'X' characters. The *mkdtemp()* function shall modify the contents of *template* by  
 47463 replacing six or more 'X' characters at the end of the pathname with the same number of  
 47464 characters from the portable filename character set. The characters shall be chosen such that the  
 47465 resulting pathname does not duplicate the name of an existing file at the time of the call to  
 47466 *mkdtemp()*. The *mkdtemp()* function shall use the resulting pathname to create the new directory  
 47467 as if by a call to:

47468 `mkdir(pathname, S_IRWXU)`

47469 The *mkstemp()* function shall create a regular file with a unique name derived from *template* and  
 47470 return a file descriptor for the file open for reading and writing. The application shall ensure that  
 47471 the string provided in *template* is a pathname ending with at least six trailing 'X' characters. The  
 47472 *mkstemp()* function shall modify the contents of *template* by replacing six or more 'X' characters  
 47473 at the end of the pathname with the same number of characters from the portable filename  
 47474 character set. The characters shall be chosen such that the resulting pathname does not duplicate  
 47475 the name of an existing file at the time of the call to *mkstemp()*. The *mkstemp()* function shall use  
 47476 the resulting pathname to create the file, and obtain a file descriptor for it, as if by a call to:

47477 `open(pathname, O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR)`

47478 By behaving as if the `O_EXCL` flag for *open()* is set, the function prevents any possible race  
 47479 condition between testing whether the file exists and opening it for use.

47480 The *mkostemp()* function shall be equivalent to the *mkstemp()* function, except that the *flag*  
 47481 argument can contain additional flags to be used as if by *open()*. Behavior is unspecified if the  
 47482 *flag* argument contains more than the following flags:

47483 `O_APPEND` Set append mode.47484 `O_CLOEXEC` Set the `FD_CLOEXEC` file descriptor flag.47485 `O_CLOFORK` Set the `FD_CLOFORK` file descriptor flag.47486 SIO `O_DSYNC` Write according to the synchronized I/O data integrity completion.47487 SIO `O_RSYNC` Synchronized read I/O operations.47488 XSI|SIO `O_SYNC` Write according to synchronized I/O file integrity completion.47489 **RETURN VALUE**

47490 Upon successful completion, the *mkdtemp()* function shall return the value of *template*.  
 47491 Otherwise, it shall return a null pointer and shall set *errno* to indicate the error.

47492 Upon successful completion, the *mkstemp()* function shall return an open file descriptor.  
 47493 Otherwise, it shall return `-1` and shall set *errno* to indicate the error.

47494 **ERRORS**

47495 These functions shall fail if:

47496 [EINVAL] The string pointed to by *template* does not end in "XXXXXX".47497 The *mkostemp()* function may fail if:47498 [EINVAL] The value of the *flag* argument is invalid.47499 Additional error conditions for the *mkdtemp()* function are defined in *mkdir()*. Additional error  
47500 conditions for the *mkstemp()* and *mkostemp()* functions are defined in *open()*.47501 **EXAMPLES**47502 **Generating a Pathname**47503 The following example creates a file with a 10-character name beginning with the characters  
47504 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file  
47505 descriptor that identifies the file.

```

47506 #include <stdlib.h>
47507 ...
47508 char template[] = "/tmp/fileXXXXXX";
47509 int fd;
47510
47510 fd = mkstemp(template);

```

47511 **APPLICATION USAGE**

47512 It is possible to run out of letters.

47513 Portable applications should pass exactly six trailing 'X's in the template and no more;  
47514 implementations may treat any additional trailing 'X's as either a fixed or replaceable part of  
47515 the template. To be sure of only passing six, a fixed string of at least one non-'X' character  
47516 should precede the six 'X's.47517 Since 'X' is in the portable filename character set, some of the replacement characters can be  
47518 'X's, leaving part (or even all) of the template effectively unchanged.47519 **RATIONALE**47520 The O\_CLOEXEC and O\_CLOFORK flags of *mkostemp()* are necessary to avoid a data race in  
47521 multi-threaded applications. Without O\_CLOFORK, a file descriptor is leaked into a child  
47522 process created by one thread in the window between another thread creating a temporary file  
47523 descriptor with *mkstemp()* and then using *fcntl()* to set the FD\_CLOFORK flag. Without  
47524 O\_CLOEXEC, a temporary file descriptor intentionally inherited by child processes is similarly  
47525 leaked into an executed program if FD\_CLOEXEC is not set atomically.47526 Implementations are encouraged to have *mkdtemp()*, *mkostemp()*, and *mkstemp()* report an  
47527 [EILSEQ] error if the last component of the pathname in *template* contains any bytes that have  
47528 the encoded value of a <newline> character.47529 **FUTURE DIRECTIONS**

47530 None.

47531 **SEE ALSO**47532 *getpid()*, *mkdir()*, *open()*, *tmpfile()*, *tmpnam()*

47533 XBD &lt;stdlib.h&gt;

47534 **CHANGE HISTORY**

47535 First released in Issue 4, Version 2.

47536 **Issue 5**

47537 Moved from X/OPEN UNIX extension to BASE.

47538 **Issue 7**

47539 Austin Group Interpretation 1003.1-2001 #143 is applied.

47540 SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

47541 The *mkstemp()* function is moved from the XSI option to the Base.47542 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

47544 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0380 [291], XSH/TC1-2008/0381 [324], and XSH/TC1-2008/0382 [291] are applied.

47546 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0215 [567,669] is applied.

47547 **Issue 8**

47548 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a &lt;newline&gt; character.

47550 Austin Group Defects 411, 1318, and 1350 are applied, adding *mkostemp()*.47551 Austin Group Defect 652 is applied, adding the [EINVAL] error for *mkstemp()*.47552 Austin Group Defect 1734 is applied, replacing the error conditions specified only for *mkdtemp()* with a reference to *mkdir()*.

47553



47554 **NAME**

47555 mkfifo, mkfifoat — make a FIFO special file

47556 **SYNOPSIS**

47557 #include &lt;sys/stat.h&gt;

47558 int mkfifo(const char \*path, mode\_t mode);

47559 OH #include &lt;fcntl.h&gt;

47560 int mkfifoat(int fd, const char \*path, mode\_t mode);

47561 **DESCRIPTION**

47562 The *mkfifo()* function shall create a new FIFO special file named by the pathname pointed to by  
 47563 *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission  
 47564 bits of the *mode* argument shall be modified by the process' file creation mask.

47565 When bits in *mode* other than the file permission bits are set, the effect is implementation-  
 47566 defined.

47567 If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].

47568 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set  
 47569 to the group ID of the parent directory or to the effective group ID of the process.  
 47570 Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the  
 47571 parent directory. Implementations may, but need not, provide an implementation-defined way  
 47572 to initialize the FIFO's group ID to the effective group ID of the calling process.

47573 Upon successful completion, *mkfifo()* shall mark for update the last data access, last data  
 47574 modification, and last file status change timestamps of the file. Also, the last data modification  
 47575 and last file status change timestamps of the directory that contains the new entry shall be  
 47576 marked for update.

47577 The *mkfifoat()* function shall be equivalent to the *mkfifo()* function except in the case where *path*  
 47578 specifies a relative path. In this case the newly created FIFO is created relative to the directory  
 47579 associated with the file descriptor *fd* instead of the current working directory. If the access mode  
 47580 of the open file description associated with the file descriptor is not O\_SEARCH, the function  
 47581 shall check whether directory searches are permitted using the current permissions of the  
 47582 directory underlying the file descriptor. If the access mode is O\_SEARCH, the function shall not  
 47583 perform the check.

47584 If *mkfifoat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 47585 directory shall be used and the behavior shall be identical to a call to *mkfifo()*.

47586 **RETURN VALUE**

47587 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 47588 return -1 and set *errno* to indicate the error. If -1 is returned, no FIFO shall be created.

47589 **ERRORS**

47590 These functions shall fail if:

47591 [EACCES] A component of the path prefix denies search permission, or write permission  
 47592 is denied on the parent directory of the FIFO to be created.

47593 [EEXIST] The named file already exists.

47594 [EILSEQ] The last pathname component of *path* is not a portable filename, and cannot be  
 47595 created in the target directory.

47596	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
47597		
47598	[ENAMETOOLONG]	
47599		The length of a component of a pathname is longer than {NAME_MAX}.
47600	[ENOENT]	A component of the path prefix of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
47601		
47602	[ENOENT] or [ENOTDIR]	
47603		The <i>path</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters. If <i>path</i> without the trailing <code>&lt;slash&gt;</code> characters would name an existing file, an [ENOENT] error shall not occur.
47604		
47605		
47606	[ENOSPC]	The directory that would contain the new file cannot be extended or the file system is out of file-allocation resources.
47607		
47608	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.
47609		
47610	[EROFS]	The named file resides on a read-only file system.
47611		The <i>mkfifoat()</i> function shall fail if:
47612	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
47613		
47614		
47615	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
47616		
47617	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
47618		
47619		These functions may fail if:
47620	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
47621		
47622	[ENAMETOOLONG]	
47623		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
47624		
47625		

## 47626 EXAMPLES

### 47627 Creating a FIFO File

47628 The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with  
47629 read/write permissions for owner, and with read permissions for group and others.

```
47630 #include <sys/types.h>
47631 #include <sys/stat.h>
47632
47633 int status;
47634 ...
47635 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
47636             S_IRGRP | S_IROTH);
```

47636 **APPLICATION USAGE**

47637 None.

47638 **RATIONALE**

47639 The syntax of this function is intended to maintain compatibility with historical  
 47640 implementations of *mknod()*. The latter function was included in the 1984 /usr/group standard  
 47641 but only for use in creating FIFO special files. The *mknod()* function was originally excluded  
 47642 from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and  
 47643 *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX  
 47644 Specification.

47645 The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the  
 47646 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2  
 47647 required that implementations provide a way to have the group ID be set to the group ID of the  
 47648 containing directory, but did not prohibit implementations also supporting a way to set the  
 47649 group ID to the effective group ID of the creating process. Conforming applications should not  
 47650 assume which group ID will be used. If it matters, an application can use *chown()* to set the  
 47651 group ID after the FIFO is created, or determine under what conditions the implementation will  
 47652 set the desired group ID.

47653 The purpose of the *mkfifoat()* function is to create a FIFO special file in directories other than the  
 47654 current working directory without exposure to race conditions. Any part of the path of a file  
 47655 could be changed in parallel to a call to *mkfifo()*, resulting in unspecified behavior. By opening a  
 47656 file descriptor for the target directory and using the *mkfifoat()* function it can be guaranteed that  
 47657 the newly created FIFO is located relative to the desired directory.

47658 Implementations are encouraged to have *mkfifo()* and *mkfifoat()* report an [EILSEQ] error if the  
 47659 last component of *path* contains any bytes that have the encoded value of a <newline> character.

47660 **FUTURE DIRECTIONS**

47661 None.

47662 **SEE ALSO**47663 *chmod()*, *mknod()*, *umask()*

47664 XBD &lt;fcntl.h&gt;, &lt;sys/stat.h&gt;, &lt;sys/types.h&gt;

47665 **CHANGE HISTORY**

47666 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

47667 **Issue 6**

47668 In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

47669 The following new requirements on POSIX implementations derive from alignment with the  
 47670 Single UNIX Specification:

47671 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
 47672 required for conforming implementations of previous POSIX specifications, it was not  
 47673 required for UNIX applications.

47674 • The [ELOOP] mandatory error condition is added.

47675 • A second [ENAMETOOLONG] is added as an optional error condition.

47676 The following changes were made to align with the IEEE P1003.1a draft standard:

47677 • The [ELOOP] optional error condition is added.

47678 **Issue 7**

47679 Austin Group Interpretation 1003.1-2001 #143 is applied.

47680 The *mkfifoat()* function is added from The Open Group Technical Standard, 2006, Extended API  
47681 Set Part 2.

47682 Changes are made related to support for finegrained timestamps.

47683 Changes are made to allow a directory to be opened for searching.

47684 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0383 [461], XSH/TC1-2008/0384  
47685 [146,435], XSH/TC1-2008/0385 [324], XSH/TC1-2008/0386 [278], and XSH/TC1-2008/0387  
47686 [278] are applied.

47687 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0216 [873], XSH/TC2-2008/0217 [591],  
47688 XSH/TC2-2008/0218 [817], XSH/TC2-2008/0219 [822], XSH/TC2-2008/0220 [817], and  
47689 XSH/TC2-2008/0221 [591] are applied.

47690 **Issue 8**

47691 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
47692 filenames containing any bytes that have the encoded value of a <newline> character.

47693 Austin Group Defect 293 is applied, adding the [EILSEQ] error.

47694 **NAME**

47695 mknod, mknodat — make directory, special file, or regular file

47696 **SYNOPSIS**

```
47697 XSI #include <sys/stat.h>
47698 int mknod(const char *path, mode_t mode, dev_t dev);
```

```
47699 OH XSI #include <fcntl.h>
```

```
47700 XSI int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

47701 **DESCRIPTION**

47702 The *mknod()* function shall create a new file named by the pathname to which the argument *path*  
 47703 points.

47704 The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the  
 47705 following symbolic constants:

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

47712 The only portable use of *mknod()* is to create a FIFO-special file. If *mode* is not S\_IFIFO or *dev* is  
 47713 not 0, the behavior of *mknod()* is unspecified.

47714 The permissions for the new file are OR'ed into the *mode* argument, and may be selected from  
 47715 any combination of the following symbolic constants:

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

47732 The user ID of the file shall be initialized to the effective user ID of the process. The group ID of  
 47733 the file shall be initialized to either the effective group ID of the process or the group ID of the  
 47734 parent directory. Implementations shall provide a way to initialize the file's group ID to the  
 47735 group ID of the parent directory. Implementations may, but need not, provide an  
 47736 implementation-defined way to initialize the file's group ID to the effective group ID of the

47737 calling process. The owner, group, and other permission bits of *mode* shall be modified by the file  
 47738 mode creation mask of the process. The *mknod()* function shall clear each bit whose  
 47739 corresponding bit in the file mode creation mask of the process is set.

47740 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].

47741 Upon successful completion, *mknod()* shall mark for update the last data access, last data  
 47742 modification, and last file status change timestamps of the file. Also, the last data modification  
 47743 and last file status change timestamps of the directory that contains the new entry shall be  
 47744 marked for update.

47745 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO-  
 47746 special.

47747 The *mknodat()* function shall be equivalent to the *mknod()* function except in the case where *path*  
 47748 specifies a relative path. In this case the newly created directory, special file, or regular file is  
 47749 located relative to the directory associated with the file descriptor *fd* instead of the current  
 47750 working directory. If the access mode of the open file description associated with the file  
 47751 descriptor is not O\_SEARCH, the function shall check whether directory searches are permitted  
 47752 using the current permissions of the directory underlying the file descriptor. If the access mode  
 47753 is O\_SEARCH, the function shall not perform the check.

47754 If *mknodat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 47755 directory shall be used and the behavior shall be identical to a call to *mknod()*.

#### 47756 RETURN VALUE

47757 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 47758 return -1 and set *errno* to indicate the error. If -1 is returned, the new file shall not be created.

#### 47759 ERRORS

47760 These functions shall fail if:

47761 [EACCES] A component of the path prefix denies search permission, or write permission  
 47762 is denied on the parent directory.

47763 [EEXIST] The named file exists.

47764 [EILSEQ] The last pathname component of *path* is not a portable filename, and cannot be  
 47765 created in the target directory.

47766 [EINVAL] An invalid argument exists.

47767 [EIO] An I/O error occurred while accessing the file system.

47768 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
 47769 argument.

47770 [ENAMETOOLONG]

47771 The length of a component of a pathname is longer than {NAME\_MAX}.

47772 [ENOENT] A component of the path prefix of *path* does not name an existing file or *path* is  
 47773 an empty string.

47774 [ENOENT] or [ENOTDIR]

47775 The *path* argument contains at least one non-*<slash>* character and ends with  
 47776 one or more trailing *<slash>* characters. If *path* without the trailing *<slash>*  
 47777 characters would name an existing file, an [ENOENT] error shall not occur.

47778	[ENOSPC]	The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.
47779		
47780	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.
47781		
47782	[EPERM]	The invoking process does not have appropriate privileges and the file type is not FIFO-special.
47783		
47784	[EROFS]	The directory in which the file is to be created is located on a read-only file system.
47785		
47786		The <i>mknodat()</i> function shall fail if:
47787	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
47788		
47789		
47790	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
47791		
47792	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
47793		
47794		These functions may fail if:
47795	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
47796		
47797	[ENAMETOOLONG]	The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
47798		
47799		
47800		

## EXAMPLES

### Creating a FIFO Special File

The following example shows how to create a FIFO special file named */home/cnd/mod\_done*, with read/write permissions for owner, and with read permissions for group and others.

```
#include <sys/types.h>
#include <sys/stat.h>

dev_t dev;
int status;
...
status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
              S_IRUSR | S_IRGRP | S_IROTH, dev);
```

## APPLICATION USAGE

The *mkfifo()* function is preferred over this function for making FIFO special files.

## RATIONALE

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown()* to set the

47821 group ID after the file is created, or determine under what conditions the implementation will  
47822 set the desired group ID.

47823 The purpose of the *mknodat()* function is to create directories, special files, or regular files in  
47824 directories other than the current working directory without exposure to race conditions. Any  
47825 part of the path of a file could be changed in parallel to a call to *mknod()*, resulting in unspecified  
47826 behavior. By opening a file descriptor for the target directory and using the *mknodat()* function it  
47827 can be guaranteed that the newly created directory, special file, or regular file is located relative  
47828 to the desired directory.

47829 Implementations are encouraged to have *mknod()* and *mknodat()* report an [EILSEQ] error if the  
47830 last component of *path* contains any bytes that have the encoded value of a <newline> character.

#### 47831 **FUTURE DIRECTIONS**

47832 None.

#### 47833 **SEE ALSO**

47834 *chmod()*, *creat()*, *exec*, *fstatat()*, *mkdir()*, *mkfifo()*, *open()*, *umask()*

47835 XBD <fcntl.h>, <sys/stat.h>

#### 47836 **CHANGE HISTORY**

47837 First released in Issue 4, Version 2.

#### 47838 **Issue 5**

47839 Moved from X/OPEN UNIX extension to BASE.

#### 47840 **Issue 6**

47841 The normative text is updated to avoid use of the term “must” for application requirements.

47842 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
47843 [ELOOP] error condition is added.

#### 47844 **Issue 7**

47845 Austin Group Interpretation 1003.1-2001 #143 is applied.

47846 The *mknodat()* function is added from The Open Group Technical Standard, 2006, Extended API  
47847 Set Part 2.

47848 Changes are made related to support for finegrained timestamps.

47849 Changes are made to allow a directory to be opened for searching.

47850 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0388 [324], XSH/TC1-2008/0389 [461],  
47851 XSH/TC1-2008/0390 [146,435], XSH/TC1-2008/0391 [278], and XSH/TC1-2008/0392 [278] are  
47852 applied.

47853 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0222 [591], XSH/TC2-2008/0223 [817],  
47854 XSH/TC2-2008/0224 [822], XSH/TC2-2008/0225 [817], and XSH/TC2-2008/0226 [591] are  
47855 applied.

#### 47856 **Issue 8**

47857 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
47858 filenames containing any bytes that have the encoded value of a <newline> character.

47859 Austin Group Defect 293 is applied, adding the [EILSEQ] error.



47860 **NAME**

47861 mkostemp — create a unique file

47862 **SYNOPSIS**

```
47863 CX #include <stdlib.h>  
47864 int mkostemp(char *template, int flag);
```

47865 **DESCRIPTION**47866 Refer to *mkdtemp()*.

47867 **NAME**

47868 mkstemp — create a unique file

47869 **SYNOPSIS**

```
47870 CX #include <stdlib.h>  
47871 int mkstemp(char *template);
```

47872 **DESCRIPTION**

47873 Refer to *mkdtemp()*.

47874 **NAME**

47875 mktime — convert broken-down time into time since the Epoch

47876 **SYNOPSIS**

47877 #include &lt;time.h&gt;

47878 time\_t mktime(struct tm \*timeptr);

47879 **DESCRIPTION**

47880 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 47881 conflict between the requirements described here and the ISO C standard is unintentional. This  
 47882 volume of POSIX.1-2024 defers to the ISO C standard.

47883 The *mktime()* function shall convert the broken-down time, expressed as local time, in some  
 47884 members of the structure pointed to by *timeptr*, into a time since the Epoch value with the same  
 47885 encoding as that of the values returned by *time()*. The *mktime()* function shall make use of only  
 47886 the *tm\_year*, *tm\_mon*, *tm\_mday*, *tm\_hour*, *tm\_min*, *tm\_sec*, and *tm\_isdst* members of the structure  
 47887 pointed to by *timeptr*; the values of these members shall not be restricted to the ranges described  
 47888 in <time.h>.

47889 CX Local timezone information shall be set as though *mktime()* called *tzset()*.

47890 The *mktime()* function shall calculate the time in seconds since the Epoch to be returned as if by  
 47891 manipulating the members of the **tm** structure according to the following steps.

- 47892 1. The *tm\_sec* member may, but should not, be brought into the range 0 to 60, inclusive. For  
 47893 each 60 seconds added to or subtracted from *tm\_sec*, a decrement or increment,  
 47894 respectively, of 1 minute shall be saved for later application.
- 47895 2. The *tm\_min* member shall be brought into the range 0 to 59, inclusive, and any saved  
 47896 decrement or increment of minutes shall then be applied, repeating the range adjustment  
 47897 afterwards if necessary. For each 60 minutes added to or subtracted from *tm\_min*, a  
 47898 decrement or increment, respectively, of 1 hour shall be saved for later application.
- 47899 3. The *tm\_hour* member shall be brought into the range 0 to 23, inclusive, and any saved  
 47900 decrement or increment of hours shall then be applied, repeating the range adjustment  
 47901 afterwards if necessary. For each 24 hours added to or subtracted from *tm\_hour*, a  
 47902 decrement or increment, respectively, of 1 day shall be saved for later application.
- 47903 4. The *tm\_mon* member shall be brought into the range 0 to 11, inclusive. For each 12 months  
 47904 added to or subtracted from *tm\_mon*, a decrement or increment, respectively, of 1 year  
 47905 shall be saved for later use.
- 47906 5. The *tm\_mday* member shall be brought into the range 1 to 31, inclusive, and any saved  
 47907 decrement or increment of days shall then be applied, repeating the range adjustment  
 47908 afterwards if necessary. Adjustments downwards shall be applied by subtracting the  
 47909 number of days (according to the Gregorian calendar) in month *tm\_mon+1* of the year  
 47910 obtained by adding/subtracting any saved increment/decrement of years to the value  
 47911 *tm\_year+1900*, and then incrementing *tm\_mon* by 1, repeated as necessary. Adjustments  
 47912 upwards shall be applied by adding the number of days in the month before month  
 47913 *tm\_mon+1* of the year obtained by adding/subtracting any saved increment/decrement of  
 47914 years to the value *tm\_year+1900*, and then decrementing *tm\_mon* by 1, repeated as  
 47915 necessary. During these adjustments, the *tm\_mon* value shall be kept within the range 0 to  
 47916 11, inclusive, by applying step 4 as necessary.
- 47917 6. If the *tm\_mday* member is greater than the number of days in month *tm\_mon+1* of the year  
 47918 obtained by adding/subtracting any saved increment/decrement of years to the value  
 47919 *tm\_year+1900*, that number of days shall be subtracted from *tm\_mday*, and *tm\_mon* shall

- 47920 be incremented by 1. If this results in *tm\_mon* having the value 12, step 4 shall be applied.
- 47921 7. The number of seconds since the Epoch in Coordinated Universal Time shall be  
 47922 calculated from the range-corrected values of the relevant **tm** structure members (or the  
 47923 original value where a member was not range corrected) as specified in the expression  
 47924 given in the definition of seconds since the Epoch (see XBD Section 4.19, on page 107),  
 47925 where the names other than *tm\_year* and *tm\_yday* in the structure and in the expression  
 47926 correspond, the *tm\_year* value used in the expression is the *tm\_year* in the structure  
 47927 plus/minus any saved increment/decrement of years, and the *tm\_yday* value used in the  
 47928 expression is the day of the year from 0 to 365 inclusive, calculated from the *tm\_mon* and  
 47929 *tm\_mday* members of the **tm** structure, for that year.
- 47930 8. The time since the Epoch shall be corrected for the offset of the local timezone's standard  
 47931 time from Coordinated Universal Time.
- 47932 9. The time since the Epoch shall be further corrected (if applicable—see below) for Daylight  
 47933 Saving Time.

47934 If the timezone is one that includes Daylight Saving Time (DST) adjustments, the value of  
 47935 *tm\_isdst* in the **tm** structure controls whether or not *mktime()* adjusts the calculated seconds since  
 47936 the Epoch value by the DST offset (after it has made the timezone adjustment), as follows:

- 47937 • If *tm\_isdst* is zero, *mktime()* shall not further adjust the seconds since the Epoch by the DST  
 47938 offset.
- 47939 • If *tm\_isdst* is positive, *mktime()* shall further adjust the seconds since the Epoch by the DST  
 47940 offset.
- 47941 • If *tm\_isdst* is negative, *mktime()* shall attempt to determine whether DST is in effect for the  
 47942 specified time; if it determines that DST is in effect it shall produce the same result as an  
 47943 equivalent call with a positive *tm\_isdst* value, otherwise it shall produce the same result as  
 47944 an equivalent call with a *tm\_isdst* value of zero. If the broken-down time specifies a time  
 47945 that is either skipped over or repeated when a transition to or from DST occurs, it is  
 47946 unspecified whether *mktime()* produces the same result as an equivalent call with a  
 47947 positive *tm\_isdst* value or as an equivalent call with a *tm\_isdst* value of zero.

47948 If the *TZ* environment variable specifies a geographical timezone for which the  
 47949 implementation's timezone database includes historical or future changes to the offset from  
 47950 Coordinated Universal Time of the timezone's standard time, and the broken-down time  
 47951 corresponds to a time that was (or will be) skipped over or repeated due to the occurrence of  
 47952 such a change, *mktime()* shall calculate the time since the Epoch value using either the offset in  
 47953 effect before the change or the offset in effect after the change.

47954 Upon successful completion, the members of the structure shall be set to the values that would  
 47955 be returned by a call to *localtime()* with the calculated time since the Epoch as its argument.

#### 47956 RETURN VALUE

47957 The *mktime()* function shall return the calculated time since the Epoch encoded as a value of  
 47958 type **time\_t**. If the time since the Epoch cannot be represented as a **time\_t** or the value to be  
 47959 returned in the *tm\_year* member of the structure pointed to by *timeptr* cannot be represented as  
 47960 an **int**, the function shall return the value (**time\_t**)−1 and set *errno* to [Eoverflow], and shall  
 47961 not change the value of the *tm\_wday* component of the structure.

47962 CX Since (**time\_t**)−1 is a valid return value for a successful call to *mktime()*, an application wishing  
 47963 to check for error situations should set *tm\_wday* to a value less than 0 or greater than 6 before  
 47964 calling *mktime()*. On return, if *tm\_wday* has not changed an error has occurred.

47965 **ERRORS**47966 The *mktime()* function shall fail if:

47967 CX [EOVERFLOW] The result cannot be represented.

47968 **EXAMPLES**

47969 What day of the week is July 4, 2001?

```

47970 #include <stdio.h>
47971 #include <time.h>
47972
47973 struct tm time_str;
47974
47975 char daybuf[20];
47976
47977 int main(void)
47978 {
47979     time_str.tm_year = 2001 - 1900;
47980     time_str.tm_mon = 7 - 1;
47981     time_str.tm_mday = 4;
47982     time_str.tm_hour = 0;
47983     time_str.tm_min = 0;
47984     time_str.tm_sec = 1;
47985     time_str.tm_isdst = -1;
47986     time_str.tm_wday = -1;
47987     if (mktime(&time_str) == (time_t)-1 && time_str.tm_wday == -1)
47988         (void)puts("-unknown-");
47989     else {
47990         (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
47991         (void)puts(daybuf);
47992     }
47993     return 0;
47994 }

```

47992 **APPLICATION USAGE**

47993 When using *mktime()* to add or subtract a fixed time period (one that always corresponds to a  
47994 fixed number of seconds) to or from a broken-down time in the local timezone, reliable results  
47995 for arbitrary TZ can only be assured by using *mktime()* to convert the original broken-down time  
47996 to a time since the Epoch, adding or subtracting the desired number of seconds to that value,  
47997 and then calling *localtime()* with the result. The alternative of adjusting the broken-down time  
47998 before calling *mktime()* may produce unexpected results if the original and updated times are on  
47999 different sides of a geographical timezone change. On implementations that follow the  
48000 recommendation of not range-correcting *tm\_sec* (see step 1 in the DESCRIPTION), reliable  
48001 results can also be assured by adding or subtracting the desired number of seconds to *tm\_sec*  
48002 (and not modifying any other members of the **tm** structure). In applications needing to be  
48003 portable to non-POSIX systems where the **time\_t** encoding is not a count of seconds, it is  
48004 recommended that conditional compilation is used such that the adjustment is performed on the  
48005 *mktime()* return value when possible, and otherwise on the *tm\_sec* member. For timezones that  
48006 are known not to have geographical timezone changes, such as TZ=UTC0, adjustments using just  
48007 *mktime()* do not have this problem.

48008 The way the *mktime()* function interprets out-of-range **tm** structure fields might not produce the  
48009 expected result when multiple adjustments are made at the same time. For example, if an  
48010 application tries to go back one day first and then one year by calling *localtime()*, decrementing  
48011 *tm\_mday* and *tm\_year*, and then calling *mktime()* this would not produce the expected result if it  
48012 was called on 2021-03-01 because *mktime()* would see the supplied year as 2020 (a leap year)

48013 and correct Mar 0 to Feb 29, whereas the intended result was Feb 28. Such issues can be avoided  
48014 by doing multiple adjustments one at a time when the order in which they are done matters.

48015 Examples of how *mktime()* handles some adjustments are:

- 48016 • If given Feb 29 in a non-leap year it treats that as the day after Feb 28 and gives back Mar 1.
- 48017 • If given Feb 0 it treats that as the day before Feb 1 and gives back Jan 31.
- 48018 • If given 21:65 it treats that as 6 minutes after 21:59 and gives back 22:05.
- 48019 • If given *tm\_isdst*=0 for a time when DST is in effect, it gives back a positive *tm\_isdst* and  
48020 alters the other fields appropriately.
- 48021 • If there is a DST transition where 02:00 standard time becomes 03:00 DST and *mktime()* is  
48022 given 02:30 (with negative *tm\_isdst*), it treats that as either 30 minutes after 02:00 standard  
48023 time or 30 minutes before 03:00 DST and gives back a zero or positive *tm\_isdst*,  
48024 respectively, with the *tm\_hour* field altered appropriately.
- 48025 • If a geographical timezone changes its UTC offset such that ``old 00:00`` becomes ``new  
48026 00:30`` and *mktime()* is given 00:20, it treats that as either 20 minutes after ``old 00:00`` or 10  
48027 minutes before ``new 00:30``, and gives back appropriately altered **struct tm** fields.

48028 If an application wants to check whether a given broken-down time is one that is skipped over, it  
48029 can do so by seeing whether the *tm\_mday*, *tm\_hour*, and *tm\_min* values it gets back from *mktime()*  
48030 are the same ones it fed in. Just checking *tm\_hour* and *tm\_min* might appear at first sight to  
48031 suffice, but *tm\_mday* could also change—without *tm\_hour* and *tm\_min* changing—if, for example,  
48032 *TZ* is set to "ABC12XYZ-12" (which might be used in a torture test) or if a geographical  
48033 timezone changes the offset from Coordinated Universal Time of its standard time by 24 hours.

#### 48034 RATIONALE

48035 In order to allow applications to distinguish between a successful return of **(time\_t)-1** and an  
48036 [EOverflow] error, *mktime()* is required not to change *tm\_wday* on error. This mechanism is  
48037 used rather than the convention used for other functions whereby the application sets *errno* to  
48038 zero before the call and the call does not change *errno* on error because the ISO C standard does  
48039 not require *mktime()* to set *errno* on error. The next revision of the ISO C standard is expected to  
48040 require that *mktime()* does not change *tm\_wday* when returning **(time\_t)-1** to indicate an error,  
48041 and that this return convention is used both for the case where the value to be returned by the  
48042 function cannot be represented as a **time\_t** and the case where the value to be returned in the  
48043 *tm\_year* member of the **tm** structure cannot be represented as an **int**.

48044 The DESCRIPTION section says that *mktime()* converts the specified broken-down time into  
48045 a time since the Epoch value. The use of the indefinite article here is necessary because, when  
48046 *tm\_isdst* is negative and the timezone has Daylight Saving Time transitions, there is not a one-to-  
48047 one correspondence between broken-down times and time since the Epoch values.

48048 The description of how the value of *tm\_isdst* affects the behavior of *mktime()* is shaded CX  
48049 because the requirements in the ISO C standard are unclear. The next revision of the ISO C  
48050 standard is expected to state the requirements using wording equivalent to the wording in this  
48051 standard.

48052 Implementations are encouraged not to range-correct *tm\_sec* (see step 1 in the DESCRIPTION) in  
48053 order for the results of making an adjustment to *tm\_sec* always to be equivalent to making the  
48054 same adjustment to the value returned by *mktime()*, even when the original and updated times  
48055 are on different sides of a geographical timezone change. This provides a way for applications to  
48056 do reliable fixed-period adjustment using only *mktime()*, as described in APPLICATION  
48057 USAGE.

48058 The described method for range-correcting the **tm** structure members uses separate variables to  
 48059 hold adjustment values to be applied later to other members, or (for the year adjustment) used  
 48060 in later calculations, because this is one way of avoiding intermediate member values that are  
 48061 not representable as an **int**. Implementations may use other methods; all that is required is that  
 48062 *tm\_year* is the only member for which an [EOverflow] error can occur.

48063 The described method for range-correcting *tm\_mday* would, if implemented that way, be highly  
 48064 inefficient for very large values. The efficiency can be improved by observing that any period of  
 48065 400 years always has the same number of days, so the month-by-month correction method need  
 48066 only be applied for a maximum of 4800 months.

#### 48067 FUTURE DIRECTIONS

48068 A future version of this standard may require that *mktime()* does not perform the optional range  
 48069 correction of the *tm\_sec* member of the **tm** structure described at step 1 in the DESCRIPTION.

48070 A future version of this standard is expected to add a *timegm()* function that is similar to  
 48071 *mktime()*, except that the **tm** structure pointed to by *timeptr* contains a broken-down time in  
 48072 Coordinated Universal Time (rather than the local timezone), where references to *localtime()* are  
 48073 replaced by references to *gmtime()*, and where there are no timezone offset or Daylight Saving  
 48074 Time adjustments. A combination of *gmtime()* and *timegm()* will be the expected way to perform  
 48075 arithmetic upon a **time\_t** value and remain compatible with the ISO C standard (where the  
 48076 internal structure of a **time\_t** is not specified), since attempting such manipulations using  
 48077 *localtime()* and *mktime()* can lead to unexpected results.

#### 48078 SEE ALSO

48079 *asctime()*, *clock()*, *ctime()*, *difftime()*, *futimens()*, *gmtime()*, *localtime()*, *strftime()*, *strptime()*,  
 48080 *time()*, *tzset()*

48081 XBD Section 4.19 (on page 107), <[time.h](#)>

#### 48082 CHANGE HISTORY

48083 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C  
 48084 standard.

#### 48085 Issue 6

48086 Extensions beyond the ISO C standard are marked.

48087 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/58 is applied, updating the RETURN  
 48088 VALUE and ERRORS sections to add the optional [EOverflow] error as a CX extension.

48089 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/59 is applied, adding the *tzset()* function  
 48090 to the SEE ALSO section.

#### 48091 Issue 7

48092 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0393 [104] is applied.

48093 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0228 [724] is applied.

#### 48094 Issue 8

48095 Austin Group Defect 1253 is applied, changing “Daylight Savings” to “Daylight Saving”.

48096 Austin Group Defect 1613 is applied, changing the way the **tm** structure members used by  
 48097 *mktime()* are specified and clarifying that a successful call sets the members to the same values  
 48098 that would be returned by *localtime()*.

48099 Austin Group Defect 1614 is applied, clarifying how *tm\_isdst* is handled and the conditions  
 48100 under which (**time\_t**)-1 is returned.

48101 Austin Group Defect 1627 is applied, clarifying how *mktime()* calculates the time in seconds

48102

since the Epoch from the members of the **tm** structure.



48103 **NAME**48104 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)48105 **SYNOPSIS**

```
48106 MLR #include <sys/mman.h>
48107 int mlock(const void *addr, size_t len);
48108 int munlock(const void *addr, size_t len);
```

48109 **DESCRIPTION**

48110 The *mlock()* function shall cause those whole pages containing any part of the address space of  
 48111 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until  
 48112 unlocked or until the process exits or *execs* another process image. The implementation may  
 48113 require that *addr* be a multiple of {PAGESIZE}.

48114 The *munlock()* function shall unlock those whole pages containing any part of the address space  
 48115 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many  
 48116 times *mlock()* has been called by the process for any of the pages in the specified range. The  
 48117 implementation may require that *addr* be a multiple of {PAGESIZE}.

48118 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address  
 48119 spaces of other processes, any locks established on those pages by another process are  
 48120 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a  
 48121 call to *munlock()* are also mapped into other portions of the address space of the calling process  
 48122 outside the range specified, any locks established on those pages via the other mappings are also  
 48123 unaffected by this call.

48124 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-  
 48125 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked  
 48126 with respect to the address space of the process. Memory residency of unlocked pages is  
 48127 unspecified.

48128 Appropriate privileges are required to lock process memory with *mlock()*.

48129 **RETURN VALUE**

48130 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.  
 48131 Otherwise, no change is made to any locks in the address space of the process, and the function  
 48132 shall return a value of -1 and set *errno* to indicate the error.

48133 **ERRORS**

48134 The *mlock()* and *munlock()* functions shall fail if:

48135 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does  
 48136 not correspond to valid mapped pages in the address space of the process.

48137 The *mlock()* function shall fail if:

48138 [EAGAIN] Some or all of the memory identified by the operation could not be locked  
 48139 when the call was made.

48140 The *mlock()* and *munlock()* functions may fail if:

48141 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

48142 The *mlock()* function may fail if:

48143 [ENOMEM] Locking the pages mapped by the specified range would exceed an  
 48144 implementation-defined limit on the amount of memory that the process may  
 48145 lock.

48146 [EPERM] The calling process does not have appropriate privileges to perform the  
48147 requested operation.

## 48148 EXAMPLES

48149 None.

## 48150 APPLICATION USAGE

48151 None.

## 48152 RATIONALE

48153 None.

## 48154 FUTURE DIRECTIONS

48155 None.

## 48156 SEE ALSO

48157 *exec, exit(), fork(), mlockall(), munmap()*

48158 XBD <sys/mman.h>

## 48159 CHANGE HISTORY

48160 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

### 48161 Issue 6

48162 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.

48163 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
48164 implementation does not support the Range Memory Locking option.

48165 **NAME**48166 mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)48167 **SYNOPSIS**

```
48168 ML #include <sys/mman.h>
48169 int mlockall(int flags);
48170 int munlockall(void);
```

48171 **DESCRIPTION**

48172 The *mlockall()* function shall cause all of the pages mapped by the address space of a process to  
 48173 be memory-resident until unlocked or until the process exits or *execs* another process image. The  
 48174 *flags* argument determines whether the pages to be locked are those currently mapped by the  
 48175 address space of the process, those that are mapped in the future, or both. The *flags* argument is  
 48176 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,  
 48177 defined in *<sys/mman.h>*:

48178 **MCL\_CURRENT** Lock all of the pages currently mapped into the address space of the process.

48179 **MCL\_FUTURE** Lock all of the pages that become mapped into the address space of the  
 48180 process in the future, when those mappings are established.

48181 If **MCL\_FUTURE** is specified, and the automatic locking of future mappings eventually causes  
 48182 the amount of locked memory to exceed the amount of available physical memory or any other  
 48183 implementation-defined limit, the behavior is implementation-defined. The manner in which the  
 48184 implementation informs the application of these situations is also implementation-defined.

48185 The *munlockall()* function shall unlock all currently mapped pages of the address space of the  
 48186 process. Any pages that become mapped into the address space of the process after a call to  
 48187 *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying  
 48188 **MCL\_FUTURE** or a subsequent call to *mlockall()* specifying **MCL\_CURRENT**. If pages mapped  
 48189 into the address space of the process are also mapped into the address spaces of other processes  
 48190 and are locked by those processes, the locks established by the other processes shall be  
 48191 unaffected by a call by this process to *munlockall()*.

48192 Upon successful return from the *mlockall()* function that specifies **MCL\_CURRENT**, all currently  
 48193 mapped pages of the address space of the process shall be memory-resident and locked. Upon  
 48194 return from the *munlockall()* function, all currently mapped pages of the address space of the  
 48195 process shall be unlocked with respect to the address space of the process. The memory  
 48196 residency of unlocked pages is unspecified.

48197 Appropriate privileges are required to lock process memory with *mlockall()*.

48198 **RETURN VALUE**

48199 Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no  
 48200 additional memory shall be locked, and the function shall return a value of  $-1$  and set *errno* to  
 48201 indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address  
 48202 space is unspecified.

48203 If it is supported by the implementation, the *munlockall()* function shall always return a value of  
 48204 zero. Otherwise, the function shall return a value of  $-1$  and set *errno* to indicate the error.

48205 **ERRORS**

48206 The *mlockall()* function shall fail if:

48207 **[EAGAIN]** Some or all of the memory identified by the operation could not be locked  
 48208 when the call was made.

48209	[EINVAL]	The <i>flags</i> argument is zero, or includes unimplemented flags.
48210		The <i>mlockall()</i> function may fail if:
48211	[ENOMEM]	Locking all of the pages currently mapped into the address space of the process would exceed an implementation-defined limit on the amount of memory that the process may lock.
48212		
48213		
48214	[EPERM]	The calling process does not have appropriate privileges to perform the requested operation.
48215		
48216	<b>EXAMPLES</b>	
48217		None.
48218	<b>APPLICATION USAGE</b>	
48219		None.
48220	<b>RATIONALE</b>	
48221		None.
48222	<b>FUTURE DIRECTIONS</b>	
48223		None.
48224	<b>SEE ALSO</b>	
48225		<i>exec</i> , <i>exit()</i> , <i>fork()</i> , <i>mlock()</i> , <i>munmap()</i>
48226		XBD < <a href="#">sys/mman.h</a> >
48227	<b>CHANGE HISTORY</b>	
48228		First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
48229	<b>Issue 6</b>	
48230		The <i>mlockall()</i> and <i>munlockall()</i> functions are marked as part of the Process Memory Locking option.
48231		
48232		The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Memory Locking option.
48233		

48234 **NAME**

48235 mmap — map pages of memory

48236 **SYNOPSIS**

48237 #include &lt;sys/mman.h&gt;

48238 void \*mmap(void \*addr, size\_t len, int prot, int flags,  
48239 int fildes, off\_t off);48240 **DESCRIPTION**48241 The *mmap()* function shall establish a mapping between an address space of a process and a  
48242 memory object.48243 The *mmap()* function shall be supported for the following memory objects:

- 48244 • Regular files
- 48245 • Anonymous memory objects
- 48246 SHM • Shared memory objects
- 48247 TYM • Typed memory objects

48248 Support for any other type of file is unspecified.

48249 The format of the call is as follows:

48250 *pa*=mmap(*addr*, *len*, *prot*, *flags*, *fildes*, *off*);

48251 The *mmap()* function shall establish a mapping between the address space of the process at an  
48252 address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off*  
48253 for *len* bytes, or to an anonymous memory object of *len* bytes. The value of *pa* is an  
48254 implementation-defined function of the parameter *addr* and the values of *flags*, further described  
48255 below. A successful *mmap()* call shall return *pa* as its result. The address range starting at *pa* and  
48256 continuing for *len* bytes shall be legitimate for the possible (not necessarily current) address  
48257 space of the process. The range of bytes starting at *off* and continuing for *len* bytes shall be  
48258 legitimate for the possible (not necessarily current) offsets in the memory object represented by  
48259 *fildes*.

48260 TYM If *fildes* represents a typed memory object opened with either the  
48261 POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG  
48262 flag, the memory object to be mapped shall be that portion of the typed memory object allocated  
48263 by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap()*  
48264 is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling  
48265 process, *mmap()* shall fail.

48266 The mapping established by *mmap()* shall replace any previous mappings for those whole pages  
48267 containing any part of the address space of the process starting at *pa* and continuing for *len*  
48268 bytes.

48269 If the size of the mapped file changes after the call to *mmap()* as a result of some other operation  
48270 on the mapped file, the effect of references to portions of the mapped region that correspond to  
48271 added or removed portions of the file is unspecified.

48272 If *len* is zero, *mmap()* shall fail and no mapping shall be established.

48273 The parameter *prot* determines whether read, write, execute, or some combination of accesses  
48274 are permitted to the data being mapped. The *prot* shall be either PROT\_NONE or the bitwise-  
48275 inclusive OR of one or more of the other flags in the following table, defined in the  
48276 <sys/mman.h> header.

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap()* shall fail.

An implementation may permit accesses other than those specified by *prot*; however, the implementation shall not permit a write to succeed where PROT\_WRITE has not been set and shall not permit any access where PROT\_NONE alone has been set. The implementation shall support at least the following values of *prot*: PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-inclusive OR of PROT\_READ and PROT\_WRITE. The file descriptor *fildev* shall have been opened with read permission, regardless of the protection options specified. If PROT\_WRITE is specified, the application shall ensure that it has opened the file descriptor *fildev* with write permission unless MAP\_PRIVATE is specified in the *flags* parameter as described below.

The parameter *flags* provides other information about the handling of the mapped data. The value of *flags* is the bitwise-inclusive OR of these options, defined in `<sys/mman.h>`:

Symbolic Constant	Description
MAP_ANON	Synonym for MAP_ANONYMOUS.
MAP_ANONYMOUS	Map anonymous memory.
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret <i>addr</i> exactly.

It is implementation-defined whether MAP\_FIXED shall be supported. MAP\_FIXED shall be supported on XSI-conformant systems.

MAP\_SHARED and MAP\_PRIVATE describe the disposition of write references to the memory object. If MAP\_SHARED is specified, write references shall change the underlying object. If MAP\_PRIVATE is specified, modifications to the mapped data by the calling process shall be visible only to the calling process and shall not change the underlying object. It is unspecified whether modifications to the underlying object done after the MAP\_PRIVATE mapping is established are visible through the MAP\_PRIVATE mapping. Either MAP\_SHARED or MAP\_PRIVATE can be specified, but not both. The mapping type is retained across *fork()*.

The state of synchronization objects such as mutexes, semaphores, barriers, and conditional variables placed in shared memory mapped with MAP\_SHARED becomes undefined when the last region in any process containing the synchronization object is unmapped.

When *fildev* represents a typed memory object opened with either the POSIX\_TYPED\_MEM\_ALLOCATE flag or the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, *mmap()* shall, if there are enough resources available, map *len* bytes allocated from the corresponding typed memory object which were not previously allocated to any process in any processor that may access that typed memory object. If there are not enough resources available, the function shall fail. If *fildev* represents a typed memory object opened with the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag, these allocated bytes shall be contiguous within the typed memory object. If *fildev* represents a typed memory object opened with the POSIX\_TYPED\_MEM\_ALLOCATE flag, these allocated bytes may be composed of non-contiguous fragments within the typed memory object. If *fildev* represents a typed memory object opened with neither the POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG flag nor the

48324 POSIX\_TYPED\_MEM\_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory  
 48325 object are mapped, exactly as when mapping a file or shared memory object. In this case, if two  
 48326 processes map an area of typed memory using the same *off* and *len* values and using file  
 48327 descriptors that refer to the same memory pool (either from the same port or from a different  
 48328 port), both processes shall map the same region of storage.

48329 When MAP\_FIXED is set in the *flags* argument, the implementation is informed that the value of  
 48330 *pa* shall be *addr*, exactly. If MAP\_FIXED is set, *mmap()* may return MAP\_FAILED and set *errno* to  
 48331 ML|MLR [EINVAL]. If a MAP\_FIXED request is successful, then any previous mappings or memory  
 48332 locks for those whole pages containing any part of the address range [*pa*,*pa+len*) shall be  
 48333 removed, as if by an appropriate call to *munmap()*, before the new mapping is established.

48334 When MAP\_FIXED is not set, the implementation uses *addr* in an implementation-defined  
 48335 manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the  
 48336 implementation deems suitable for a mapping of *len* bytes to the file. All implementations  
 48337 interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*,  
 48338 subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a  
 48339 process address near which the mapping should be placed. When the implementation selects a  
 48340 value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

48341 If MAP\_FIXED is specified and *addr* is non-zero, it shall have the same remainder as the *off*  
 48342 parameter, modulo the page size as returned by *sysconf()* when passed *\_SC\_PAGESIZE* or  
 48343 *\_SC\_PAGE\_SIZE*. The implementation may require that *off* is a multiple of the page size. If  
 48344 MAP\_FIXED is specified, the implementation may require that *addr* is a multiple of the page  
 48345 size. The system performs mapping operations over whole pages. Thus, while the parameter *len*  
 48346 need not meet a size or alignment constraint, the system shall include, in any mapping  
 48347 operation, any partial page specified by the address range starting at *pa* and continuing for *len*  
 48348 bytes.

48349 If MAP\_ANONYMOUS (or its synonym MAP\_ANON) is specified, *fildev* is *-1*, and *off* is 0, then  
 48350 *mmap()* shall ignore *fildev* and instead establish a mapping to a new anonymous memory object  
 48351 of size *len*. The effect of specifying MAP\_ANONYMOUS (or MAP\_ANON) with other values of  
 48352 *fildev* or *off* is unspecified. Anonymous memory objects shall be initialized to all bits zero.

48353 The system shall always zero-fill any partial page at the end of an object. Further, the system  
 48354 shall never write out any modified portions of the last page of an object which are beyond its  
 48355 end. References within the address range starting at *pa* and continuing for *len* bytes to whole  
 48356 pages following the end of an object shall result in delivery of a SIGBUS signal.

48357 An implementation may generate SIGBUS signals when a reference would cause an error in the  
 48358 mapped object, such as out-of-space condition.

48359 The *mmap()* function shall add an extra reference to the file associated with the file descriptor  
 48360 *fildev* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be  
 48361 removed when there are no more mappings to the file.

48362 The last data access timestamp of the mapped file may be marked for update at any time  
 48363 between the *mmap()* call and the corresponding *munmap()* call. The initial read or write  
 48364 reference to a mapped region shall cause the file's last data access timestamp to be marked for  
 48365 update if it has not already been marked for update.

48366 The last data modification and last file status change timestamps of a file that is mapped with  
 48367 MAP\_SHARED and PROT\_WRITE shall be marked for update at some point in the interval  
 48368 between a write reference to the mapped region and the next call to *msync()* with MS\_ASYNC or  
 48369 MS\_SYNC for that portion of the file by any process. If there is no such call and if the  
 48370 underlying file is modified as a result of a write reference, then these timestamps shall be

- 48371 marked for update at some time after the write reference.
- 48372 There may be implementation-defined limits on the number of memory regions that can be  
48373 mapped (per process or per system).
- 48374 XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a  
48375 process is decreased by the use of *shmat()* is implementation-defined.
- 48376 If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the  
48377 mappings in the address range starting at *addr* and continuing for *len* bytes may have been  
48378 unmapped.
- 48379 **RETURN VALUE**
- 48380 Upon successful completion, the *mmap()* function shall return the address at which the mapping  
48381 was placed (*pa*); otherwise, it shall return a value of MAP\_FAILED and set *errno* to indicate the  
48382 error. The symbol MAP\_FAILED is defined in the `<sys/mman.h>` header. No successful return  
48383 from *mmap()* shall return the value MAP\_FAILED.
- 48384 **ERRORS**
- 48385 The *mmap()* function shall fail if:
- 48386 ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall()*, due to  
48387 a lack of resources.
- 48388 [EINVAL] The value of *len* is zero.
- 48389 [EINVAL] The value of *flags* is invalid (neither MAP\_PRIVATE nor MAP\_SHARED is  
48390 set).
- 48391 [EMFILE] The number of mapped regions would exceed an implementation-defined  
48392 limit (per process or per system).
- 48393 [ENOMEM] MAP\_FIXED was specified, and the range [*addr,addr+len*) exceeds that allowed  
48394 for the address space of a process; or, if MAP\_FIXED was not specified and  
48395 there is insufficient room in the address space to effect the mapping.
- 48396 ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall()*,  
48397 because it would require more space than the system is able to supply.
- 48398 TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory  
48399 object designated by *fildev* to allocate *len* bytes.
- 48400 [ENOTSUP] MAP\_FIXED or MAP\_PRIVATE was specified in the *flags* argument and the  
48401 implementation does not support this functionality.
- 48402 The implementation does not support the combination of accesses requested  
48403 in the *prot* argument.
- 48404 [ENXIO] MAP\_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is  
48405 invalid for the specified object.
- 48406 TYM [ENXIO] The *fildev* argument refers to a typed memory object that is not accessible from  
48407 the calling process.
- 48408 In addition, if MAP\_ANONYMOUS (or MAP\_ANON) is not set in *flags*, the *mmap()* function  
48409 shall fail if:
- 48410 [EACCES] The *fildev* argument is not open for read, regardless of the protection specified,  
48411 or *fildev* is not open for write and PROT\_WRITE was specified for a  
48412 MAP\_SHARED type mapping.



- 48413 [EBADF] The *fildes* argument is not a valid open file descriptor.
- 48414 [ENODEV] The *fildes* argument refers to a file whose type is not supported by *mmap()*.
- 48415 [EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset maximum established in the open file description associated with *fildes*.
- 48416
- 48417 [ENXIO] Addresses in the range [*off*,*off+len*) are invalid for the object specified by *fildes*.
- 48418 The *mmap()* function may fail if:
- 48419 [EINVAL] The *addr* argument (if MAP\_FIXED was specified) or *off* is not a multiple of the page size as returned by *sysconf()*, or is considered invalid by the implementation.
- 48420
- 48421

#### 48422 EXAMPLES

48423 None.

#### 48424 APPLICATION USAGE

48425 Use of *mmap()* may reduce the amount of memory available to other memory allocation functions.

48426

48427 Use of MAP\_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*. The use of MAP\_FIXED is discouraged, as it may prevent an implementation from making the most effective use of resources. Most implementations require that *off* and *addr* are multiples of the page size as returned by *sysconf()*.

48428

48429

48430

48431 The application must ensure correct synchronization when using *mmap()* in conjunction with any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

48432

48433 The *mmap()* function allows access to resources via address space manipulations, instead of *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the address to which the file was mapped. So, using pseudo-code to illustrate the way in which an existing program might be changed to use *mmap()*, the following:

48434

48435

48436

```
48437 fildes = open(...)
48438 lseek(fildes, some_offset)
48439 read(fildes, buf, len)
48440 /* Use data in buf. */
```

48441 becomes:

```
48442 fildes = open(...)
48443 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
48444 /* Use data at address. */
```

#### 48445 RATIONALE

48446 After considering several other alternatives, it was decided to adopt the *mmap()* definition found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is minimal, in that it describes only what has been built, and what appears to be necessary for a general and portable mapping facility.

48447

48448

48449

48450 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose mapping facility. It can be used to map any appropriate object, such as memory, files, devices, and so on, into the address space of a process.

48451

48452

48453 When a mapping is established, it is possible that the implementation may need to map more than is requested into the address space of the process because of hardware requirements. An application, however, cannot count on this behavior. Implementations that do not use a paged architecture may simply allocate a common memory region and return the address of it; such

48454

48455

48456

48457 implementations probably do not allocate any more than is necessary. References past the end of  
48458 the requested area are unspecified.

48459 If an application requests a mapping that overlaps existing mappings in the process, it might be  
48460 desirable that an implementation detect this and inform the application. However, if the  
48461 program specifies a fixed address mapping (which requires some implementation knowledge to  
48462 determine a suitable address, if the function is supported at all), then the program is presumed  
48463 to be successfully managing its own address space and should be trusted when it asks to map  
48464 over existing data structures. Furthermore, it is also desirable to make as few system calls as  
48465 possible, and it might be considered onerous to require an *munmap()* before an *mmap()* to the  
48466 same address range. This volume of POSIX.1-2024 specifies that the new mapping replaces any  
48467 existing mappings (implying an automatic *munmap()* on the address range), following existing  
48468 practice in this regard. The standard developers also considered whether there should be a way  
48469 for new mappings to overlay existing mappings, but found no existing practice for this.

48470 It is not expected that all hardware implementations are able to support all combinations of  
48471 permissions at all addresses. Implementations are required to disallow write access to mappings  
48472 without write permission and to disallow access to mappings without any access permission.  
48473 Other than these restrictions, implementations may allow access types other than those  
48474 requested by the application. For example, if the application requests only *PROT\_WRITE*, the  
48475 implementation may also allow read access. A call to *mmap()* fails if the implementation cannot  
48476 support allowing all the access requested by the application. For example, some  
48477 implementations cannot support a request for both write access and execute access  
48478 simultaneously. All implementations must support requests for no access, read access, write  
48479 access, and both read and write access. Strictly conforming code must only rely on the required  
48480 checks. These restrictions allow for portability across a wide range of hardware.

48481 The *MAP\_FIXED* address treatment is likely to fail for non-page-aligned values and for certain  
48482 architecture-dependent address ranges. Conforming implementations cannot count on being  
48483 able to choose address values for *MAP\_FIXED* without utilizing non-portable, implementation-  
48484 defined knowledge. Nonetheless, *MAP\_FIXED* is provided as a standard interface conforming  
48485 to existing practice for utilizing such knowledge when it is available.

48486 Similarly, in order to allow implementations that do not support virtual addresses, support for  
48487 directly specifying any mapping addresses via *MAP\_FIXED* is not required and thus a  
48488 conforming application may not count on it.

48489 The *MAP\_PRIVATE* function can be implemented efficiently when memory protection hardware  
48490 is available. When such hardware is not available, implementations can implement such  
48491 “mappings” by simply making a real copy of the relevant data into process private memory,  
48492 though this tends to behave similarly to *read()*.

48493 The function has been defined to allow for many different models of using shared memory.  
48494 However, all uses are not equally portable across all machine architectures. In particular, the  
48495 *mmap()* function allows the system as well as the application to specify the address at which to  
48496 map a specific region of a memory object. The most portable way to use the function is always to  
48497 let the system choose the address, specifying *NULL* as the value for the argument *addr* and not  
48498 to specify *MAP\_FIXED*.

48499 If it is intended that a particular region of a memory object be mapped at the same address in a  
48500 group of processes (on machines where this is even possible), then *MAP\_FIXED* can be used to  
48501 pass in the desired mapping address. The system can still be used to choose the desired address  
48502 if the first such mapping is made without specifying *MAP\_FIXED*, and then the resulting  
48503 mapping address can be passed to subsequent processes for them to pass in via *MAP\_FIXED*.  
48504 The availability of a specific address range cannot be guaranteed, in general.

48505 The `mmap()` function can be used to map a region of memory that is larger than the current size  
 48506 of the object. Memory access within the mapping but beyond the current end of the underlying  
 48507 objects may result in SIGBUS signals being sent to the process. The reason for this is that the size  
 48508 of the object can be manipulated by other processes and can change at any moment. The  
 48509 implementation should tell the application that a memory reference is outside the object where  
 48510 this can be detected; otherwise, written data may be lost and read data may not reflect actual  
 48511 data in the object.

48512 Note that references beyond the end of the object do not extend the object as the new end cannot  
 48513 be determined precisely by most virtual memory hardware. Instead, the size can be directly  
 48514 manipulated by `ftruncate()`.

48515 Process memory locking does apply to shared memory regions, and the `MCL_FUTURE`  
 48516 argument to `mlockall()` can be relied upon to cause new shared memory regions to be  
 48517 automatically locked.

48518 Existing implementations of `mmap()` return the value `-1` when unsuccessful. Since the casting of  
 48519 this value to type `void *` cannot be guaranteed by the ISO C standard to be distinct from a  
 48520 successful value, this volume of POSIX.1-2024 defines the symbol `MAP_FAILED`, which a  
 48521 conforming implementation does not return as the result of a successful call.

48522 Some historical implementations only supported `MAP_ANON`, some only supported  
 48523 `MAP_ANONYMOUS`, and some supported both spellings. This standard includes both  
 48524 spellings partly for application compatibility and partly because neither spelling was clearly  
 48525 more popular than the other at the time this feature was considered for standardization.

#### 48526 FUTURE DIRECTIONS

48527 None.

#### 48528 SEE ALSO

48529 [exec](#), [fcntl\(\)](#), [fork\(\)](#), [lockf\(\)](#), [msync\(\)](#), [munmap\(\)](#), [mprotect\(\)](#), [posix\\_typed\\_mem\\_open\(\)](#), [shmat\(\)](#),  
 48530 [sysconf\(\)](#)

48531 XBD [<sys/mman.h>](#)

#### 48532 CHANGE HISTORY

48533 First released in Issue 4, Version 2.

#### 48534 Issue 5

48535 Moved from X/OPEN UNIX extension to BASE.

48536 Aligned with `mmap()` in the POSIX Realtime Extension as follows:

- 48537 • The DESCRIPTION is extensively reworded.
- 48538 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 48539 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 48540 • The value returned on failure is the value of the constant `MAP_FAILED`; this was  
 48541 previously defined as `-1`.

48542 Large File Summit extensions are added.

#### 48543 Issue 6

48544 The `mmap()` function is marked as part of the Memory Mapped Files option.

48545 The Open Group Corrigendum U028/6 is applied, changing `(void *)-1` to `MAP_FAILED`.

48546 The following new requirements on POSIX implementations derive from alignment with the  
 48547 Single UNIX Specification:

- 48548 • The DESCRIPTION is updated to describe the use of MAP\_FIXED.
- 48549 • The DESCRIPTION is updated to describe the addition of an extra reference to the file  
48550 associated with the file descriptor passed to *mmap()*.
- 48551 • The DESCRIPTION is updated to state that there may be implementation-defined limits on  
48552 the number of memory regions that can be mapped.
- 48553 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*  
48554 argument.
- 48555 • The [EINVAL] and [EMFILE] error conditions are added.
- 48556 • The [EOVERFLOW] error condition is added. This change is to support large files.
- 48557 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 48558 • The DESCRIPTION is updated to describe the cases when MAP\_PRIVATE and  
48559 MAP\_FIXED need not be supported.
- 48560 The following changes are made for alignment with IEEE Std 1003.1j-2000:
- 48561 • Semantics for typed memory objects are added to the DESCRIPTION.
- 48562 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 48563 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.
- 48564 The normative text is updated to avoid use of the term “must” for application requirements.
- 48565 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code  
48566 in the SYNOPSIS from MF | SHM to MC3 (notation for MF | SHM | TYM).
- 48567 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/60 is applied, updating the  
48568 DESCRIPTION and ERRORS sections to add the [EINVAL] error when *len* is zero.
- 48569 **Issue 7**
- 48570 Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment  
48571 requirements and adding a note about the state of synchronization objects becoming undefined  
48572 when a shared region is unmapped.
- 48573 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to  
48574 the Base.
- 48575 Changes are made related to support for finegrained timestamps.
- 48576 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0229 [852] is applied.
- 48577 **Issue 8**
- 48578 Austin Group Defect 850 is applied, adding anonymous memory objects.

48579 **NAME**

48580 modf, modff, modfl — decompose a floating-point number

48581 **SYNOPSIS**

48582 #include &lt;math.h&gt;

48583 double modf(double *x*, double \**iptr*);48584 float modff(float *value*, float \**iptr*);48585 long double modfl(long double *value*, long double \**iptr*);48586 **DESCRIPTION**

48587 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 48588 conflict between the requirements described here and the ISO C standard is unintentional. This  
 48589 volume of POSIX.1-2024 defers to the ISO C standard.

48590 These functions shall break the argument *x* into integral and fractional parts, each of which has  
 48591 the same sign as the argument. It stores the integral part as a **double** (for the *modf()* function), a  
 48592 **float** (for the *modff()* function), or a **long double** (for the *modfl()* function), in the object pointed  
 48593 to by *iptr*.

48594 **RETURN VALUE**48595 Upon successful completion, these functions shall return the signed fractional part of *x*.

48596 MX The returned value shall be exact and shall be independent of the current rounding direction  
 48597 mode.

48598 If *x* is NaN, a NaN shall be returned, and \**iptr* shall be set to a NaN.48599 If *x* is  $\pm\text{Inf}$ ,  $\pm 0$  shall be returned, and \**iptr* shall be set to  $\pm\text{Inf}$ .48600 **ERRORS**

48601 No errors are defined.

48602 **EXAMPLES**

48603 None.

48604 **APPLICATION USAGE**48605 The *modf()* function computes the function result and \**iptr* such that:48606 `a = modf(x, iptr) ;`48607 `x == a+*iptr ;`

48608 allowing for the usual floating-point inaccuracies.

48609 **RATIONALE**

48610 None.

48611 **FUTURE DIRECTIONS**

48612 None.

48613 **SEE ALSO**48614 *frexp()*, *isnan()*, *ldexp()*

48615 XBD &lt;math.h&gt;

48616 **CHANGE HISTORY**

48617 First released in Issue 1. Derived from Issue 1 of the SVID.

48618 **Issue 5**

48619 The DESCRIPTION is updated to indicate how an application should check for an error. This  
48620 text was previously published in the APPLICATION USAGE section.

48621 **Issue 6**

48622 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999  
48623 standard.

48624 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
48625 revised to align with the ISO/IEC 9899:1999 standard.

48626 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
48627 marked.

48628 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example  
48629 in the APPLICATION USAGE section.

48630 **Issue 8**

48631 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
48632 standard.

48633 **NAME**

48634 mprotect — set protection of memory mapping

48635 **SYNOPSIS**

48636 #include &lt;sys/mman.h&gt;

48637 int mprotect(void \*addr, size\_t len, int prot);

48638 **DESCRIPTION**

48639 The *mprotect()* function shall change the access protections to be that specified by *prot* for those  
 48640 whole pages containing any part of the address space of the process starting at address *addr* and  
 48641 continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some  
 48642 combination of accesses are permitted to the data being mapped. The *prot* argument should be  
 48643 either PROT\_NONE or the bitwise-inclusive OR of one or more of PROT\_READ, PROT\_WRITE,  
 48644 and PROT\_EXEC.

48645 If an implementation cannot support the combination of access types specified by *prot*, the call to  
 48646 *mprotect()* shall fail.

48647 An implementation may permit accesses other than those specified by *prot*; however, no  
 48648 implementation shall permit a write to succeed where PROT\_WRITE has not been set or shall  
 48649 permit any access where PROT\_NONE alone has been set. Implementations shall support at  
 48650 least the following values of *prot*: PROT\_NONE, PROT\_READ, PROT\_WRITE, and the bitwise-  
 48651 inclusive OR of PROT\_READ and PROT\_WRITE. If PROT\_WRITE is specified, the application  
 48652 shall ensure that it has opened the mapped objects in the specified address range with write  
 48653 permission, unless MAP\_PRIVATE was specified in the original mapping, regardless of whether  
 48654 the file descriptors used to map the objects have since been closed.

48655 The implementation may require that *addr* be a multiple of the page size as returned by  
 48656 *sysconf()*.

48657 The behavior of this function is unspecified if the mapping was not established by a call to  
 48658 *mmap()*.

48659 When *mprotect()* fails for reasons other than [EINVAL], the protections on some of the pages in  
 48660 the range [*addr,addr+len*) may have been changed.

48661 **RETURN VALUE**

48662 Upon successful completion, *mprotect()* shall return 0; otherwise, it shall return -1 and set *errno*  
 48663 to indicate the error.

48664 **ERRORS**

48665 The *mprotect()* function shall fail if:

48666 [EACCES] The *prot* argument specifies a protection that violates the access permission the  
 48667 process has to the underlying memory object.

48668 [EAGAIN] The *prot* argument specifies PROT\_WRITE over a MAP\_PRIVATE mapping  
 48669 and there are insufficient memory resources to reserve for locking the private  
 48670 page.

48671 [ENOMEM] Addresses in the range [*addr,addr+len*) are invalid for the address space of a  
 48672 process, or specify one or more pages which are not mapped.

48673 [ENOMEM] The *prot* argument specifies PROT\_WRITE on a MAP\_PRIVATE mapping, and  
 48674 it would require more space than the system is able to supply for locking the  
 48675 private pages, if required.

48676 [ENOTSUP] The implementation does not support the combination of accesses requested  
48677 in the *prot* argument.

48678 The *mprotect()* function may fail if:

48679 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

#### 48680 EXAMPLES

48681 None.

#### 48682 APPLICATION USAGE

48683 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.

#### 48684 RATIONALE

48685 None.

#### 48686 FUTURE DIRECTIONS

48687 None.

#### 48688 SEE ALSO

48689 *mmap()*, *sysconf()*

48690 XBD <[sys/mman.h](#)>

#### 48691 CHANGE HISTORY

48692 First released in Issue 4, Version 2.

#### 48693 Issue 5

48694 Moved from X/OPEN UNIX extension to BASE.

48695 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:

- 48696 • The DESCRIPTION is largely reworded.
- 48697 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- 48698 • [EAGAIN] is moved from the optional to the mandatory error conditions.

#### 48699 Issue 6

48700 The *mprotect()* function is marked as part of the Memory Protection option.

48701 The following new requirements on POSIX implementations derive from alignment with the  
48702 Single UNIX Specification:

- 48703 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple  
48704 of the page size as returned by *sysconf()*.
- 48705 • The [EINVAL] error condition is added.

48706 The normative text is updated to avoid use of the term “must” for application requirements.

#### 48707 Issue 7

48708 SD5-XSH-ERN-22 is applied, deleting erroneous APPLICATION USAGE.

48709 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment  
48710 requirements.

48711 The *mprotect()* function is moved from the Memory Protection option to the Base.



48712 **NAME**48713 mq\_close — close a message queue (**REALTIME**)48714 **SYNOPSIS**

```
48715 MSG #include <mqueue.h>
48716 int mq_close(mqd_t mqdes);
```

48717 **DESCRIPTION**

48718 The *mq\_close()* function shall remove the association between the message queue descriptor, *mqdes*, and its message queue. The results of using this message queue descriptor after successful return from this *mq\_close()*, and until the return of this message queue descriptor from a subsequent *mq\_open()*, are undefined.

48722 If a message queue descriptor is implemented using a file descriptor, *mq\_close()* shall close the file descriptor.

48724 If the process has successfully attached a notification request to the message queue via this *mqdes*, this attachment shall be removed, and the message queue is available for another process to attach for notification.

48727 **RETURN VALUE**

48728 Upon successful completion, the *mq\_close()* function shall return a value of zero; otherwise, the function shall return a value of  $-1$  and set *errno* to indicate the error.

48730 **ERRORS**

48731 The *mq\_close()* function shall fail if:

48732 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

48733 **EXAMPLES**

48734 None.

48735 **APPLICATION USAGE**

48736 None.

48737 **RATIONALE**

48738 None.

48739 **FUTURE DIRECTIONS**

48740 None.

48741 **SEE ALSO**

48742 [mq\\_open\(\)](#), [mq\\_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

48743 XBD [<mqueue.h>](#)

48744 **CHANGE HISTORY**

48745 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

48746 **Issue 6**

48747 The *mq\_close()* function is marked as part of the Message Passing option.

48748 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

48750 **Issue 8**

48751 Austin Group Defect 368 is applied, adding a requirement that if a message queue descriptor is implemented using a file descriptor, *mq\_close()* closes the file descriptor.

48753 **NAME**48754 mq\_getattr — get message queue attributes (**REALTIME**)48755 **SYNOPSIS**

```
48756 MSG #include <mqueue.h>
48757 int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

48758 **DESCRIPTION**48759 The *mq\_getattr()* function shall obtain status information and attributes of the message queue  
48760 and the open message queue description associated with the message queue descriptor.48761 The *mqdes* argument specifies a message queue descriptor.48762 The results shall be returned in the **mq\_attr** structure referenced by the *mqstat* argument.48763 Upon return, the following members shall have the values associated with the open message  
48764 queue description as set when the message queue was opened and as modified by subsequent  
48765 *mq\_setattr()* calls: *mq\_flags*.48766 The following attributes of the message queue shall be returned as set at message queue  
48767 creation: *mq\_maxmsg*, *mq\_msgsize*.48768 Upon return, the following members within the **mq\_attr** structure referenced by the *mqstat*  
48769 argument shall be set to the current state of the message queue:48770 *mq\_curmsgs* The number of messages currently on the queue.48771 **RETURN VALUE**48772 Upon successful completion, the *mq\_getattr()* function shall return zero. Otherwise, the function  
48773 shall return  $-1$  and set *errno* to indicate the error.48774 **ERRORS**48775 The *mq\_getattr()* function may fail if:48776 [EBADF] The *mqdes* argument is not a valid message queue descriptor.48777 **EXAMPLES**48778 See *mq\_notify()*.48779 **APPLICATION USAGE**

48780 None.

48781 **RATIONALE**

48782 None.

48783 **FUTURE DIRECTIONS**

48784 None.

48785 **SEE ALSO**48786 *mq\_notify()*, *mq\_open()*, *mq\_send()*, *mq\_setattr()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*48787 XBD <**mqueue.h**>48788 **CHANGE HISTORY**

48789 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

48790 **Issue 6**48791 The *mq\_getattr()* function is marked as part of the Message Passing option.48792 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
48793 implementation does not support the Message Passing option.

48794  
48795

The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

48796  
48797

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/61 is applied, updating the ERRORS section to change the [EBADF] error from mandatory to optional.

48798 **NAME**48799 mq\_notify — notify process that a message is available (**REALTIME**)48800 **SYNOPSIS**

```
48801 MSG #include <mqueue.h>
48802 int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

48803 **DESCRIPTION**

48804 If the argument *notification* is not NULL, this function shall register the calling process to be  
 48805 notified of message arrival at an empty message queue associated with the specified message  
 48806 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to  
 48807 the process when the message queue transitions from empty to non-empty. At any time, only  
 48808 one process may be registered for notification by a message queue. If the calling process or any  
 48809 other process has already registered for notification of message arrival at the specified message  
 48810 queue, subsequent attempts to register for that message queue shall fail.

48811 If *notification* is NULL and the process is currently registered for notification by the specified  
 48812 message queue, the existing registration shall be removed.

48813 When the notification is sent to the registered process, its registration shall be removed. The  
 48814 message queue shall then be available for registration.

48815 If a process has registered for notification of message arrival at a message queue and some  
 48816 thread is blocked in *mq\_receive()* or *mq\_timedreceive()* waiting to receive a message when a  
 48817 message arrives at the queue, the arriving message shall satisfy the appropriate *mq\_receive()* or  
 48818 *mq\_timedreceive()*, respectively. The resulting behavior is as if the message queue remains empty,  
 48819 and no notification shall be sent.

48820 **RETURN VALUE**

48821 Upon successful completion, the *mq\_notify()* function shall return a value of zero; otherwise, the  
 48822 function shall return a value of  $-1$  and set *errno* to indicate the error.

48823 **ERRORS**

48824 The *mq\_notify()* function shall fail if:

48825 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

48826 [EBUSY] A process is already registered for notification by the message queue.

48827 The *mq\_notify()* function may fail if:

48828 [EINVAL] The *notification* argument is NULL and the process is currently not registered.

48829 **EXAMPLES**

48830 The following program registers a notification request for the message queue named in its  
 48831 command-line argument. Notification is performed by creating a thread. The thread executes a  
 48832 function which reads one message from the queue and then terminates the process.

```
48833 #include <pthread.h>
48834 #include <signal.h>
48835 #include <mqueue.h>
48836 #include <assert.h>
48837 #include <stdio.h>
48838 #include <stdlib.h>
48839 #include <unistd.h>

48840 static void /* Thread start function */
48841 tfunc(union sigval sv)
```

```

48842     {
48843         struct mq_attr attr;
48844         ssize_t nr;
48845         void *buf;
48846         mqd_t mqdes = *((mqd_t *) sv.sival_ptr);
48847
48848         /* Determine maximum msg size; allocate buffer to receive msg */
48849         if (mq_getattr(mqdes, &attr) == -1) {
48850             perror("mq_getattr");
48851             exit(EXIT_FAILURE);
48852         }
48853         buf = malloc(attr.mq_msgsize);
48854
48855         if (buf == NULL) {
48856             perror("malloc");
48857             exit(EXIT_FAILURE);
48858         }
48859         nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
48860         if (nr == -1) {
48861             perror("mq_receive");
48862             exit(EXIT_FAILURE);
48863         }
48864         printf("Read %ld bytes from message queue\n", (long) nr);
48865         free(buf);
48866         exit(EXIT_SUCCESS);      /* Terminate the process */
48867     }
48868
48869     int
48870     main(int argc, char *argv[])
48871     {
48872         mqd_t mqdes;
48873         struct sigevent not;
48874
48875         assert(argc == 2);
48876
48877         mqdes = mq_open(argv[1], O_RDONLY);
48878         if (mqdes == (mqd_t) -1) {
48879             perror("mq_open");
48880             exit(EXIT_FAILURE);
48881         }
48882         not.sigev_notify = SIGEV_THREAD;
48883         not.sigev_notify_function = tfunc;
48884         not.sigev_notify_attributes = NULL;
48885         not.sigev_value.sival_ptr = &mqdes;    /* Arg. to thread func. */
48886         if (mq_notify(mqdes, &not) == -1) {
48887             perror("mq_notify");
48888             exit(EXIT_FAILURE);
48889         }
48890
48891         pause();    /* Process will be terminated by thread function */
48892     }

```

48887 **APPLICATION USAGE**

48888 Since the `<mqueue.h>` header is only required to declare the `sigevent` structure tag as naming  
48889 an incomplete structure type, in order to use `mq_notify()` and pass it a pointer to a `sigevent`  
48890 structure, applications need to include `<signal.h>` so that `sigevent` will be fully defined.

48891 **RATIONALE**

48892 None.

48893 **FUTURE DIRECTIONS**

48894 None.

48895 **SEE ALSO**

48896 `mq_open()`, `mq_send()`, `mq_receive()`, `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`

48897 XBD `<mqueue.h>`

48898 **CHANGE HISTORY**

48899 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

48900 **Issue 6**

48901 The `mq_notify()` function is marked as part of the Message Passing option.

48902 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
48903 implementation does not support the Message Passing option.

48904 The `mq_timedsend()` function is added to the SEE ALSO section for alignment with IEEE Std  
48905 1003.1d-1999.

48906 **Issue 7**

48907 SD5-XSH-ERN-38 is applied, adding the `mq_timedreceive()` function to the DESCRIPTION.

48908 Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.

48909 An example is added.

48910 **Issue 8**

48911 Austin Group Defect 1282 is applied, changing the EXAMPLES and APPLICATION USAGE  
48912 sections.

48913 **NAME**48914 mq\_open — open a message queue (**REALTIME**)48915 **SYNOPSIS**

```
48916 MSG #include <mqueue.h>
48917 mqd_t mq_open(const char *name, int oflag, ...);
```

48918 **DESCRIPTION**

48919 The *mq\_open()* function shall establish the connection between a process and a message queue  
 48920 with a message queue descriptor. It shall create an open message queue description that refers to  
 48921 the message queue, and a message queue descriptor that refers to that open message queue  
 48922 description. The message queue descriptor is used by other functions to refer to that message  
 48923 queue. The *name* argument points to a string naming a message queue. It is unspecified whether  
 48924 the name appears in the file system and is visible to other functions that take pathnames as  
 48925 arguments. The *name* argument conforms to the construction rules for a pathname, except that  
 48926 the interpretation of <slash> characters other than the leading <slash> character in *name* is  
 48927 implementation-defined, and that the length limits for the *name* argument are implementation-  
 48928 defined and need not be the same as the pathname limits {PATH\_MAX} and {NAME\_MAX}. If  
 48929 *name* begins with the <slash> character, then processes calling *mq\_open()* with the same value of  
 48930 *name* shall refer to the same message queue object, as long as that name has not been removed. If  
 48931 *name* does not begin with the <slash> character, the effect is implementation-defined. If the *name*  
 48932 argument is not the name of an existing message queue and creation is not requested, *mq\_open()*  
 48933 shall fail and return an error.

48934 A message queue descriptor may be implemented using a file descriptor, in which case  
 48935 applications can open up to at least {OPEN\_MAX} file and message queues.

48936 The *oflag* argument requests the desired receive and/or send access to the message queue. The  
 48937 requested access permission to receive messages or send messages shall be granted if the calling  
 48938 process would be granted read or write access, respectively, to an equivalently protected file.

48939 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications  
 48940 shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

48941 **O\_RDONLY** Open the message queue for receiving messages. The process can use the  
 48942 returned message queue descriptor with *mq\_receive()*, but not *mq\_send()*. A  
 48943 message queue may be open multiple times in the same or different processes  
 48944 for receiving messages.

48945 **O\_WRONLY** Open the queue for sending messages. The process can use the returned  
 48946 message queue descriptor with *mq\_send()* but not *mq\_receive()*. A message  
 48947 queue may be open multiple times in the same or different processes for  
 48948 sending messages.

48949 **O\_RDWR** Open the queue for both receiving and sending messages. The process can use  
 48950 any of the functions allowed for **O\_RDONLY** and **O\_WRONLY**. A message  
 48951 queue may be open multiple times in the same or different processes for  
 48952 sending messages.

48953 Any combination of the remaining flags may be specified in the value of *oflag*:

48954 **O\_CREAT** Create a message queue. It requires two additional arguments: *mode*, which  
 48955 shall be of type **mode\_t**, and *attr*, which shall be a pointer to an **mq\_attr**  
 48956 structure. If the pathname *name* has already been used to create a message  
 48957 queue that still exists, then this flag shall have no effect, except as noted under  
 48958 **O\_EXCL**. Otherwise, a message queue shall be created without any messages

48959 in it. The user ID of the message queue shall be set to the effective user ID of  
 48960 the process. The group ID of the message queue shall be set to the effective  
 48961 group ID of the process; however, if the *name* argument is visible in the file  
 48962 system, the group ID may be set to the group ID of the containing directory.  
 48963 When bits in *mode* other than the file permission bits are specified, the effect is  
 48964 unspecified. If *attr* is NULL, the message queue shall be created with  
 48965 implementation-defined default message queue attributes. If *attr* is non-NULL  
 48966 and the calling process has appropriate privileges on *name*, the message queue  
 48967 *mq\_maxmsg* and *mq\_msgsize* attributes shall be set to the values of the  
 48968 corresponding members in the **mq\_attr** structure referred to by *attr*. The  
 48969 values of the *mq\_flags* and *mq\_curmsgs* members of the **mq\_attr** structure shall  
 48970 be ignored. If *attr* is non-NULL, but the calling process does not have  
 48971 appropriate privileges on *name*, the *mq\_open()* function shall fail and return an  
 48972 error without creating the message queue.

48973 **O\_EXCL** If O\_EXCL and O\_CREAT are set, *mq\_open()* shall fail if the message queue  
 48974 *name* exists. The check for the existence of the message queue and the creation  
 48975 of the message queue if it does not exist shall be atomic with respect to other  
 48976 threads executing *mq\_open()* naming the same *name* with O\_EXCL and  
 48977 O\_CREAT set. If O\_EXCL is set and O\_CREAT is not set, the result is  
 48978 undefined.

48979 **O\_NONBLOCK** Determines whether an *mq\_send()* or *mq\_receive()* waits for resources or  
 48980 messages that are not currently available, or fails with *errno* set to [EAGAIN];  
 48981 see *mq\_send()* and *mq\_receive()* for details.

48982 The *mq\_open()* function does not add or remove messages from the queue.

#### 48983 RETURN VALUE

48984 Upon successful completion, the function shall return a message queue descriptor; otherwise,  
 48985 the function shall return (**mqd\_t**)−1 and set *errno* to indicate the error.

#### 48986 ERRORS

48987 The *mq\_open()* function shall fail if:

48988 [EACCES] The message queue exists and the permissions specified by *oflag* are denied, or  
 48989 the message queue does not exist and permission to create the message queue  
 48990 is denied.

48991 [EEXIST] O\_CREAT and O\_EXCL are set and the named message queue already exists.

48992 [EINTR] The *mq\_open()* function was interrupted by a signal.

48993 [EINVAL] The *mq\_open()* function is not supported for the given name.

48994 [EINVAL] O\_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either  
 48995 *mq\_maxmsg* or *mq\_msgsize* was less than or equal to zero.

48996 [EMFILE] Too many message queue descriptors or file descriptors are currently in use by  
 48997 this process.

48998 [ENFILE] Too many message queue descriptors or file descriptors are currently open in  
 48999 the system.

49000 [ENOENT] O\_CREAT is not set and the named message queue does not exist.

49001 [ENOSPC] There is insufficient space for the creation of the new message queue.



49002 If any of the following conditions occur, the *mq\_open()* function may return (**mqd\_t**)−1 and set  
 49003 *errno* to the corresponding value.

49004 [ENAMETOOLONG]

49005 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems  
 49006 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI  
 49007 systems, or has a pathname component that is longer than  
 49008 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or  
 49009 longer than `{_XOPEN_NAME_MAX}` on XSI systems.

#### 49010 EXAMPLES

49011 None.

#### 49012 APPLICATION USAGE

49013 None.

#### 49014 RATIONALE

49015 None.

#### 49016 FUTURE DIRECTIONS

49017 A future version might require the *mq\_open()* and *mq\_unlink()* functions to have semantics  
 49018 similar to normal file system operations.

#### 49019 SEE ALSO

49020 *mq\_close()*, *mq\_getattr()*, *mq\_receive()*, *mq\_send()*, *mq\_setattr()*, *mq\_unlink()*, *msgctl()*, *msgget()*,  
 49021 *msgrcv()*, *msgsnd()*

49022 XBD <[mqqueue.h](#)>

#### 49023 CHANGE HISTORY

49024 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

#### 49025 Issue 6

49026 The *mq\_open()* function is marked as part of the Message Passing option.

49027 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 49028 implementation does not support the Message Passing option.

49029 The *mq\_timedreceive()* and *mq\_timedsend()* functions are added to the SEE ALSO section for  
 49030 alignment with IEEE Std 1003.1d-1999.

49031 The DESCRIPTION of O\_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

49032 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/62 is applied, updating the description of  
 49033 the permission bits in the DESCRIPTION. The change is made for consistency with the  
 49034 *shm\_open()* and *sem\_open()* functions.

#### 49035 Issue 7

49036 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and  
 49037 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

49038 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

49039 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the  
 49040 user ID and group ID of the message queue.

49041 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0394 [259] is applied.

49042 **Issue 8**  
49043

Austin Group Defect 368 is applied, changing the [ENFILE] error.

49044 **NAME**49045 mq\_receive, mq\_timedreceive — receive a message from a message queue (**REALTIME**)49046 **SYNOPSIS**

```
49047 MSG #include <queue.h>
49048 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
49049 unsigned *msg_prio);
49050 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
49051 size_t msg_len, unsigned *restrict msg_prio,
49052 const struct timespec *restrict abstime);
```

49053 **DESCRIPTION**

49054 The *mq\_receive()* function shall receive the oldest of the highest priority message(s) from the  
 49055 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg\_len*  
 49056 argument, is less than the *mq\_msgsize* attribute of the message queue, the function shall fail and  
 49057 return an error. Otherwise, the selected message shall be removed from the queue and copied to  
 49058 the buffer pointed to by the *msg\_ptr* argument.

49059 If the value of *msg\_len* is greater than {SSIZE\_MAX}, the result is implementation-defined.

49060 If the argument *msg\_prio* is not NULL, the priority of the selected message shall be stored in the  
 49061 location referenced by *msg\_prio*.

49062 If the specified message queue is empty and O\_NONBLOCK is not set in the message queue  
 49063 description associated with *mqdes*, *mq\_receive()* shall block until a message is enqueued on the  
 49064 message queue or until *mq\_receive()* is interrupted by a signal. If more than one thread is waiting  
 49065 to receive a message when a message arrives at an empty queue and the Priority Scheduling  
 49066 option is supported, then the thread of highest priority that has been waiting the longest shall be  
 49067 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the  
 49068 message. If the specified message queue is empty and O\_NONBLOCK is set in the message  
 49069 queue description associated with *mqdes*, no message shall be removed from the queue, and  
 49070 *mq\_receive()* shall return an error.

49071 The *mq\_timedreceive()* function shall receive the oldest of the highest priority messages from the  
 49072 message queue specified by *mqdes* as described for the *mq\_receive()* function. However, if  
 49073 O\_NONBLOCK was not specified when the message queue was opened via the *mq\_open()*  
 49074 function, and no message exists on the queue to satisfy the receive, the wait for such a message  
 49075 shall be terminated when the specified timeout expires. If O\_NONBLOCK is set, this function is  
 49076 equivalent to *mq\_receive()*.

49077 The timeout expires when the absolute time specified by *abstime* passes, as measured by the  
 49078 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 49079 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the  
 49080 call.

49081 The timeout shall be based on the CLOCK\_REALTIME clock. The resolution of the timeout shall  
 49082 be the resolution of the clock on which it is based.

49083 Under no circumstance shall the operation fail with a timeout if a message can be removed from  
 49084 the message queue immediately. The validity of the *abstime* parameter need not be checked if a  
 49085 message can be removed from the message queue immediately.

49086 **RETURN VALUE**

49087 Upon successful completion, the `mq_receive()` and `mq_timedreceive()` functions shall return the  
 49088 length of the selected message in bytes and the message shall be removed from the queue.  
 49089 Otherwise, no message shall be removed from the queue, the functions shall return a value of `-1`,  
 49090 and set `errno` to indicate the error.

49091 **ERRORS**

49092 These functions shall fail if:

49093 [EAGAIN] `O_NONBLOCK` was set in the message description associated with `mqdes`, and  
 49094 the specified message queue is empty.

49095 [EBADF] The `mqdes` argument is not a valid message queue descriptor open for reading.

49096 [EMSGSIZE] The specified message buffer size, `msg_len`, is less than the message size  
 49097 attribute of the message queue.

49098 [EINTR] The `mq_receive()` or `mq_timedreceive()` operation was interrupted by a signal.

49099 [EINVAL] The process or thread would have blocked, and the `abstime` parameter  
 49100 specified a nanoseconds field value less than zero or greater than or equal to  
 49101 1 000 million.

49102 [ETIMEDOUT] The `O_NONBLOCK` flag was not set when the message queue was opened,  
 49103 but no message arrived on the queue before the specified timeout expired.

49104 These functions may fail if:

49105 [EBADMSG] The implementation has detected a data corruption problem with the  
 49106 message.

49107 **EXAMPLES**

49108 None.

49109 **APPLICATION USAGE**

49110 None.

49111 **RATIONALE**

49112 None.

49113 **FUTURE DIRECTIONS**

49114 None.

49115 **SEE ALSO**

49116 [`mq\_open\(\)`](#), [`mq\_send\(\)`](#), [`msgctl\(\)`](#), [`msgget\(\)`](#), [`msgrcv\(\)`](#), [`msgsnd\(\)`](#), [`time\(\)`](#)

49117 XBD [`<mqqueue.h>`](#), [`<time.h>`](#)

49118 **CHANGE HISTORY**

49119 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

49120 **Issue 6**

49121 The `mq_receive()` function is marked as part of the Message Passing option.

49122 The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to  
 49123 `msg_len` rather than `maxsize`.

49124 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 49125 implementation does not support the Message Passing option.

49126 The following new requirements on POSIX implementations derive from alignment with the  
 49127 Single UNIX Specification:

49128 • In this function it is possible for the return value to exceed the range of the type `ssize_t`  
49129 (since `size_t` has a larger range of positive values than `ssize_t`). A sentence restricting the  
49130 size of the `size_t` object is added to the description to resolve this conflict.

49131 The `mq_timedreceive()` function is added for alignment with IEEE Std 1003.1d-1999.

49132 The `restrict` keyword is added to the `mq_timedreceive()` prototype for alignment with the  
49133 ISO/IEC 9899:1999 standard.

49134 IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for `mq_timedreceive()`  
49135 from `int` to `ssize_t`.

49136 **Issue 7**

49137 The `mq_timedreceive()` function is moved from the Timeouts option to the Base.

49138 Functionality relating to the Timers option is moved to the Base.

49139 **Issue 8**

49140 Austin Group Defect 592 is applied, removing text relating to `<time.h>` from the SYNOPSIS and  
49141 DESCRIPTION sections.

49142 **NAME**49143 mq\_send, mq\_timedsend — send a message to a message queue (**REALTIME**)49144 **SYNOPSIS**

```
49145 MSG #include <queue.h>
49146 int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
49147             unsigned msg_prio);
49148 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
49149                 unsigned msg_prio, const struct timespec *abstime);
```

49150 **DESCRIPTION**

49151 The *mq\_send()* function shall add the message pointed to by the argument *msg\_ptr* to the  
 49152 message queue specified by *mqdes*. The *msg\_len* argument specifies the length of the message, in  
 49153 bytes, pointed to by *msg\_ptr*. The value of *msg\_len* shall be less than or equal to the *mq\_msgsize*  
 49154 attribute of the message queue, or *mq\_send()* shall fail.

49155 If the specified message queue is not full, *mq\_send()* shall behave as if the message is inserted  
 49156 into the message queue at the position indicated by the *msg\_prio* argument. A message with a  
 49157 larger numeric value of *msg\_prio* shall be inserted before messages with lower values of  
 49158 *msg\_prio*. A message shall be inserted after other messages in the queue, if any, with equal  
 49159 *msg\_prio*. The value of *msg\_prio* shall be less than {MQ\_PRIO\_MAX}.

49160 If the specified message queue is full and O\_NONBLOCK is not set in the message queue  
 49161 description associated with *mqdes*, *mq\_send()* shall block until space becomes available to  
 49162 enqueue the message, or until *mq\_send()* is interrupted by a signal. If more than one thread is  
 49163 waiting to send when space becomes available in the message queue and the Priority Scheduling  
 49164 option is supported, then the thread of the highest priority that has been waiting the longest  
 49165 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is  
 49166 unblocked. If the specified message queue is full and O\_NONBLOCK is set in the message  
 49167 queue description associated with *mqdes*, the message shall not be queued and *mq\_send()* shall  
 49168 return an error.

49169 The *mq\_timedsend()* function shall add a message to the message queue specified by *mqdes* in the  
 49170 manner defined for the *mq\_send()* function. However, if the specified message queue is full and  
 49171 O\_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for  
 49172 sufficient room in the queue shall be terminated when the specified timeout expires. If  
 49173 O\_NONBLOCK is set in the message queue description, this function shall be equivalent to  
 49174 *mq\_send()*.

49175 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the  
 49176 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 49177 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the  
 49178 call.

49179 The timeout shall be based on the CLOCK\_REALTIME clock. The resolution of the timeout shall  
 49180 be the resolution of the clock on which it is based.

49181 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the  
 49182 queue to add the message immediately. The validity of the *abstime* parameter need not be  
 49183 checked when there is sufficient room in the queue.

49184 **RETURN VALUE**

49185 Upon successful completion, the *mq\_send()* and *mq\_timedsend()* functions shall return a value of  
 49186 zero. Otherwise, no message shall be enqueued, the functions shall return  $-1$ , and *errno* shall be  
 49187 set to indicate the error.

**49188 ERRORS**

49189 The *mq\_send()* and *mq\_timedsend()* functions shall fail if:

- 49190 [EAGAIN] The O\_NONBLOCK flag is set in the message queue description associated  
49191 with *mqdes*, and the specified message queue is full.
- 49192 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for writing.
- 49193 [EINTR] A signal interrupted the call to *mq\_send()* or *mq\_timedsend()*.
- 49194 [EINVAL] The value of *msg\_prio* was outside the valid range.
- 49195 [EINVAL] The process or thread would have blocked, and the *abstime* parameter  
49196 specified a nanoseconds field value less than zero or greater than or equal to  
49197 1 000 million.
- 49198 [EMSGSIZE] The specified message length, *msg\_len*, exceeds the message size attribute of  
49199 the message queue.
- 49200 [ETIMEDOUT] The O\_NONBLOCK flag was not set when the message queue was opened,  
49201 but the timeout expired before the message could be added to the queue.

**49202 EXAMPLES**

49203 None.

**49204 APPLICATION USAGE**

49205 The value of the symbol {MQ\_PRIO\_MAX} limits the number of priority levels supported by the  
49206 application. Message priorities range from 0 to {MQ\_PRIO\_MAX}-1.

**49207 RATIONALE**

49208 None.

**49209 FUTURE DIRECTIONS**

49210 None.

**49211 SEE ALSO**

49212 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_setattr\(\)](#), [time\(\)](#)

49213 XBD [<mqqueue.h>](#), [<time.h>](#)

**49214 CHANGE HISTORY**

49215 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**49216 Issue 6**

49217 The *mq\_send()* function is marked as part of the Message Passing option.

49218 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
49219 implementation does not support the Message Passing option.

49220 The *mq\_timedsend()* function is added for alignment with IEEE Std 1003.1d-1999.

**49221 Issue 7**

49222 The *mq\_timedsend()* function is moved from the Timeouts option to the Base.

49223 Functionality relating to the Timers option is moved to the Base.

**49224 Issue 8**

49225 Austin Group Defect 592 is applied, removing text relating to [<time.h>](#) from the SYNOPSIS and  
49226 DESCRIPTION sections.

49227 **NAME**49228 mq\_setattr — set message queue attributes (**REALTIME**)49229 **SYNOPSIS**

```
49230 MSG #include <mqqueue.h>
49231 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
49232               struct mq_attr *restrict omqstat);
```

49233 **DESCRIPTION**

49234 The *mq\_setattr()* function shall set attributes associated with the open message queue  
 49235 description referenced by the message queue descriptor specified by *mqdes*.

49236 The message queue attributes corresponding to the following members defined in the **mq\_attr**  
 49237 structure shall be set to the specified values upon successful completion of *mq\_setattr()*:

49238 *mq\_flags* The value of this member is the bitwise-logical OR of zero or more of  
 49239 **O\_NONBLOCK** and any implementation-defined flags.

49240 The values of the *mq\_maxmsg*, *mq\_msgsize*, and *mq\_curmsgs* members of the **mq\_attr** structure  
 49241 shall be ignored by *mq\_setattr()*.

49242 If *omqstat* is non-NULL, the *mq\_setattr()* function shall store, in the location referenced by  
 49243 *omqstat*, the previous message queue attributes and the current queue status. These values shall  
 49244 be the same as would be returned by a call to *mq\_getattr()* at that point.

49245 **RETURN VALUE**

49246 Upon successful completion, the function shall return a value of zero and the attributes of the  
 49247 message queue shall have been changed as specified.

49248 Otherwise, the message queue attributes shall be unchanged, and the function shall return a  
 49249 value of  $-1$  and set *errno* to indicate the error.

49250 **ERRORS**

49251 The *mq\_setattr()* function shall fail if:

49252 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

49253 **EXAMPLES**

49254 None.

49255 **APPLICATION USAGE**

49256 None.

49257 **RATIONALE**

49258 None.

49259 **FUTURE DIRECTIONS**

49260 None.

49261 **SEE ALSO**

49262 *mq\_open()*, *mq\_send()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

49263 XBD **<mqqueue.h>**

49264 **CHANGE HISTORY**

49265 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.



49266 **Issue 6**

49267 The *mq\_setattr()* function is marked as part of the Message Passing option.

49268 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
49269 implementation does not support the Message Passing option.

49270 The *mq\_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std  
49271 1003.1d-1999.

49272 The **restrict** keyword is added to the *mq\_setattr()* prototype for alignment with the  
49273 ISO/IEC 9899:1999 standard.

49274 **NAME**

49275 mq\_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)

49276 **SYNOPSIS**

```
49277 MSG #include <mqueue.h>
49278      ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
49279                          size_t msg_len, unsigned *restrict msg_prio,
49280                          const struct timespec *restrict abstime);
```

49281 **DESCRIPTION**

49282 Refer to [mq\\_receive\(\)](#).

49283 **NAME**49284 mq\_timedsend — send a message to a message queue (**ADVANCED REALTIME**)49285 **SYNOPSIS**

```
49286 MSG #include <mqueue.h>
49287 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
49288 unsigned msg_prio, const struct timespec *abstime);
```

49289 **DESCRIPTION**49290 Refer to [mq\\_send\(\)](#).

49291 **NAME**49292 mq\_unlink — remove a message queue (**REALTIME**)49293 **SYNOPSIS**

```
49294 MSG #include <mqueue.h>
49295 int mq_unlink(const char *name);
```

49296 **DESCRIPTION**

49297 The *mq\_unlink()* function shall remove the message queue named by the string *name*. If one or  
 49298 more processes have the message queue open when *mq\_unlink()* is called, destruction of the  
 49299 message queue shall be postponed until all references to the message queue have been closed.  
 49300 However, the *mq\_unlink()* call need not block until all references have been closed; it may return  
 49301 immediately.

49302 After a successful call to *mq\_unlink()*, reuse of the name shall subsequently cause *mq\_open()* to  
 49303 behave as if no message queue of this name exists (that is, *mq\_open()* shall fail if **O\_CREAT** is not  
 49304 set, or shall create a new message queue if **O\_CREAT** is set).

49305 **RETURN VALUE**

49306 Upon successful completion, the function shall return a value of zero. Otherwise, the named  
 49307 message queue shall be unchanged by this function call, and the function shall return a value of  
 49308  $-1$  and set *errno* to indicate the error.

49309 **ERRORS**49310 The *mq\_unlink()* function shall fail if:

49311 [EACCES] Permission is denied to unlink the named message queue.

49312 [EINTR] The call to *mq\_unlink()* blocked waiting for all references to the named  
49313 message queue to be closed and a signal interrupted the call.

49314 [ENOENT] The named message queue does not exist.

49315 The *mq\_unlink()* function may fail if:

49316 [ENAMETOOLONG]

49317 The length of the *name* argument exceeds  $\{\_POSIX\_PATH\_MAX\}$  on systems  
 49318 XSI that do not support the XSI option or exceeds  $\{\_XOPEN\_PATH\_MAX\}$  on XSI  
 49319 systems, or has a pathname component that is longer than  
 49320 XSI  $\{\_POSIX\_NAME\_MAX\}$  on systems that do not support the XSI option or  
 49321 longer than  $\{\_XOPEN\_NAME\_MAX\}$  on XSI systems. A call to *mq\_unlink()*  
 49322 with a *name* argument that contains the same message queue name as was  
 49323 previously used in a successful *mq\_open()* call shall not give an  
 49324 [ENAMETOOLONG] error.

49325 **EXAMPLES**

49326 None.

49327 **APPLICATION USAGE**

49328 None.

49329 **RATIONALE**

49330 None.

**49331 FUTURE DIRECTIONS**

49332 A future version might require the *mq\_open()* and *mq\_unlink()* functions to have semantics  
49333 similar to normal file system operations.

**49334 SEE ALSO**

49335 *mq\_close()*, *mq\_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

49336 XBD <[mqqueue.h](#)>

**49337 CHANGE HISTORY**

49338 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**49339 Issue 6**

49340 The *mq\_unlink()* function is marked as part of the Message Passing option.

49341 The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion,  
49342 the named message queue is unchanged by this function.

49343 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
49344 implementation does not support the Message Passing option.

**49345 Issue 7**

49346 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a  
49347 ``shall fail'' to a ``may fail'' error .

49348 Austin Group Interpretation 1003.1-2001 #141 is applied.

49349 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0230 [504] is applied.

49350 **NAME**

49351 mrnd48 — generate uniformly distributed pseudo-random signed long integers

49352 **SYNOPSIS**

49353 XSI `#include <stdlib.h>`

49354 `long mrnd48(void);`

49355 **DESCRIPTION**

49356 Refer to *drand48()*.

49357 **NAME**

49358 msgctl — XSI message control operations

49359 **SYNOPSIS**

```
49360 XSI      #include <sys/msg.h>
49361      int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

49362 **DESCRIPTION**

49363 The *msgctl()* function operates on XSI message queues (see XBD [Section 3.206](#), on page 61). It is  
 49364 unspecified whether this function interoperates with the realtime interprocess communication  
 49365 facilities defined in [Section 2.8](#) (on page 527).

49366 The *msgctl()* function shall provide message control operations as specified by *cmd*. The  
 49367 following values for *cmd*, and the message control operations they specify, are:

49368 **IPC\_STAT** Place the current value of each member of the **msqid\_ds** data structure  
 49369 associated with *msqid* into the structure pointed to by *buf*. The contents of this  
 49370 structure are defined in **<sys/msg.h>**.

49371 **IPC\_SET** Set the value of the following members of the **msqid\_ds** data structure  
 49372 associated with *msqid* to the corresponding value found in the structure  
 49373 pointed to by *buf*:

```
49374      msg_perm.uid
49375      msg_perm.gid
49376      msg_perm.mode
49377      msg_qbytes
```

49378 Also, the *msg\_ctime* timestamp shall be set to the current time, as described in  
 49379 [Section 2.7.1](#) (on page 526).

49380 **IPC\_SET** can only be executed by a process with appropriate privileges or that  
 49381 has an effective user ID equal to the value of **msg\_perm.cuid** or  
 49382 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*. Only a  
 49383 process with appropriate privileges can raise the value of **msg\_qbytes**.

49384 **IPC\_RMID** Remove the message queue identifier specified by *msqid* from the system and  
 49385 destroy the message queue and **msqid\_ds** data structure associated with it.  
 49386 **IPC\_RMID** can only be executed by a process with appropriate privileges or  
 49387 one that has an effective user ID equal to the value of **msg\_perm.cuid** or  
 49388 **msg\_perm.uid** in the **msqid\_ds** data structure associated with *msqid*.

49389 **RETURN VALUE**

49390 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 49391 indicate the error.

49392 **ERRORS**

49393 The *msgctl()* function shall fail if:

49394 **[EACCES]** The argument *cmd* is **IPC\_STAT** and the calling process does not have read  
 49395 permission; see [Section 2.7](#) (on page 526).

49396 **[EINVAL]** The value of *msqid* is not a valid message queue identifier; or the value of *cmd*  
 49397 is not a valid command.

49398 **[EPERM]** The argument *cmd* is **IPC\_RMID** or **IPC\_SET** and the effective user ID of the  
 49399 calling process is not equal to that of a process with appropriate privileges and  
 49400 it is not equal to the value of **msg\_perm.cuid** or **msg\_perm.uid** in the data

49401 structure associated with *msqid*.

49402 [EPERM] The argument *cmd* is `IPC_SET`, an attempt is being made to increase to the  
49403 value of `msg_qbytes`, and the effective user ID of the calling process does not  
49404 have appropriate privileges.

49405 **EXAMPLES**

49406 None.

49407 **APPLICATION USAGE**

49408 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
49409 (IPC). Application developers who need to use IPC should design their applications so that  
49410 modules using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to  
49411 use the alternative interfaces.

49412 **RATIONALE**

49413 None.

49414 **FUTURE DIRECTIONS**

49415 None.

49416 **SEE ALSO**

49417 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
49418 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

49419 XBD [Section 3.206](#) (on page 61), [<sys/msg.h>](#)

49420 **CHANGE HISTORY**

49421 First released in Issue 2. Derived from Issue 2 of the SVID.

49422 **Issue 5**

49423 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
49424 DIRECTIONS to a new APPLICATION USAGE section.

49425 **Issue 7**

49426 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0395 [345] is applied.



49427 **NAME**

49428 msgget — get the XSI message queue identifier

49429 **SYNOPSIS**

```
49430 XSI #include <sys/msg.h>
49431 int msgget(key_t key, int msgflg);
```

49432 **DESCRIPTION**

49433 The `msgget()` function operates on XSI message queues (see XBD Section 3.206, on page 61). It is  
 49434 unspecified whether this function interoperates with the realtime interprocess communication  
 49435 facilities defined in Section 2.8 (on page 527).

49436 The `msgget()` function shall return the message queue identifier associated with the argument  
 49437 `key`.

49438 A message queue identifier, associated message queue, and data structure (see `<sys/msg.h>`),  
 49439 shall be created for the argument `key` if one of the following is true:

- 49440 • The argument `key` is equal to `IPC_PRIVATE`.
- 49441 • The argument `key` does not already have a message queue identifier associated with it, and  
 49442 (`msgflg & IPC_CREAT`) is non-zero.

49443 Upon creation, the data structure associated with the new message queue identifier shall be  
 49444 initialized as follows:

- 49445 • `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` shall be set to the  
 49446 effective user ID and effective group ID, respectively, of the calling process.
- 49447 • The low-order 9 bits of `msg_perm.mode` shall be set to the low-order 9 bits of `msgflg`.
- 49448 • `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` shall be set to 0.
- 49449 • `msg_ctime` shall be set to the current time, as described in Section 2.7.1 (on page 526).
- 49450 • `msg_qbytes` shall be set to the system limit.

49451 **RETURN VALUE**

49452 Upon successful completion, `msgget()` shall return a non-negative integer, namely a message  
 49453 queue identifier. Otherwise, it shall return `-1` and set `errno` to indicate the error.

49454 **ERRORS**

49455 The `msgget()` function shall fail if:

- |                         |          |   |
|-------------------------|----------|---|
| 49456<br>49457<br>49458 | [EACCES] | A message queue identifier exists for the argument <code>key</code> , but operation permission as specified by the low-order 9 bits of <code>msgflg</code> would not be granted; see Section 2.7 (on page 526). |
| 49459<br>49460          | [EEXIST] | A message queue identifier exists for the argument <code>key</code> but <code>((msgflg &amp; IPC_CREAT) &amp;&amp; (msgflg &amp; IPC_EXCL))</code> is non-zero.   |
| 49461<br>49462          | [ENOENT] | A message queue identifier does not exist for the argument <code>key</code> and <code>(msgflg &amp; IPC_CREAT)</code> is 0.   |
| 49463<br>49464<br>49465 | [ENOSPC] | A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.  |

49466 **EXAMPLES**

49467 None.

49468 **APPLICATION USAGE**

49469 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
49470 (IPC). Application developers who need to use IPC should design their applications so that  
49471 modules using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to  
49472 use the alternative interfaces.

49473 **RATIONALE**

49474 None.

49475 **FUTURE DIRECTIONS**

49476 None.

49477 **SEE ALSO**

49478 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), [ftok\(\)](#), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
49479 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgctl\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

49480 [XBD Section 3.206](#) (on page 61), [<sys/msg.h>](#)49481 **CHANGE HISTORY**

49482 First released in Issue 2. Derived from Issue 2 of the SVID.

49483 **Issue 5**

49484 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
49485 DIRECTIONS to a new APPLICATION USAGE section.

49486 **Issue 7**

49487 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0396 [345] and XSH/TC1-2008/0397  
49488 [344] are applied.

49489 **NAME**

49490 msgrcv — XSI message receive operation

49491 **SYNOPSIS**

```
49492 XSI      #include <sys/msg.h>
49493
49493      ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
49494                  int msgflg);
```

49495 **DESCRIPTION**

49496 The `msgrcv()` function operates on XSI message queues (see XBD [Section 3.206](#), on page 61). It is  
 49497 unspecified whether this function interoperates with the realtime interprocess communication  
 49498 facilities defined in [Section 2.8](#) (on page 527).

49499 The `msgrcv()` function shall read a message from the queue associated with the message queue  
 49500 identifier specified by `msqid` and place it in the user-defined buffer pointed to by `msgp`.

49501 The application shall ensure that the argument `msgp` points to a user-defined buffer that contains  
 49502 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 49503 the data bytes of the message. The structure below is an example of what this user-defined  
 49504 buffer might look like:

```
49505 struct mymsg {
49506     long    mtype;      /* Message type. */
49507     char    mtext[1];  /* Message text. */
49508 }
```

49509 The structure member `mtype` is the received message's type as specified by the sending process.

49510 The structure member `mtext` is the text of the message.

49511 The argument `msgsz` specifies the size in bytes of `mtext`. The received message shall be truncated  
 49512 to `msgsz` bytes if it is larger than `msgsz` and `(msgflg & MSG_NOERROR)` is non-zero. The  
 49513 truncated part of the message shall be lost and no indication of the truncation shall be given to  
 49514 the calling process.

49515 If the value of `msgsz` is greater than `{SSIZE_MAX}`, the result is implementation-defined.

49516 The argument `msgtyp` specifies the type of message requested as follows:

- 49517 • If `msgtyp` is 0, the first message on the queue shall be received.
- 49518 • If `msgtyp` is greater than 0, the first message of type `msgtyp` shall be received.
- 49519 • If `msgtyp` is less than 0, the first message of the lowest type that is less than or equal to the  
 49520 absolute value of `msgtyp` shall be received.

49521 The argument `msgflg` specifies the action to be taken if a message of the desired type is not on the  
 49522 queue. These are as follows:

- 49523 • If `(msgflg & IPC_NOWAIT)` is non-zero, the calling thread shall return immediately with a  
 49524 return value of `-1` and `errno` set to `[ENOMSG]`.
- 49525 • If `(msgflg & IPC_NOWAIT)` is 0, the calling thread shall suspend execution until one of the  
 49526 following occurs:
  - 49527 — A message of the desired type is placed on the queue.
  - 49528 — The message queue identifier `msqid` is removed from the system; when this occurs,  
 49529 `errno` shall be set to `[EIDRM]` and `-1` shall be returned.

49530 — The calling thread receives a signal that is to be caught; in this case a message is not  
 49531 received and the calling thread resumes execution in the manner prescribed in  
 49532 *sigaction()*.

49533 Upon successful completion, the following actions are taken with respect to the data structure  
 49534 associated with *msqid*:

- 49535 • **msg\_qnum** shall be decremented by 1.
- 49536 • **msg\_lrp** shall be set to the process ID of the calling process.
- 49537 • **msg\_rtime** shall be set to the current time, as described in [Section 2.7.1](#) (on page 526).

#### 49538 RETURN VALUE

49539 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually  
 49540 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return  $-1$ ,  
 49541 and *errno* shall be set to indicate the error.

#### 49542 ERRORS

49543 The *msgrcv()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 49544 | [E2BIG]  | The value of <i>mtext</i> is greater than <i>msgsz</i> and ( <i>msgflg</i> & MSG_NOERROR) is 0.         |
| 49545 | [EACCES] | Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 526).   |
| 49546 |          |   |
| 49547 | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.                                   |
| 49548 | [EINTR]  | The <i>msgrcv()</i> function was interrupted by a signal.   |
| 49549 | [EINVAL] | <i>msqid</i> is not a valid message queue identifier.   |
| 49550 | [ENOMSG] | The queue does not contain a message of the desired type and ( <i>msgflg</i> & IPC_NOWAIT) is non-zero. |
| 49551 |          |   |

#### 49552 EXAMPLES

##### 49553 Receiving a Message

49554 The following example receives the first message on the queue (based on the value of the *msgtyp*  
 49555 argument, 0). The queue is identified by the *msqid* argument (assuming that the value has  
 49556 previously been set). This call specifies that an error should be reported if no message is  
 49557 available, but not if the message is too large. The message size is calculated directly using the  
 49558 *sizeof* operator.

```

49559 #include <sys/msg.h>
49560 ...
49561 int result;
49562 int msqid;
49563 struct message {
49564     long type;
49565     char text[20];
49566 } msg;
49567 long msgtyp = 0;
49568 ...
49569 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
49570                msgtyp, MSG_NOERROR | IPC_NOWAIT);
  
```

**49571 APPLICATION USAGE**

49572 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
49573 (IPC). Application developers who need to use IPC should design their applications so that  
49574 modules using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to  
49575 use the alternative interfaces.

**49576 RATIONALE**

49577 None.

**49578 FUTURE DIRECTIONS**

49579 None.

**49580 SEE ALSO**

49581 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
49582 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgsnd\(\)](#),  
49583 [sigaction\(\)](#)

49584 XBD [Section 3.206](#) (on page 61), [<sys/msg.h>](#)

**49585 CHANGE HISTORY**

49586 First released in Issue 2. Derived from Issue 2 of the SVID.

**49587 Issue 5**

49588 The type of the return value is changed from **int** to **ssize\_t**, and a warning is added to the  
49589 DESCRIPTION about values of *msgsz* larger than {SSIZE\_MAX}.

49590 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
49591 DIRECTIONS to the APPLICATION USAGE section.

**49592 Issue 6**

49593 The normative text is updated to avoid use of the term “must” for application requirements.

**49594 Issue 7**

49595 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0398 [345] and XSH/TC1-2008/0399  
49596 [421] are applied.

49597 **NAME**

49598 msgsnd — XSI message send operation

49599 **SYNOPSIS**

```
49600 XSI #include <sys/msg.h>
49601 int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

49602 **DESCRIPTION**

49603 The `msgsnd()` function operates on XSI message queues (see XBD [Section 3.206](#), on page 61). It is  
 49604 unspecified whether this function interoperates with the realtime interprocess communication  
 49605 facilities defined in [Section 2.8](#) (on page 527).

49606 The `msgsnd()` function shall send a message to the queue associated with the message queue  
 49607 identifier specified by `msqid`.

49608 The application shall ensure that the argument `msgp` points to a user-defined buffer that contains  
 49609 first a field of type **long** specifying the type of the message, and then a data portion that holds  
 49610 the data bytes of the message. The structure below is an example of what this user-defined  
 49611 buffer might look like:

```
49612 struct mymsg {
49613     long    mtype;        /* Message type. */
49614     char    mtext[1];    /* Message text. */
49615 }
```

49616 The structure member `mtype` is a non-zero positive type **long** that can be used by the receiving  
 49617 process for message selection.

49618 The structure member `mtext` is any text of length `msgsz` bytes. The argument `msgsz` can range  
 49619 from 0 to a system-imposed maximum.

49620 The argument `msgflg` specifies the action to be taken if one or more of the following is true:

- 49621 • The number of bytes already on the queue is equal to **msg\_qbytes**; see `<sys/msg.h>`.
- 49622 • The total number of messages on all queues system-wide is equal to the system-imposed  
 49623 limit.

49624 These actions are as follows:

- 49625 • If `(msgflg & IPC_NOWAIT)` is non-zero, the message shall not be sent and the calling  
 49626 thread shall return immediately.
- 49627 • If `(msgflg & IPC_NOWAIT)` is 0, the calling thread shall suspend execution until one of the  
 49628 following occurs:
  - 49629 — The condition responsible for the suspension no longer exists, in which case the  
 49630 message is sent.
  - 49631 — The message queue identifier `msqid` is removed from the system; when this occurs,  
 49632 `errno` shall be set to [EIDRM] and `-1` shall be returned.
  - 49633 — The calling thread receives a signal that is to be caught; in this case the message is not  
 49634 sent and the calling thread resumes execution in the manner prescribed in [sigaction\(\)](#).

49635 Upon successful completion, the following actions are taken with respect to the data structure  
 49636 associated with *msqid*; see `<sys/msg.h>`:

- 49637 • **msg\_qnum** shall be incremented by 1.
- 49638 • **msg\_lspid** shall be set to the process ID of the calling process.
- 49639 • **msg\_stime** shall be set to the current time, as described in [Section 2.7.1](#) (on page 526).

#### 49640 RETURN VALUE

49641 Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,  
 49642 *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

#### 49643 ERRORS

49644 The *msgsnd()* function shall fail if:

- |                         |          |   |
|-------------------------|----------|---|
| 49645<br>49646          | [EACCES] | Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 526).   |
| 49647<br>49648          | [EAGAIN] | The message cannot be sent for one of the reasons cited above and ( <i>msgflg</i> & <code>IPC_NOWAIT</code> ) is non-zero.  |
| 49649                   | [EIDRM]  | The message queue identifier <i>msqid</i> is removed from the system.   |
| 49650                   | [EINTR]  | The <i>msgsnd()</i> function was interrupted by a signal.   |
| 49651<br>49652<br>49653 | [EINVAL] | The value of <i>msqid</i> is not a valid message queue identifier, or the value of <i>mtyp</i> e is less than 1; or the value of <i>msgsz</i> is greater than the system-imposed limit. |

#### 49654 EXAMPLES

##### 49655 Sending a Message

49656 The following example sends a message to the queue identified by the *msqid* argument  
 49657 (assuming that value has previously been set). This call specifies that an error should be  
 49658 reported if no message is available. The message size is calculated directly using the *sizeof*  
 49659 operator.

```

49660 #include <sys/msg.h>
49661 ...
49662 int result;
49663 int msqid;
49664 struct message {
49665     long type;
49666     char text[20];
49667 } msg;
49668 msg.type = 1;
49669 strcpy(msg.text, "This is message 1");
49670 ...
49671 result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
  
```

#### 49672 APPLICATION USAGE

49673 The POSIX Realtime Extension defines alternative interfaces for interprocess communication  
 49674 (IPC). Application developers who need to use IPC should design their applications so that  
 49675 modules using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to  
 49676 use the alternative interfaces.

49677 **RATIONALE**

49678 None.

49679 **FUTURE DIRECTIONS**

49680 None.

49681 **SEE ALSO**49682 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), [mq\\_close\(\)](#), [mq\\_getattr\(\)](#), [mq\\_notify\(\)](#),  
49683 [mq\\_open\(\)](#), [mq\\_receive\(\)](#), [mq\\_send\(\)](#), [mq\\_setattr\(\)](#), [mq\\_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#),  
49684 [sigaction\(\)](#)49685 [XBD Section 3.206](#) (on page 61), [<sys/msg.h>](#)49686 **CHANGE HISTORY**

49687 First released in Issue 2. Derived from Issue 2 of the SVID.

49688 **Issue 5**49689 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
49690 DIRECTIONS to a new APPLICATION USAGE section.49691 **Issue 6**

49692 The normative text is updated to avoid use of the term “must” for application requirements.

49693 **Issue 7**49694 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0400 [345] and XSH/TC1-2008/0401  
49695 [359] are applied.



49696 **NAME**

49697       msync — synchronize memory with physical storage

49698 **SYNOPSIS**

```
49699 XSI|SIO #include <sys/mman.h>
49700 int msync(void *addr, size_t len, int flags);
```

49701 **DESCRIPTION**

49702 The *msync()* function shall write all modified data to permanent storage locations, if any, in  
 49703 those whole pages containing any part of the address space of the process starting at address  
 49704 *addr* and continuing for *len* bytes. If no such storage exists, *msync()* need not have any effect. If  
 49705 requested, the *msync()* function shall then invalidate cached copies of data.

49706 The implementation may require that *addr* be a multiple of the page size as returned by  
 49707 *sysconf()*.

49708 For mappings to files, the *msync()* function shall ensure that all write operations are completed  
 49709 as defined for synchronized I/O data integrity completion. It is unspecified whether the  
 49710 implementation also writes out other file attributes. When the *msync()* function is called on  
 49711 MAP\_PRIVATE mappings, any modified data shall not be written to the underlying object and  
 49712 shall not cause such data to be made visible to other processes. It is unspecified whether data in  
 49713 MAP\_PRIVATE mappings has any permanent storage locations. The effect of *msync()* on an  
 49714 SHM anonymous memory object, shared memory object, or  
 49715 TYM typed memory object is unspecified. The behavior of this function is unspecified if the mapping  
 49716 was not established by a call to *mmap()*.

49717 The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following  
 49718 flags defined in the `<sys/mman.h>` header:

49719	Symbolic Constant	Description
49720	MS_ASYNC	Perform asynchronous writes.
49721	MS_SYNC	Perform synchronous writes.
49722	MS_INVALIDATE	Invalidate cached data.

49723 When MS\_ASYNC is specified, *msync()* shall return immediately once all the write operations  
 49724 are initiated or queued for servicing; when MS\_SYNC is specified, *msync()* shall not return until  
 49725 all write operations are completed as defined for synchronized I/O data integrity completion.  
 49726 Either MS\_ASYNC or MS\_SYNC shall be specified, but not both.

49727 When MS\_INVALIDATE is specified, *msync()* shall invalidate all cached copies of mapped data  
 49728 that are inconsistent with the permanent storage locations such that subsequent references shall  
 49729 obtain data that was consistent with the permanent storage locations sometime between the call  
 49730 to *msync()* and the first subsequent memory reference to the data.

49731 If *msync()* causes any write to a file, the file's last data modification and last file status change  
 49732 timestamps shall be marked for update.

49733 **RETURN VALUE**

49734 Upon successful completion, *msync()* shall return 0; otherwise, it shall return `-1` and set *errno* to  
 49735 indicate the error.

49736 **ERRORS**

49737 The *msync()* function shall fail if:

- 49738 [EBUSY] Some or all of the addresses in the range starting at *addr* and continuing for *len*  
49739 bytes are locked, and MS\_INVALIDATE is specified.
- 49740 [EINVAL] The value of *flags* is invalid.
- 49741 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are  
49742 outside the range allowed for the address space of a process or specify one or  
49743 more pages that are not mapped.
- 49744 The *msync()* function may fail if:
- 49745 [EINVAL] The value of *addr* is not a multiple of the page size as returned by *sysconf()*.

**EXAMPLES**

49746 None.  
49747

**APPLICATION USAGE**

- 49748 The *msync()* function is only supported if the Synchronized Input and Output option is  
49749 supported, and thus need not be available on all implementations.  
49750
- 49751 The *msync()* function should be used by programs that require a memory object to be in a  
49752 known state; for example, in building transaction facilities.
- 49753 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees  
49754 that *msync()* is the only control over when pages are or are not written to disk.

**RATIONALE**

- 49755 The *msync()* function writes out data in a mapped region to the permanent storage for the  
49756 underlying object. The call to *msync()* ensures data integrity of the file.  
49757
- 49758 After the data is written out, any cached data may be invalidated if the MS\_INVALIDATE flag  
49759 was specified. This is useful on systems that do not support read/write consistency.

**FUTURE DIRECTIONS**

49760 None.  
49761

**SEE ALSO**

- 49762 [\*mmap\(\)\*](#), [\*sysconf\(\)\*](#)  
49763  
49764 XBD [\*\*<sys/mman.h>\*\*](#)

**CHANGE HISTORY**

49765 First released in Issue 4, Version 2.  
49766

**Issue 5**

- 49767 Moved from X/OPEN UNIX extension to BASE.  
49768
- 49769 Aligned with *msync()* in the POSIX Realtime Extension as follows:
- 49770 • The DESCRIPTION is extensively reworded.
  - 49771 • [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

**Issue 6**

- 49772 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input  
49773 and Output options.  
49774
- 49775 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 49776 • The [EBUSY] mandatory error condition is added.
- 49777 The following new requirements on POSIX implementations derive from alignment with the  
49778 Single UNIX Specification:

49779 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple  
49780 of the page size.

49781 • The second [EINVAL] error condition is made mandatory.

49782 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to  
49783 typed memory objects.

49784 **Issue 7**

49785 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment  
49786 requirements.

49787 SD5-XSH-ERN-110 is applied.

49788 The *msync()* function is marked as part of the Synchronized Input and Output option or XSI  
49789 option as the Memory Mapped Files is moved to the Base.

49790 Changes are made related to support for finegrained timestamps.

49791 **Issue 8**

49792 Austin Group Defect 850 is applied, adding anonymous memory objects.

49793 **NAME**49794 `mtx_destroy`, `mtx_init` — destroy and initialize a mutex49795 **SYNOPSIS**

```
49796 #include <threads.h>
49797 void mtx_destroy(mtx_t *mtx);
49798 int  mtx_init(mtx_t *mtx, int type);
```

49799 **DESCRIPTION**

49800 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 49801 conflict between the requirements described here and the ISO C standard is unintentional. This  
 49802 volume of POSIX.1-2024 defers to the ISO C standard.

49803 The `mtx_destroy()` function shall release any resources used by the mutex pointed to by `mtx`. A  
 49804 destroyed mutex object can be reinitialized using `mtx_init()`; the results of otherwise referencing  
 49805 the object after it has been destroyed are undefined. It shall be safe to destroy an initialized  
 49806 mutex that is unlocked. Attempting to destroy a locked mutex, or a mutex that another thread is  
 49807 attempting to lock, or a mutex that is being used in a `cond_timedwait()` or `cond_wait()` call by  
 49808 another thread, results in undefined behavior. The behavior is undefined if the value specified  
 49809 by the `mtx` argument to `mtx_destroy()` does not refer to an initialized mutex.

49810 The `mtx_init()` function shall initialize a mutex object with properties indicated by `type`, whose  
 49811 valid values include:

49812	<code>mtx_plain</code>	for a simple non-recursive mutex,
49813	<code>mtx_timed</code>	for a non-recursive mutex that supports timeout,
49814	<code>mtx_plain</code>   <code>mtx_recursive</code>	for a simple recursive mutex, or
49815	<code>mtx_timed</code>   <code>mtx_recursive</code>	for a recursive mutex that supports timeout.

49816 If the `mtx_init()` function succeeds, it shall set the mutex pointed to by `mtx` to a value that  
 49817 uniquely identifies the newly initialized mutex. Upon successful initialization, the state of the  
 49818 mutex shall become initialized and unlocked. Attempting to initialize an already initialized  
 49819 mutex results in undefined behavior.

49820 CX See [Section 2.9.9](#) (on page 548) for further requirements.

49821 These functions shall not be affected if the calling thread executes a signal handler during the  
 49822 call.

49823 **RETURN VALUE**

49824 The `mtx_destroy()` function shall not return a value.

49825 The `mtx_init()` function shall return `thrd_success` on success or `thrd_error` if the request  
 49826 could not be honored.

49827 **ERRORS**

49828 No errors are defined.

49829 **EXAMPLES**

49830 None.

49831 **APPLICATION USAGE**

49832 A mutex can be destroyed immediately after it is unlocked. However, since attempting to  
49833 destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is  
49834 being used in a *cond\_timedwait()* or *cond\_wait()* call by another thread results in undefined  
49835 behavior, care must be taken to ensure that no other thread may be referencing the mutex.

49836 **RATIONALE**

49837 These functions are not affected by signal handlers for the reasons stated in XRAT [Section B.2.3](#)  
49838 (on page 3742).

49839 **FUTURE DIRECTIONS**

49840 None.

49841 **SEE ALSO**49842 [cond\\_timedwait\(\)](#), [mtx\\_lock\(\)](#)49843 XBD [<threads.h>](#)49844 **CHANGE HISTORY**

49845 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

49846 **NAME**

49847       mtx\_lock, mtx\_timedlock, mtx\_trylock, mtx\_unlock — lock and unlock a mutex

49848 **SYNOPSIS**

```
49849       #include <threads.h>
49850       int mtx_lock(mtx_t *mtx);
49851       int mtx_timedlock(mtx_t *restrict mtx,
49852                        const struct timespec *restrict ts);
49853       int mtx_trylock(mtx_t *mtx);
49854       int mtx_unlock(mtx_t *mtx);
```

49855 **DESCRIPTION**

49856 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
49857 conflict between the requirements described here and the ISO C standard is unintentional. This  
49858 volume of POSIX.1-2024 defers to the ISO C standard.

49859       The *mtx\_lock()* function shall block until it locks the mutex pointed to by *mtx*. If the mutex is  
49860 non-recursive, the application shall ensure that it is not already locked by the calling thread.

49861       The *mtx\_timedlock()* function shall block until it locks the mutex pointed to by *mtx* or until after  
49862 the TIME\_UTC-based calendar time pointed to by *ts*. The application shall ensure that the  
49863 CX specified mutex supports timeout. Under no circumstance shall the function fail with a timeout  
49864 if the mutex can be locked immediately. The validity of the *ts* parameter need not be checked if  
49865 the mutex can be locked immediately.

49866       The *mtx\_trylock()* function shall endeavor to lock the mutex pointed to by *mtx*. If the mutex is  
49867 already locked (by any thread, including the current thread), the function shall return without  
49868 blocking. If the mutex is recursive and the mutex is currently owned by the calling thread, the  
49869 mutex lock count (see below) shall be incremented by one and the *mtx\_trylock()* function shall  
49870 immediately return success.

49871 CX       These functions shall not be affected if the calling thread executes a signal handler during the  
49872 call; if a signal is delivered to a thread waiting for a mutex, upon return from the signal handler  
49873 the thread shall resume waiting for the mutex as if it was not interrupted.

49874       If a call to *mtx\_lock()*, *mtx\_timedlock()* or *mtx\_trylock()* locks the mutex, prior calls to  
49875 *mtx\_unlock()* on the same mutex shall synchronize with this lock operation.

49876       The *mtx\_unlock()* function shall unlock the mutex pointed to by *mtx*. The application shall  
49877 CX ensure that the mutex pointed to by *mtx* is locked by the calling thread. If there are threads  
49878 blocked on the mutex object referenced by *mtx* when *mtx\_unlock()* is called, resulting in the  
49879 mutex becoming available, the scheduling policy shall determine which thread shall acquire the  
49880 mutex.

49881       A recursive mutex shall maintain the concept of a lock count. When a thread successfully  
49882 acquires a mutex for the first time, the lock count shall be set to one. Every time a thread relocks  
49883 this mutex, the lock count shall be incremented by one. Each time the thread unlocks the mutex,  
49884 the lock count shall be decremented by one. When the lock count reaches zero, the mutex shall  
49885 become available for other threads to acquire.

49886       For purposes of determining the existence of a data race, mutex lock and unlock operations on  
49887 mutexes of type **mtx\_t** behave as atomic operations. All lock and unlock operations on a  
49888 particular mutex occur in some particular total order.

49889       If *mtx* does not refer to an initialized mutex object, the behavior of these functions is undefined.

**49890 RETURN VALUE**

49891 The *mtx\_lock()* and *mtx\_unlock()* functions shall return `thrd_success` on success, or  
49892 `thrd_error` if the request could not be honored.

49893 The *mtx\_timedlock()* function shall return `thrd_success` on success, or `thrd_timedout` if the  
49894 time specified was reached without acquiring the requested resource, or `thrd_error` if the  
49895 request could not be honored.

49896 The *mtx\_trylock()* function shall return `thrd_success` on success, or `thrd_busy` if the  
49897 resource requested is already in use, or `thrd_error` if the request could not be honored. The  
49898 *mtx\_trylock()* function can spuriously fail to lock an unused resource, in which case it shall  
49899 return `thrd_busy`.

**49900 ERRORS**

49901 See RETURN VALUE.

**49902 EXAMPLES**

49903 None.

**49904 APPLICATION USAGE**

49905 None.

**49906 RATIONALE**

49907 These functions are not affected by signal handlers for the reasons stated in XRAT [Section B.2.3](#)  
49908 (on page 3742).

49909 Since `<pthread.h>` has no equivalent of the `mtx_timed` mutex property, if the `<threads.h>`  
49910 interfaces are implemented as a thin wrapper around `<pthread.h>` interfaces (meaning `mtx_t`  
49911 and `pthread_mutex_t` are the same type), all mutexes support timeout and *mtx\_timedlock()* will  
49912 not fail for a mutex that was not initialized with `mtx_timed`. Alternatively, implementations  
49913 can use a less thin wrapper where `mtx_t` contains additional properties that are not held in  
49914 `pthread_mutex_t` in order to be able to return a failure indication from *mtx\_timedlock()* calls  
49915 where the mutex was not initialized with `mtx_timed`.

**49916 FUTURE DIRECTIONS**

49917 None.

**49918 SEE ALSO**

49919 [\*mtx\\_destroy\(\)\*](#), [\*timespec\\_get\(\)\*](#)

49920 XBD [`<threads.h>`](#)

**49921 CHANGE HISTORY**

49922 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

49923 **NAME**

49924           munlock — unlock a range of process address space

49925 **SYNOPSIS**

```
49926 MLR       #include <sys/mman.h>  
49927           int munlock(const void *addr, size_t len);
```

49928 **DESCRIPTION**

49929           Refer to *mlock()*.



49930 **NAME**

49931           munlockall — unlock the address space of a process

49932 **SYNOPSIS**

49933 ML        #include &lt;sys/mman.h&gt;

49934           int munlockall(void);

49935 **DESCRIPTION**49936           Refer to *mlockall()*.

49937 **NAME**

49938           munmap — unmap pages of memory

49939 **SYNOPSIS**

49940           #include &lt;sys/mman.h&gt;

49941           int munmap(void \*addr, size\_t len);

49942 **DESCRIPTION**

49943           The *munmap()* function shall remove any mappings for those entire pages containing any part of  
 49944           the address space of the process starting at *addr* and continuing for *len* bytes. Further references  
 49945           to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no  
 49946           mappings in the specified address range, then *munmap()* has no effect.

49947           The implementation may require that *addr* be a multiple of the page size as returned by  
 49948           *sysconf()*.

49949           If a mapping to be removed was private, any modifications made in this address range shall be  
 49950           discarded.

49951   ML|MLR   Any memory locks (see *mlock()* and *mlockall()*) associated with this address range shall be  
 49952           removed, as if by an appropriate call to *munlock()*.

49953   TYM       If a mapping removed from a typed memory object causes the corresponding address range of  
 49954           the memory pool to be inaccessible by any process in the system except through allocatable  
 49955           mappings (that is, mappings of typed memory objects opened with the  
 49956           POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag), then that range of the memory pool shall  
 49957           become deallocated and may become available to satisfy future typed memory allocation  
 49958           requests.

49959           A mapping removed from a typed memory object opened with the  
 49960           POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag shall not affect in any way the availability of  
 49961           that typed memory for allocation.

49962           The behavior of this function is unspecified if the mapping was not established by a call to  
 49963           *mmap()*.

49964 **RETURN VALUE**

49965           Upon successful completion, *munmap()* shall return 0; otherwise, it shall return -1 and set *errno*  
 49966           to indicate the error.

49967 **ERRORS**

49968           The *munmap()* function shall fail if:

49969           [EINVAL]       Addresses in the range [*addr*,*addr*+*len*) are outside the valid range for the  
 49970                           address space of a process.

49971           [EINVAL]       The *len* argument is 0.

49972           The *munmap()* function may fail if:

49973           [EINVAL]       The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

49974 **EXAMPLES**

49975 None.

49976 **APPLICATION USAGE**

49977 None.

49978 **RATIONALE**49979 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

49980 It is possible that an application has applied process memory locking to a region that contains  
 49981 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,  
 49982 causes those locks to be removed.

49983 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.49984 **FUTURE DIRECTIONS**

49985 None.

49986 **SEE ALSO**49987 [mlock\(\)](#), [mlockall\(\)](#), [mmap\(\)](#), [posix\\_typed\\_mem\\_open\(\)](#), [sysconf\(\)](#)49988 XBD [<sys/mman.h>](#)49989 **CHANGE HISTORY**

49990 First released in Issue 4, Version 2.

49991 **Issue 5**

49992 Moved from X/OPEN UNIX extension to BASE.

49993 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- 49994 • The DESCRIPTION is extensively reworded.
- 49995 • The SIGBUS error is no longer permitted to be generated.

49996 **Issue 6**

49997 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory  
 49998 Objects option.

49999 The following new requirements on POSIX implementations derive from alignment with the  
 50000 Single UNIX Specification:

- 50001 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple  
 50002 of the page size.
- 50003 • The [EINVAL] error conditions are added.

50004 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 50005 • Semantics for typed memory objects are added to the DESCRIPTION.
- 50006 • The *posix\_typed\_mem\_open()* function is added to the SEE ALSO section.

50007 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code  
 50008 in the SYNOPSIS from MF | SHM to MC3 (notation for MF | SHM | TYM).

50009 **Issue 7**

50010 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment  
 50011 requirements.

50012 The *munmap()* function is moved from the Memory Mapped Files option to the Base.

50013 **NAME**

50014 nan, nanf, nanl — return quiet NaN

50015 **SYNOPSIS**

```
50016 #include <math.h>
50017 double nan(const char *tagp);
50018 float nanf(const char *tagp);
50019 long double nanl(const char *tagp);
```

50020 **DESCRIPTION**

50021 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 50022 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50023 volume of POSIX.1-2024 defers to the ISO C standard.

50024 The function call *nan("n-char-sequence")* shall be equivalent to:

```
50025 strtod("NAN(n-char-sequence)", (char **) NULL);
```

50026 The function call *nan("")* shall be equivalent to:

```
50027 strtod("NAN()", (char **) NULL)
```

50028 If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be  
 50029 equivalent to:

```
50030 strtod("NAN", (char **) NULL)
```

50031 Function calls to *nanf()* and *nanl()* are equivalent to the corresponding function calls to *strtof()*  
 50032 and *strtold()*.

50033 **RETURN VALUE**

50034 These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

50035 MX The returned value shall be exact and shall be independent of the current rounding direction  
 50036 mode.

50037 If the implementation does not support quiet NaNs, these functions shall return zero.

50038 **ERRORS**

50039 No errors are defined.

50040 **EXAMPLES**

50041 None.

50042 **APPLICATION USAGE**

50043 None.

50044 **RATIONALE**

50045 None.

50046 **FUTURE DIRECTIONS**

50047 None.

50048 **SEE ALSO**

50049 [strtod\(\)](#)

50050 XBD [<math.h>](#)

50051 **CHANGE HISTORY**

50052 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

50053 **Issue 8**

50054 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
50055 standard.

50056 **NAME**

50057 nanosleep — high resolution sleep

50058 **SYNOPSIS**

```
50059 CX #include <time.h>
50060 int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

50061 **DESCRIPTION**

50062 The *nanosleep()* function shall cause the current thread to be suspended from execution until  
 50063 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the  
 50064 calling thread, and its action is to invoke a signal-catching function or to terminate the process.  
 50065 The suspension time may be longer than requested because the argument value is rounded up to  
 50066 an integer multiple of the sleep resolution or because of the scheduling of other activity by the  
 50067 system. But, except for the case of being interrupted by a signal, the suspension time shall not be  
 50068 less than the time specified by *rqtp*, as measured by the system clock `CLOCK_REALTIME`.

50069 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.

50070 **RETURN VALUE**

50071 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall  
 50072 be zero.

50073 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a  
 50074 value of `-1` and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the  
 50075 **timespec** structure referenced by it is updated to contain the amount of time remaining in the  
 50076 interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments can  
 50077 point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned.

50078 If *nanosleep()* fails, it shall return a value of `-1` and set *errno* to indicate the error.

50079 **ERRORS**

50080 The *nanosleep()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 50081 | [EINTR]  | The <i>nanosleep()</i> function was interrupted by a signal.  |
| 50082 | [EINVAL] | The <i>rqtp</i> argument specified a nanosecond value less than zero or greater than or equal to 1 000 million. |
| 50083 |          |   |

50084 **EXAMPLES**

50085 None.

50086 **APPLICATION USAGE**

50087 None.

50088 **RATIONALE**

50089 It is common to suspend execution of a thread for an interval in order to poll the status of a non-  
 50090 interrupting function. A large number of actual needs can be met with a simple extension to  
 50091 *sleep()* that provides finer resolution.

50092 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the  
 50093 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,  
 50094 it is possible to write such a routine using no static storage and reserving no system facilities.  
 50095 Although it is possible to write a function with similar functionality to *sleep()* using the  
 50096 remainder of the *timer\_\**(*)* functions, such a function requires the use of signals and the  
 50097 reservation of some signal number. This volume of POSIX.1-2024 requires that *nanosleep()* be  
 50098 non-intrusive of the signals function.

50099 The *nanosleep()* function shall return a value of 0 on success and `-1` on failure or if interrupted.

50100 This latter case is different from *sleep()*. This was done because the remaining time is returned  
50101 via an argument structure pointer, *rmtpt*, instead of as the return value.

50102 **FUTURE DIRECTIONS**

50103 None.

50104 **SEE ALSO**

50105 *clock\_nanosleep()*, *sleep()*

50106 XBD <**time.h**>

50107 **CHANGE HISTORY**

50108 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

50109 **Issue 6**

50110 The *nanosleep()* function is marked as part of the Timers option.

50111 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
50112 implementation does not support the Timers option.

50113 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO  
50114 section to include the *clock\_nanosleep()* function.

50115 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/63 is applied, correcting text in the  
50116 RATIONALE section.

50117 **Issue 7**

50118 SD5-XBD-ERN-33 is applied.

50119 The *nanosleep()* function is moved from the Timers option to the Base.

50120 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0231 [909] is applied.

50121 **NAME**

50122 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

50123 **SYNOPSIS**

```
50124 #include <math.h>
50125 double nearbyint(double x);
50126 float nearbyintf(float x);
50127 long double nearbyintl(long double x);
```

50128 **DESCRIPTION**

50129 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 50130 conflict between the requirements described here and the ISO C standard is unintentional. This  
 50131 volume of POSIX.1-2024 defers to the ISO C standard.

50132 These functions shall round their argument to an integer value in floating-point format, using  
 50133 the current rounding direction and without raising the inexact floating-point exception.

50134 **RETURN VALUE**

50135 MX Upon successful completion, these functions shall return the rounded integer value. The result  
 50136 shall have the same sign as  $x$ .

50137 MX If  $x$  is NaN, a NaN shall be returned.

50138 If  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.

50139 If  $x$  is  $\pm \text{Inf}$ ,  $x$  shall be returned.

50140 **ERRORS**

50141 No errors are defined.

50142 **EXAMPLES**

50143 None.

50144 **APPLICATION USAGE**

50145 The integral value returned by these functions need not be expressible as an `intmax_t`. The  
 50146 return value should be tested before assigning it to an integer type to avoid the undefined  
 50147 results of an integer overflow.

50148 **RATIONALE**

50149 None.

50150 **FUTURE DIRECTIONS**

50151 None.

50152 **SEE ALSO**

50153 [feclearexcept\(\)](#), [fetestexcept\(\)](#)

50154 XBD [Section 4.23](#) (on page 109), [<math.h>](#)

50155 **CHANGE HISTORY**

50156 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

50157 **Issue 7**

50158 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0402 [346,428] is applied.



50159 **NAME**

50160 newlocale — create or modify a locale object

50161 **SYNOPSIS**

```
50162 CX #include <locale.h>
50163 locale_t newlocale(int category_mask, const char *locale,
50164 locale_t base);
```

50165 **DESCRIPTION**

50166 The *newlocale()* function shall create a new locale object or modify an existing one. If the *base*  
 50167 argument is **(locale\_t)0**, a new locale object shall be created, otherwise the locale specified by  
 50168 *base* shall be modified. In the latter case it is unspecified whether the resulting locale object shall  
 50169 be that pointed to by *base* modified in place, or whether that object shall be freed after a new  
 50170 locale object is first created using some values from it.

50171 The *category\_mask* argument specifies the locale categories to be set or modified. Values for  
 50172 *category\_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants  
 50173 *LC\_CTYPE\_MASK*, *LC\_NUMERIC\_MASK*, *LC\_TIME\_MASK*, *LC\_COLLATE\_MASK*,  
 50174 *LC\_MONETARY\_MASK*, and *LC\_MESSAGES\_MASK*, or any of the implementation-defined  
 50175 mask values defined in **<locale.h>**.

50176 For each category with the corresponding bit set in *category\_mask* the data from the locale named  
 50177 by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale  
 50178 named by *locale* shall replace the existing data within the locale object. If a completely new locale  
 50179 object is created, the data for all sections not requested by *category\_mask* shall be taken from the  
 50180 POSIX locale.

50181 The following preset values of *locale* are defined for all settings of *category\_mask*:

50182 "POSIX" Specifies the minimal environment for C-language translation called the  
 50183 POSIX locale.

50184 "C" Equivalent to "POSIX".

50185 "" Specifies an implementation-defined native environment. This corresponds to  
 50186 the value of the associated environment variables, *LC\_\** and *LANG*; see XBD  
 50187 [Chapter 7](#) (on page 127) and [Chapter 8](#) (on page 167).

50188 If the *base* argument is not **(locale\_t)0** and the *newlocale()* function call succeeds, the contents of  
 50189 *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before  
 50190 calling *newlocale()*. If the function call fails and the *base* argument is not **(locale\_t)0**, the contents  
 50191 of *base* shall remain valid and unchanged.

50192 The behavior is undefined if the *base* argument is the special locale object  
 50193 *LC\_GLOBAL\_LOCALE*, or is not a valid locale object handle and is not **(locale\_t)0**.

50194 **RETURN VALUE**

50195 Upon successful completion, the *newlocale()* function shall return a handle which the caller may  
 50196 use on subsequent calls to *duplocale()*, *freelocale()*, and other functions taking a **locale\_t**  
 50197 argument.

50198 Upon failure, the *newlocale()* function shall return **(locale\_t)0** and set *errno* to indicate the error.

50199 **ERRORS**

50200 The *newlocale()* function shall fail if:

50201 [ENOMEM] There is not enough memory available to create the locale object or load the  
 50202 locale data.

50203 [EINVAL] The *category\_mask* contains a bit that does not correspond to a valid category.

50204 [ENOENT] For any of the categories in *category\_mask*, the locale data is not available.

50205 The *newlocale()* function may fail if:

50206 [EINVAL] The *locale* argument is not a valid string pointer.

## 50207 EXAMPLES

### 50208 Constructing a Locale Object from Different Locales

50209 The following example shows the construction of a locale where the *LC\_CTYPE* category data  
 50210 comes from a locale *loc1* and the *LC\_TIME* category data from a locale *loc2*:

```
50211 #include <locale.h>
50212 ...
50213 locale_t loc, new_loc;
50214 /* Get the "loc1" data. */
50215 loc = newlocale (LC_CTYPE_MASK, "loc1", (locale_t)0);
50216 if (loc == (locale_t) 0)
50217     abort ();
50218 /* Get the "loc2" data. */
50219 new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
50220 if (new_loc != (locale_t) 0)
50221     /* We don't abort if this fails. In this case this
50222      * simply used to unchanged locale object. */
50223     loc = new_loc;
50224 ...
```

### 50225 Freeing up a Locale Object

50226 The following example shows a code fragment to free a locale object created by *newlocale()*:

```
50227 #include <locale.h>
50228 ...
50229 /* Every locale object allocated with newlocale() should be
50230  * freed using freelocale():
50231  */
50232 locale_t loc;
50233 /* Get the locale. */
50234 loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", (locale_t)0);
50235 /* ... Use the locale object ... */
50236 ...
50237 /* Free the locale object resources. */
50238 freelocale (loc);
```

50239 **APPLICATION USAGE**

50240 Handles for locale objects created by the *newlocale()* function should either be released by a  
50241 corresponding call to *freelocale()*, or be used as a base locale to another *newlocale()* call.

50242 The special locale object LC\_GLOBAL\_LOCALE must not be passed for the *base* argument, even  
50243 when returned by the *uselocale()* function.

50244 **RATIONALE**

50245 None.

50246 **FUTURE DIRECTIONS**

50247 None.

50248 **SEE ALSO**

50249 *duplocale()*, *freelocale()*, *getlocalename\_1()*, *uselocale()*

50250 XBD Chapter 7 (on page 127), Chapter 8 (on page 167), <locale.h>

50251 **CHANGE HISTORY**

50252 First released in Issue 7.

50253 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0403 [227], XSH/TC1-2008/0404 [283],  
50254 XSH/TC1-2008/0405 [295], and XSH/TC1-2008/0406 [227] are applied.

50255 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0232 [781] and XSH/TC2-2008/0233  
50256 [673] are applied.

50257 **Issue 8**

50258 Austin Group Defect 1220 is applied, adding *getlocalename\_1()* to the SEE ALSO section.

50259 Austin Group Defect 1243 is applied, clarifying the handling of a non-zero *base* argument.

50260 Austin Group Defect 1264 is applied, changing “default locale” to “POSIX locale”.

50261 **NAME**

50262 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable  
50263 floating-point number

50264 **SYNOPSIS**

```
50265 #include <math.h>
50266 double nextafter(double x, double y);
50267 float nextafterf(float x, float y);
50268 long double nextafterl(long double x, long double y);
50269 double nexttoward(double x, long double y);
50270 float nexttowardf(float x, long double y);
50271 long double nexttowardl(long double x, long double y);
```

50272 **DESCRIPTION**

50273 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
50274 conflict between the requirements described here and the ISO C standard is unintentional. This  
50275 volume of POSIX.1-2024 defers to the ISO C standard.

50276 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable  
50277 floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter()* shall  
50278 return the largest representable floating-point number less than *x*. The *nextafter()*, *nextafterf()*,  
50279 and *nextafterl()* functions shall return *y* if *x* equals *y*.

50280 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions shall be equivalent to the  
50281 corresponding *nextafter()* functions, except that the second parameter shall have type **long**  
50282 **double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

50283 An application wishing to check for error situations should set *errno* to zero and call  
50284 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
50285 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
50286 zero, an error has occurred.

50287 **RETURN VALUE**

50288 Upon successful completion, these functions shall return the next representable floating-point  
50289 value following *x* in the direction of *y*.

50290 If  $x=y$ , *y* (of the type *x*) shall be returned.

50291 MX Even though underflow or overflow can occur, the returned value shall be independent of the  
50292 current rounding direction mode.

50293 If *x* is finite and the correct function value would overflow, a range error shall occur and  
50294  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (with the same sign as *x*) shall be returned as  
50295 appropriate for the return type of the function.

50296 MX If *x* or *y* is NaN, a NaN shall be returned.

50297 MX If  $x \neq y$  and the correct function value is subnormal, zero, or underflows, a range error shall  
50298 occur, and

50299 MXX the correct function value (if representable) or

50300 MX 0.0 shall be returned.

50301 **ERRORS**

50302 These functions shall fail if:

50303 Range Error The correct value overflows.

50304 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
50305 then *errno* shall be set to [ERANGE]. If the integer expression

50306 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 50307 floating-point exception shall be raised.

50308 MX **Range Error** The correct value is subnormal or underflows.

50309 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 50310 then *errno* shall be set to [ERANGE]. If the integer expression  
 50311 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 50312 floating-point exception shall be raised.

#### 50313 **EXAMPLES**

50314 None.

#### 50315 **APPLICATION USAGE**

50316 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 50317 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

50318 When **<tgmath.h>** is included, note that the return type of *nextafter()* depends on the generic  
 50319 typing deduced from both arguments, while the return type of *nexttoward()* depends only on the  
 50320 generic typing of the first argument.

#### 50321 **RATIONALE**

50322 None.

#### 50323 **FUTURE DIRECTIONS**

50324 None.

#### 50325 **SEE ALSO**

50326 [\*feclearexcept\(\)\*](#), [\*fetestexcept\(\)\*](#)

50327 XBD Section 4.23 (on page 109), **<math.h>**, **<tgmath.h>**

#### 50328 **CHANGE HISTORY**

50329 First released in Issue 4, Version 2.

#### 50330 **Issue 5**

50331 Moved from X/OPEN UNIX extension to BASE.

#### 50332 **Issue 6**

50333 The *nextafter()* function is no longer marked as an extension.

50334 The *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions are added  
 50335 for alignment with the ISO/IEC 9899:1999 standard.

50336 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 50337 revised to align with the ISO/IEC 9899:1999 standard.

50338 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
 50339 marked.

#### 50340 **Issue 7**

50341 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0407 [68] and XSH/TC1-2008/0408  
 50342 [357] are applied.

#### 50343 **Issue 8**

50344 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
 50345 standard.

50346 **NAME**

50347 nftw — walk a file tree

50348 **SYNOPSIS**

```
50349 XSI #include <ftw.h>
50350 int nftw(const char *path, int (*fn)(const char *,
50351 const struct stat *, int, struct FTW *), int fd_limit, int flags);
```

50352 **DESCRIPTION**

50353 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The  
 50354 argument *flags* is a bitwise-inclusive OR of zero or more of the following flags:

50355 **FTW\_CHDIR** If set, *nftw()* shall change the current working directory to each directory as it  
 50356 reports files in that directory. If clear, *nftw()* shall not change the current  
 50357 working directory.

50358 **FTW\_DEPTH** If set, *nftw()* shall report all files in a directory before reporting the directory  
 50359 itself. If clear, *nftw()* shall report any directory before reporting the files in that  
 50360 directory.

50361 **FTW\_MOUNT** If set, *nftw()* shall only report files that have the same device ID (*st\_dev*) as  
 50362 *path* and shall not descend below directories that have a different device ID  
 50363 than *path*. If clear, *nftw()* shall report all files encountered during the walk,  
 50364 unless **FTW\_XDEV** is set.

50365 **FTW\_PHYS** If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

50366 **FTW\_XDEV** If set, *nftw()* shall not descend below directories that have a different device  
 50367 ID (*st\_dev*) than *path*; that is, when a directory with a different device ID is  
 50368 encountered, *nftw()* shall report the directory itself (unless **FTW\_MOUNT** is  
 50369 set) but shall not report any files below the directory. If clear, *nftw()* shall  
 50370 report all files encountered during the walk, unless **FTW\_MOUNT** is set.

50371 **Note:** If both **FTW\_MOUNT** and **FTW\_XDEV** are set, *nftw()* obeys both flags but the end result is the  
 50372 same as if **FTW\_XDEV** were clear.

50373 If **FTW\_PHYS** is clear and **FTW\_DEPTH** is set, *nftw()* shall follow links instead of reporting  
 50374 them, but shall not report any directory that would be a descendant of itself. If **FTW\_PHYS** is  
 50375 clear and **FTW\_DEPTH** is clear, *nftw()* shall follow links instead of reporting them, but shall not  
 50376 report the contents of any directory that would be a descendant of itself.

50377 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- 50378 • The first argument is the pathname of the object.
- 50379 • The second argument is a pointer to the **stat** buffer containing information on the object,  
 50380 filled in as if *fstatat()*, *stat()*, or *lstat()* had been called to retrieve the information.
- 50381 • The third argument is an integer giving additional information. Its value is one of the  
 50382 following:
  - 50383 **FTW\_D** The object is a directory.
  - 50384 **FTW\_DNR** The object is a directory that cannot be read. The *fn* function shall not be  
 50385 called for any of its descendants.
  - 50386 **FTW\_DP** The object is a directory and subdirectories have been visited. (This condition  
 50387 shall only occur if the **FTW\_DEPTH** flag is included in *flags*.)

- 50388 FTW\_F The object is a non-directory file.
- 50389 FTW\_NS The *stat()* function failed on the object because of lack of appropriate  
50390 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any  
50391 other reason is considered an error and *nftw()* shall return  $-1$ .
- 50392 FTW\_SL The object is a symbolic link. (This condition shall only occur if the  
50393 FTW\_PHYS flag is included in *flags*.)
- 50394 FTW\_SLN The object is a symbolic link that does not name an existing file. The **stat**  
50395 buffer passed to *fn* shall contain information on the symbolic link. (This  
50396 condition shall only occur if the FTW\_PHYS flag is not included in *flags*.)
- 50397 • The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the
  - 50398 object's filename in the pathname passed as the first argument to *fn*. The value of **level**
  - 50399 indicates depth relative to the root of the walk, where the root level is 0.

50400 The results are unspecified if the application-supplied *fn* function does not preserve the current  
50401 working directory.

50402 The argument *fd\_limit* sets the maximum number of file descriptors that shall be used by *nftw()*  
50403 while traversing the file tree. At most one file descriptor shall be used for each directory level.  
50404 The FD\_CLOEXEC flag shall be set on any file descriptor opened by *nftw()* (see <fcntl.h>) not  
50405 including those opened by the user-supplied *fn* function. Every file descriptor opened by *nftw()*  
50406 not including those opened by the user-supplied *fn* function shall be closed before *nftw()*  
50407 returns.

50408 The *nftw()* function need not be thread-safe.

#### 50409 RETURN VALUE

50410 The *nftw()* function shall continue until the first of the following conditions occurs:

- 50411 • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that  
50412 value.
- 50413 • The *nftw()* function detects an error other than [EACCES] (see FTW\_DNR and FTW\_NS  
50414 above), in which case *nftw()* shall return  $-1$  and set *errno* to indicate the error.
- 50415 • The tree is exhausted, in which case *nftw()* shall return 0.

#### 50416 ERRORS

50417 The *nftw()* function shall fail if:

- 50418 [EACCES] Search permission is denied for any component of *path* or read permission is  
50419 denied for *path*, or *fn* returns  $-1$  and does not reset *errno*.
- 50420 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
50421 argument.
- 50422 [ENAMETOOLONG]  
50423 The length of a component of a pathname is longer than {NAME\_MAX}.
- 50424 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 50425 [ENOTDIR] A component of *path* names an existing file that is neither a directory nor a  
50426 symbolic link to a directory.
- 50427 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current  
50428 programming environment for one or more files found in the file hierarchy.

50429 The *nftw()* function may fail if:

50430 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
50431 resolution of the *path* argument.

50432 [EMFILE] All file descriptors available to the process are currently open.

50433 [ENAMETOOLONG]  
50434 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
50435 symbolic link produced an intermediate result with a length that exceeds  
50436 {PATH\_MAX}.

50437 [ENFILE] Too many files are currently open in the system.

50438 In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

**EXAMPLES**

50439 The following program traverses the directory tree under the path named in its first command-  
50440 line argument, or under the current directory if no argument is supplied. It displays various  
50441 information about each file. The second command-line argument can be used to specify  
50442 characters that control the value assigned to the *flags* argument when calling *nftw()*.  
50443

```
50444 #include <ftw.h>
50445 #include <stdio.h>
50446 #include <stdlib.h>
50447 #include <string.h>
50448 #include <stdint.h>

50449 static int
50450 display_info(const char *fpath, const struct stat *sb,
50451             int tflag, struct FTW *ftwbuf)
50452 {
50453     printf("%-3s %2d %7jd   %-40s %d %s\n",
50454           (tflag == FTW_D) ? "d" : (tflag == FTW_DNR) ? "dnr" :
50455           (tflag == FTW_DP) ? "dp" : (tflag == FTW_F) ?
50456           (S_ISBLK(sb->st_mode) ? "f b" :
50457           S_ISCHR(sb->st_mode) ? "f c" :
50458           S_ISFIFO(sb->st_mode) ? "f p" :
50459           S_ISREG(sb->st_mode) ? "f r" :
50460           S_ISSOCK(sb->st_mode) ? "f s" : "f ?") :
50461           (tflag == FTW_NS) ? "ns" : (tflag == FTW_SL) ? "sl" :
50462           (tflag == FTW_SLN) ? "sln" : "?", ftwbuf->level,
50463           (intmax_t) ((tflag == FTW_NS) ? -1 : sb->st_size),
50464           fpath, ftwbuf->base, fpath + ftwbuf->base);
50465     return 0; /* To tell nftw() to continue */
50466 }

50467 int
50468 main(int argc, char *argv[])
50469 {
50470     int flags = 0;

50471     if (argc > 2 && strchr(argv[2], 'd') != NULL)
50472         flags |= FTW_DEPTH;
50473     if (argc > 2 && strchr(argv[2], 'p') != NULL)
50474         flags |= FTW_PHYS;

50475     if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1)
50476     {
```



```

50477         perror("nftw");
50478         exit(EXIT_FAILURE);
50479     }
50480     exit(EXIT_SUCCESS);
50481 }

```

#### 50482 APPLICATION USAGE

50483 The *nftw()* function may allocate dynamic storage during its operation. If *nftw()* is forcibly  
50484 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*  
50485 or an interrupt routine, *nftw()* does not have a chance to free that storage, so it remains  
50486 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has  
50487 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next  
50488 invocation.

50489 When restricting the walk to files on one file system, it can sometimes be desirable for the  
50490 crossing points themselves to be reported and sometimes for them not to be reported. (Crossing  
50491 points are mount points and, if FTW\_PHYS is clear, symbolic links to directories on other file  
50492 systems.) With FTW\_XDEV *nftw()* reports them and with FTW\_MOUNT it does not. However,  
50493 with FTW\_MOUNT it also does not report symbolic links to non-directory files on other file  
50494 systems (if FTW\_PHYS is clear). If there is a need for an application to exclude crossing points  
50495 but include symbolic links to non-directory files on other file systems, this can be achieved by  
50496 using FTW\_XDEV and performing a check such as the following in the function pointed to by *fn*:

```

50497     if (tflag == FTW_D && sb->st_dev != saved_dev)
50498         return 0;

```

50499 (where *saved\_dev* is the *st\_dev* value for *path*).

#### 50500 RATIONALE

50501 Earlier versions of this standard did not make clear that, as well as not reporting them,  
50502 FTW\_MOUNT prevents descent below directories that have a different device ID than *path* if  
50503 they are encountered by following a symbolic link (rather than by being a mount point). This  
50504 meant that if such a directory contained any symbolic links to files with the same device ID as  
50505 *path*, *nftw()* with FTW\_PHYS clear was required to report them. However, this was not how  
50506 *nftw()* implementations behaved and the standard has been amended to match existing practice.

#### 50507 FUTURE DIRECTIONS

50508 None.

#### 50509 SEE ALSO

50510 [\*fdopendir\(\)\*](#), [\*fstatat\(\)\*](#), [\*readdir\(\)\*](#)

50511 XBD [\*<fcntl.h>\*](#), [\*<ftw.h>\*](#)

#### 50512 CHANGE HISTORY

50513 First released in Issue 4, Version 2.

#### 50514 Issue 5

50515 Moved from X/OPEN UNIX extension to BASE.

50516 In the DESCRIPTION, the definition of the *depth* argument is clarified.

#### 50517 Issue 6

50518 The Open Group Base Resolution bwg97-003 is applied.

50519 The ERRORS section is updated as follows:

- 50520
  - The wording of the mandatory [ELOOP] error condition is updated.
- 50521
  - A second optional [ELOOP] error condition is added.
- 50522
  - The [Eoverflow] mandatory error condition is added.
- 50523

Text is added to the DESCRIPTION to say that the *nftw()* function need not be reentrant and that
- 50524

the results are unspecified if the application-supplied *fn* function does not preserve the current
- 50525

working directory.
- 50526

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/64 is applied, changing the argument
- 50527

*depth* to *fd\_limit* throughout and changing “to a maximum of 5 levels deep” to “using a
- 50528

maximum of 5 file descriptors” in the EXAMPLES section.
- 50529 **Issue 7**
- 50530

Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.
- 50531

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 50532

SD5-XBD-ERN-61 is applied.
- 50533

APPLICATION USAGE is added and the EXAMPLES section is replaced with a new example.
- 50534

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0409 [403], XSH/TC1-2008/0410 [324],
- 50535

and XSH/TC1-2008/0411 [403] are applied.
- 50536 **Issue 8**
- 50537

Austin Group Defect 368 is applied, adding a requirement for the FD\_CLOEXEC flag to be set.
- 50538

Austin Group Defect 1121 is applied, changing the description of FTW\_SLN and the handling of
- 50539

FTW\_NS in the EXAMPLES section.
- 50540

Austin Group Defect 1133 is applied, adding FTW\_XDEV.
- 50541

Austin Group Defect 1210 is applied, changing the description of FTW\_MOUNT and the
- 50542

RATIONALE section.
- 50543

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

50544 **NAME**

50545       ngettext, ngettext\_l — message handling functions

50546 **SYNOPSIS**

50547       #include &lt;libintl.h&gt;

50548       char \*ngettext(const char \*msgid, const char \*msgid\_plural,  
50549                    unsigned long int n);50550       char \*ngettext\_l(const char \*msgid, const char \*msgid\_plural,  
50551                    unsigned long int n, locale\_t locale);50552 **DESCRIPTION**50553       Refer to *gettext*.

50554 **NAME**

50555 nice — change the nice value of a process

50556 **SYNOPSIS**

```
50557 XSI      #include <unistd.h>
50558      int nice(int incr);
```

50559 **DESCRIPTION**

50560 The *nice()* function shall add the value of *incr* to the nice value of the calling process. A nice  
 50561 value of a process is a non-negative number for which a more positive value shall result in less  
 50562 favorable scheduling.

50563 A maximum nice value of  $2^{\{NZERO\}}-1$  and a minimum nice value of 0 shall be imposed by the  
 50564 system. Requests for values above or below these limits shall result in the nice value being set to  
 50565 the corresponding limit. Only a process with appropriate privileges can lower the nice value.

50566 PS|TPS Calling the *nice()* function has no effect on the priority of processes or threads with policy  
 50567 SCHED\_FIFO or SCHED\_RR. The effect on processes or threads with other scheduling policies  
 50568 is implementation-defined.

50569 The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the  
 50570 nice value shall affect all system scope threads in the process.

50571 As  $-1$  is a permissible return value in a successful situation, an application wishing to check for  
 50572 error situations should set *errno* to 0, then call *nice()*, and if it returns  $-1$ , check to see whether  
 50573 *errno* is non-zero.

50574 **RETURN VALUE**

50575 Upon successful completion, *nice()* shall return the new nice value  $-\{NZERO\}$ . Otherwise,  $-1$   
 50576 shall be returned, the nice value of the process shall not be changed, and *errno* shall be set to  
 50577 indicate the error.

50578 **ERRORS**

50579 The *nice()* function shall fail if:

50580 [EPERM] The *incr* argument is negative and the calling process does not have  
 50581 appropriate privileges.

50582 **EXAMPLES**50583 **Changing the Nice Value**

50584 The following example adds the value of the *incr* argument,  $-20$ , to the nice value of the calling  
 50585 process.

```
50586 #include <unistd.h>
50587 ...
50588 int incr = -20;
50589 int ret;

50590 ret = nice(incr);
```

50591 **APPLICATION USAGE**

50592 None.

50593 **RATIONALE**

50594 None.

50595 **FUTURE DIRECTIONS**

50596 None.

50597 **SEE ALSO**50598 *exec*, *getpriority()*

50599 XBD &lt;limits.h&gt;, &lt;unistd.h&gt;

50600 **CHANGE HISTORY**

50601 First released in Issue 1. Derived from Issue 1 of the SVID.

50602 **Issue 5**50603 A statement is added to the description indicating the effects of this function on the different  
50604 scheduling policies and multi-threaded processes.

50605 **NAME**

50606 nl\_langinfo, nl\_langinfo\_l — language information

50607 **SYNOPSIS**

50608 #include &lt;langinfo.h&gt;

50609 char \*nl\_langinfo(nl\_item item);

50610 char \*nl\_langinfo\_l(nl\_item item, locale\_t locale);

50611 **DESCRIPTION**

50612 The *nl\_langinfo()* and *nl\_langinfo\_l()* functions shall return a pointer to a string containing  
50613 information relevant to the particular language or cultural area defined in the current locale, or  
50614 in the locale represented by *locale*, respectively (see <langinfo.h>). The manifest constant names  
50615 and values of *item* are defined in <langinfo.h>. For example:

50616 nl\_langinfo(ABDAY\_1)

50617 would return a pointer to the string "Dom" if the identified language was Portuguese, and  
50618 "Sun" if the identified language was English.

50619 nl\_langinfo\_l(ABDAY\_1, loc)

50620 would return a pointer to the string "Dom" if the identified language of the locale represented by  
50621 *loc* was Portuguese, and "Sun" if the identified language of the locale represented by *loc* was  
50622 English.

50623 The *nl\_langinfo()* function need not be thread-safe.

50624 The behavior is undefined if the *locale* argument to *nl\_langinfo\_l()* is the special locale object  
50625 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

50626 **RETURN VALUE**

50627 In a locale where *langinfo* data is not defined, these functions shall return a pointer to the  
50628 corresponding string in the POSIX locale. In all locales, these functions shall return a pointer to  
50629 an empty string if *item* contains an invalid setting.

50630 The application shall not modify the string returned. The pointer returned by *nl\_langinfo()*  
50631 might be invalidated or the string content might be overwritten by a subsequent call to  
50632 *nl\_langinfo()* in any thread or to *nl\_langinfo\_l()* in the same thread or the initial thread, by  
50633 subsequent calls to *setlocale()* with a category corresponding to the category of *item* (see  
50634 <langinfo.h>) or the category LC\_ALL, or by subsequent calls to *uselocale()* which change the  
50635 category corresponding to the category of *item*. The pointer returned by *nl\_langinfo\_l()* might be  
50636 invalidated or the string content might be overwritten by a subsequent call to *nl\_langinfo\_l()* in  
50637 the same thread or to *nl\_langinfo()* in any thread, or by subsequent calls to *freelocale()* or  
50638 *newlocale()* which free or modify the locale object that was passed to *nl\_langinfo\_l()*. The  
50639 returned pointer and the string content might also be invalidated if the calling thread is  
50640 terminated.

50641 **ERRORS**

50642 No errors are defined.

50643 **EXAMPLES**50644 **Getting Date and Time Formatting Information**

50645 The following example returns a pointer to a string containing date and time formatting  
50646 information, as defined in the *LC\_TIME* category of the current locale.

```
50647 #include <time.h>
50648 #include <langinfo.h>
50649 ...
50650 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
50651 ...
```

50652 **APPLICATION USAGE**

50653 The array pointed to by the return value should not be modified by the program, but may be  
50654 modified by further calls to these functions.

50655 **RATIONALE**

50656 The possible interactions between internal data used by *nl\_langinfo()* and *nl\_langinfo\_l()* are  
50657 complicated by the fact that *nl\_langinfo\_l()* must be thread-safe but *nl\_langinfo()* need not be.  
50658 The various implementation choices are:

- 50659 1. *nl\_langinfo\_l()* and *nl\_langinfo()* use separate buffers, or at least one of them does not use  
50660 an internal string buffer. In this case there are no interactions.
- 50661 2. *nl\_langinfo\_l()* and *nl\_langinfo()* share an internal per-thread buffer. There can be  
50662 interactions, but only in the same thread.
- 50663 3. *nl\_langinfo\_l()* uses an internal per-thread buffer, and *nl\_langinfo()* uses (in all threads)  
50664 the same buffer that *nl\_langinfo\_l()* uses in the initial thread. There can be interactions,  
50665 but only when *nl\_langinfo\_l()* is called in the initial thread.

50666 **FUTURE DIRECTIONS**

50667 None.

50668 **SEE ALSO**

50669 [\*setlocale\(\)\*](#), [\*uselocale\(\)\*](#)

50670 XBD Chapter 7 (on page 127), [\*<langinfo.h>\*](#), [\*<locale.h>\*](#), [\*<nl\\_types.h>\*](#)

50671 **CHANGE HISTORY**

50672 First released in Issue 2.

50673 **Issue 5**

50674 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

50675 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

50676 **Issue 7**

50677 Austin Group Interpretation 1003.1-2001 #156 is applied.

50678 The *nl\_langinfo()* function is moved from the XSI option to the Base.

50679 The *nl\_langinfo\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
50680 API Set Part 4.

50681 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0412 [302], XSH/TC1-2008/0413 [75],  
50682 XSH/TC1-2008/0414 [283], XSH/TC1-2008/0415 [75,402], XSH/TC1-2008/0416 [283], and  
50683 XSH/TC1-2008/0417 [402] are applied.

50684

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0234 [656] is applied.



50685 **NAME**

50686 nrand48 — generate uniformly distributed pseudo-random non-negative long integers

50687 **SYNOPSIS**

```
50688 XSI #include <stdlib.h>  
50689 long nrand48(unsigned short xsubi[3]);
```

50690 **DESCRIPTION**50691 Refer to *drand48()*.

50692 **NAME**

50693           ntohl, ntohs — convert values between host and network byte order

50694 **SYNOPSIS**

50695           #include <arpa/inet.h>

50696           uint32\_t ntohl(uint32\_t *netlong*);

50697           uint16\_t ntohs(uint16\_t *netshort*);

50698 **DESCRIPTION**

50699           Refer to *htonl()*.

50700 **NAME**

50701 open, openat — open file

50702 **SYNOPSIS**50703 OH `#include <sys/stat.h>`50704 `#include <fcntl.h>`50705 `int open(const char *path, int oflag, ...);`50706 `int openat(int fd, const char *path, int oflag, ...);`50707 **DESCRIPTION**

50708 The `open()` function shall establish the connection between a file and a file descriptor. It shall  
 50709 create an open file description that refers to a file and a file descriptor that refers to that open file  
 50710 description. The file descriptor is used by other I/O functions to refer to that file. The `path`  
 50711 argument points to a pathname naming the file.

50712 The `open()` function shall return a file descriptor for the named file, allocated as described in  
 50713 [Section 2.6](#) (on page 525). The open file description is new, and therefore the file descriptor shall  
 50714 not share it with any other process in the system. The `FD_CLOEXEC` file descriptor flag  
 50715 associated with the new file descriptor shall be cleared unless the `O_CLOEXEC` flag is set in  
 50716 `oflag`. The `FD_CLOFORK` file descriptor flag associated with the new file descriptor shall be  
 50717 cleared unless the `O_CLOFORK` flag is set in `oflag`.

50718 The file offset used to mark the current position within the file shall be set to the beginning of  
 50719 the file.

50720 The file status flags and file access modes of the open file description shall be set according to  
 50721 the value of `oflag`.

50722 Values for `oflag` are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 50723 in `<fcntl.h>`. Applications shall specify exactly one of the first five values (file access modes)  
 50724 below in the value of `oflag`:

50725 `O_EXEC` Open for execute only (non-directory files). If `path` names a directory and  
 50726 `O_EXEC` is not the same value as `O_SEARCH`, `open()` shall fail.

50727 `O_RDONLY` Open for reading only.

50728 `O_RDWR` Open for reading and writing. If `path` names a FIFO, and the  
 50729 implementation does not support opening a FIFO for simultaneous read  
 50730 and write, then `open()` shall fail.

50731 `O_SEARCH` Open directory for search only. If `path` names a non-directory file and  
 50732 `O_SEARCH` is not the same value as `O_EXEC`, `open()` shall fail.

50733 `O_WRONLY` Open for writing only.

50734 Any combination of the following may be used:

50735 `O_APPEND` If set, the file offset shall be set to the end of the file prior to each write.

50736 `O_CLOEXEC` If set, the `FD_CLOEXEC` flag for the new file descriptor shall be set.

50737 `O_CLOFORK` If set, the `FD_CLOFORK` flag for the new file descriptor shall be set.

50738 `O_CREAT` If the file exists, this flag has no effect except as noted under `O_EXCL`  
 50739 below. Otherwise, if `O_DIRECTORY` is not set the file shall be created as a  
 50740 regular file; the user ID of the file shall be set to the effective user ID of the  
 50741 process; the group ID of the file shall be set to the group ID of the file's  
 50742 parent directory or to the effective group ID of the process; and the access  
 50743 permission bits (see `<sys/stat.h>`) of the file mode shall be set to the value

50744			of the argument following the <i>oflag</i> argument taken as type <b>mode_t</b>
50745			modified as follows: a bitwise AND is performed on the file-mode bits
50746			and the corresponding bits in the complement of the process' file mode
50747			creation mask. Thus, all bits in the file mode whose corresponding bit in
50748			the file mode creation mask is set are cleared. When bits other than the
50749			file permission bits are set, the effect is unspecified. The argument
50750			following the <i>oflag</i> argument does not affect whether the file is open for
50751			reading, writing, or for both. Implementations shall provide a way to
50752			initialize the file's group ID to the group ID of the parent directory.
50753			Implementations may, but need not, provide an implementation-defined
50754			way to initialize the file's group ID to the effective group ID of the calling
50755			process.
50756		O_DIRECTORY	If <i>path</i> resolves to a non-directory file, fail and set <i>errno</i> to [ENOTDIR].
50757	SIO	O_DSYNC	Write I/O operations on the file descriptor shall complete as defined by
50758			synchronized I/O data integrity completion.
50759		O_EXCL	If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The
50760			check for the existence of the file and the creation of the file if it does not
50761			exist shall be atomic with respect to other threads executing <i>open()</i>
50762			naming the same filename in the same directory with O_EXCL and
50763			O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a
50764			symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the
50765			contents of the symbolic link. If O_EXCL is set and O_CREAT is not set,
50766			the result is undefined.
50767		O_NOCTTY	If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the
50768			terminal device to become the controlling terminal for the process. If <i>path</i>
50769			does not identify a terminal device, O_NOCTTY shall be ignored.
50770		O_NOFOLLOW	If <i>path</i> names a symbolic link, fail and set <i>errno</i> to [ELOOP].
50771		O_NONBLOCK	When opening a FIFO with O_RDONLY or O_WRONLY set:
50772			• If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return
50773			without delay. An <i>open()</i> for writing-only shall return an error if no
50774			process currently has the file open for reading.
50775			• If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the
50776			calling thread until a thread opens the file for writing. An <i>open()</i> for
50777			writing-only shall block the calling thread until a thread opens the
50778			file for reading.
50779			When opening a block special or character special file that supports non-
50780			blocking opens:
50781			• If O_NONBLOCK is set, the <i>open()</i> function shall return without
50782			blocking for the device to be ready or available. Subsequent
50783			behavior of the device is device-specific.
50784			• If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling
50785			thread until the device is ready or available before returning.
50786			Otherwise, the O_NONBLOCK flag shall not cause an error, but it is
50787			unspecified whether the file status flags will include the O_NONBLOCK
50788			flag.

50789	SIO	<b>O_RSYNC</b>	Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
50790			
50791			
50792			
50793			
50794			
50795			
50796	XSI SIO	<b>O_SYNC</b>	Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
50797			
50798	XSI		The O_SYNC flag shall be supported for regular files, even if the Synchronized Input and Output option is not supported.
50799			
50800		<b>O_TRUNC</b>	If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC without either O_RDWR or O_WRONLY is undefined.
50801			
50802			
50803			
50804			
50805			
50806		<b>O_TTY_INIT</b>	If <i>path</i> identifies a terminal device other than a pseudo-terminal, the device is not already open in any process, and either O_TTY_INIT is set in <i>oflag</i> or O_TTY_INIT has the value zero, <i>open()</i> shall set any non-standard <b>termios</b> structure terminal parameters to a state that provides conforming behavior (see XBD Section 11.2, on page 205) and initialize the <b>winsize</b> structure associated with the terminal to appropriate default settings. It is unspecified whether O_TTY_INIT has any effect if the device is already open in any process. If <i>path</i> identifies the subsidiary side of a pseudo-terminal that is not already open in any process, <i>open()</i> shall set any non-standard <b>termios</b> structure terminal parameters to a state that provides conforming behavior and initialize the <b>winsize</b> structure associated with the terminal to appropriate default settings, regardless of whether O_TTY_INIT is set. If <i>path</i> does not identify a terminal device, O_TTY_INIT shall be ignored.
50807			
50808			
50809			
50810			
50811			
50812			
50813			
50814			
50815			
50816			
50817			
50818			
50819			
50820			If O_CREAT and O_DIRECTORY are set and the requested access mode is neither O_WRONLY nor O_RDWR, the result is unspecified.
50821			
50822			If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall mark for update the last data access, last data modification, and last file status change timestamps of the file and the last data modification and last file status change timestamps of the parent directory.
50823			
50824			
50825			
50826			If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall mark for update the last data modification and last file status change timestamps of the file.
50827			
50828	SIO		If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.
50829			The application shall ensure that it specifies the O_TTY_INIT flag on the first open of a terminal device since system boot or since the device was closed by the process that last had it open. The application need not specify the O_TTY_INIT flag when opening pseudo-terminals. If <i>path</i> names the manager side of a pseudo-terminal device, then it is unspecified whether <i>open()</i> locks the subsidiary side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i> before opening the subsidiary side.
50830			
50831	XSI		
50832			
50833			
50834			

50835 The largest value that can be represented correctly in an object of type `off_t` shall be established  
50836 as the offset maximum in the open file description.

50837 The `openat()` function shall be equivalent to the `open()` function except in the case where `path`  
50838 specifies a relative path. In this case the file to be opened is determined relative to the directory  
50839 associated with the file descriptor `fd` instead of the current working directory. If the access mode  
50840 of the open file description associated with the file descriptor is not `O_SEARCH`, the function  
50841 shall check whether directory searches are permitted using the current permissions of the  
50842 directory underlying the file descriptor. If the access mode is `O_SEARCH`, the function shall not  
50843 perform the check.

50844 The `oflag` parameter and the optional fourth parameter correspond exactly to the parameters of  
50845 `open()`.

50846 If `openat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working  
50847 directory shall be used and the behavior shall be identical to a call to `open()`.

#### 50848 RETURN VALUE

50849 Upon successful completion, these functions shall open the file and return a non-negative  
50850 integer representing the file descriptor. Otherwise, these functions shall return `-1` and set `errno`  
50851 to indicate the error. If `-1` is returned, no files shall be created or modified.

#### 50852 ERRORS

50853 These functions shall fail if:

50854 [EACCES] Search permission is denied on a component of the path prefix, or the file  
50855 exists and the permissions specified by `oflag` are denied, or the file does not  
50856 exist and write permission is denied for the parent directory of the file to be  
50857 created, or `O_TRUNC` is specified and write permission is denied.

50858 [EEXIST] `O_CREAT` and `O_EXCL` are set, and the named file exists.

50859 [EILSEQ] `O_CREAT` was specified, the file did not exist, and the last pathname  
50860 component of `path` is not a portable filename and cannot be created in the  
50861 target directory.

50862 [EINTR] A signal was caught during `open()`.

50863 [EINVAL] The `path` argument names a FIFO, `O_RDWR` was specified, and the  
50864 SIO implementation considers this an error; or synchronized I/O flags were  
50865 specified and the implementation does not support synchronized I/O for the  
50866 file.

50867 [EISDIR] The named file is a directory and `oflag` includes `O_WRONLY` or `O_RDWR`, or  
50868 includes `O_CREAT` without `O_DIRECTORY`, or includes `O_EXEC` when  
50869 `O_EXEC` is not the same value as `O_SEARCH`.

50870 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`  
50871 argument, or `O_NOFOLLOW` was specified and the `path` argument names a  
50872 symbolic link.

50873 [EMFILE] All file descriptors available to the process are currently open.

50874 [ENAMETOOLONG]  
50875 The length of a component of a pathname is longer than `{NAME_MAX}`.

50876 [ENFILE] The maximum allowable number of files is currently open in the system.

50877	[ENOENT]	O_CREAT is not set and a component of <i>path</i> does not name an existing file, or
50878		O_CREAT is set and a component of the path prefix of <i>path</i> does not name an
50879		existing file, or <i>path</i> points to an empty string.
50880	[ENOENT] or [ENOTDIR]	
50881		O_CREAT is set, and the <i>path</i> argument contains at least one non- <code>&lt;slash&gt;</code>
50882		character and ends with one or more trailing <code>&lt;slash&gt;</code> characters. If <i>path</i>
50883		without the trailing <code>&lt;slash&gt;</code> characters would name an existing file, an
50884		[ENOENT] error shall not occur.
50885	[ENOSPC]	The directory or file system that would contain the new file cannot be
50886		expanded, the file does not exist, and O_CREAT is specified.
50887	[ENOTDIR]	A component of the path prefix names an existing file that is neither a
50888		directory nor a symbolic link to a directory; or O_CREAT and O_EXCL are not
50889		specified, the <i>path</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and
50890		ends with one or more trailing <code>&lt;slash&gt;</code> characters, and the last pathname
50891		component names an existing file that is neither a directory nor a symbolic
50892		link to a directory; or O_DIRECTORY was specified and the <i>path</i> argument
50893		names a non-directory file; or the <i>path</i> argument names a non-directory file
50894		and O_SEARCH was specified when O_SEARCH is not the same value as
50895		O_EXEC.
50896	[ENXIO]	O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no
50897		process has the file open for reading.
50898	[ENXIO]	The named file is a character special or block special file, and the device
50899		associated with this special file does not exist.
50900	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented
50901		correctly in an object of type <code>off_t</code> .
50902	[EROFS]	The named file resides on a read-only file system and either O_WRONLY,
50903		O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the <i>oflag</i>
50904		argument.
50905		The <i>openat()</i> function shall fail if:
50906	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not
50907		O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit
50908		directory searches.
50909	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is
50910		neither AT_FDCWD nor a valid file descriptor open for reading or searching.
50911	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated
50912		with a non-directory file.
50913		These functions may fail if:
50914	XSI [EAGAIN]	The <i>path</i> argument names the subsidiary side of a pseudo-terminal device that
50915		is locked.
50916	[EINVAL]	The value of the <i>oflag</i> argument is not valid.
50917	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
50918		resolution of the <i>path</i> argument.

50919	[ENAMETOOLONG]	
50920		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
50921		symbolic link produced an intermediate result with a length that exceeds
50922		{PATH_MAX}.
50923	[EOPNOTSUPP]	The <i>path</i> argument names a socket.
50924	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is
50925		O_WRONLY or O_RDWR.

## 50926 EXAMPLES

### 50927 Opening a File for Writing by the Owner

50928 The following example opens the file `/tmp/file`, either by creating it (if it does not already exist),  
 50929 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,  
 50930 the access permission bits in the file mode of the file are set to permit reading and writing by the  
 50931 owner, and to permit reading only by group members and others.

50932 If the call to `open()` is successful, the file is opened for writing.

```
50933 #include <fcntl.h>
50934 ...
50935 int fd;
50936 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
50937 char *pathname = "/tmp/file";
50938 ...
50939 fd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);
50940 ...
```

### 50941 Opening a File Using an Existence Check

50942 The following example uses the `open()` function to try to create the `LOCKFILE` file and open it  
 50943 for writing. Since the `open()` function specifies the `O_EXCL` flag, the call fails if the file already  
 50944 exists. In that case, the program assumes that someone else is updating the password file and  
 50945 exits.

```
50946 #include <fcntl.h>
50947 #include <stdio.h>
50948 #include <stdlib.h>
50949 #define LOCKFILE "/etc/ptmp"
50950 ...
50951 int pfd; /* Integer for file descriptor returned by open() call. */
50952 ...
50953 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
50954               S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
50955 {
50956     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
50957     exit(1);
50958 }
50959 ...
```



50960 **Opening a File for Writing**

50961 The following example opens a file for writing, creating the file if it does not already exist. If the  
50962 file does exist, the system truncates the file to zero bytes.

```
50963 #include <fcntl.h>
50964 #include <stdio.h>
50965 #include <stdlib.h>
50966 #define LOCKFILE "/etc/ptmp"
50967 ...
50968 int pfd;
50969 char pathname[PATH_MAX+1];
50970 ...
50971 if ((pfd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC,
50972               S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
50973 {
50974     perror("Cannot open output file\n"); exit(1);
50975 }
50976 ...
```

50977 **APPLICATION USAGE**

50978 POSIX.1-2024 does not require that terminal parameters be automatically set to any state on first  
50979 open, nor that they be reset after the last close. It is possible for a non-conforming application to  
50980 leave a terminal device in a state where the next process to use that device finds it in a non-  
50981 conforming state, but has no way of determining this. To ensure that the device is set to a  
50982 conforming initial state, applications which perform a first open of a terminal (other than a  
50983 pseudo-terminal) should do so using the O\_TTY\_INIT flag to set the parameters associated with  
50984 the terminal to a conforming state.

50985 Except as specified in this volume of POSIX.1-2024, the flags allowed in *oflag* are not mutually-  
50986 exclusive and any number of them may be used simultaneously. Not all combinations of flags  
50987 make sense. For example, using O\_SEARCH | O\_CREAT will successfully open a pre-existing  
50988 directory for searching, but if there is no existing file by that name, then it is unspecified whether  
50989 a regular file will be created. Likewise, if a non-directory file descriptor is successfully returned,  
50990 it is unspecified whether that descriptor will have execute permissions as if by O\_EXEC (note  
50991 that it is unspecified whether O\_EXEC and O\_SEARCH have the same value).

50992 The O\_CLOEXEC and O\_CLOFORK flags of *open()* are necessary to avoid a data race in multi-  
50993 threaded applications. Without O\_CLOFORK, a file descriptor is leaked into a child process  
50994 created by one thread in the window between another thread creating a file descriptor with  
50995 *open()* and then using *fcntl()* to set the FD\_CLOFORK flag. Without O\_CLOEXEC, a file  
50996 descriptor intentionally inherited by child processes is similarly leaked into an executed  
50997 program if FD\_CLOEXEC is not set atomically.

50998 **RATIONALE**

50999 Some implementations permit opening FIFOs with O\_RDWR. Since FIFOs could be  
51000 implemented in other ways, and since two file descriptors can be used to the same effect, an  
51001 implementation is allowed to reject the use of O\_RDWR on a FIFO.

51002 See *getgroups()* about the group of a newly created file.

51003 The use of *open()* to create a regular file is preferable to the use of *creat()*, because the latter is  
51004 redundant and included only for historical reasons.

51005 The use of the O\_TRUNC flag on FIFOs and directories (pipes cannot be *open()*-ed) must be  
51006 permissible without unexpected side-effects (for example, *creat()* on a FIFO must not remove

51007 data). Since terminal special files might have type-ahead data stored in the buffer, O\_TRUNC  
 51008 should not affect their content, particularly if a program that normally opens a regular file  
 51009 should open the current controlling terminal instead. Other file types, particularly  
 51010 implementation-defined ones, are left implementation-defined.

51011 POSIX.1-2024 permits [EACCES] to be returned for conditions other than those explicitly listed.

51012 The O\_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a  
 51013 controlling terminal as a side-effect of opening a terminal file. This volume of POSIX.1-2024 does  
 51014 not specify how a controlling terminal is acquired, but it allows an implementation to provide  
 51015 this on *open()* if the O\_NOCTTY flag is not set and other conditions specified in XBD [Chapter 11](#)  
 51016 (on page 199) are met.

51017 In historical implementations the value of O\_RDONLY is zero. Because of that, it is not possible  
 51018 to detect the presence of O\_RDONLY and another option. Future implementations should  
 51019 encode O\_RDONLY and O\_WRONLY as bit flags so that:

```
51020 O_RDONLY | O_WRONLY == O_RDWR
```

51021 O\_EXEC and O\_SEARCH are specified as two of the five file access modes. Since O\_EXEC does  
 51022 not apply to directories, and O\_SEARCH only applies to directories, their values need not be  
 51023 distinct. Although this standard requires *open()* to fail on an attempt to use O\_EXEC on a  
 51024 directory, or O\_SEARCH on a non-directory, this only applies in implementations where the two  
 51025 modes have distinct values. Since O\_RDONLY has historically had the value zero,  
 51026 implementations are not able to distinguish between O\_SEARCH and O\_SEARCH |  
 51027 O\_RDONLY, and similarly for O\_EXEC.

51028 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,  
 51029 the *open()* function, when called with O\_CREAT and O\_EXCL, is required to fail with [EEXIST]  
 51030 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This  
 51031 behavior is required so that privileged applications can create a new file in a known location  
 51032 without the possibility that a symbolic link might cause the file to be created in a different  
 51033 location.

51034 For example, a privileged application that must create a file with a predictable name in a user-  
 51035 writable directory, such as the user's home directory, could be compromised if the user creates a  
 51036 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can  
 51037 influence the contents of a file, the user could compromise the system by creating a new system  
 51038 configuration or spool file that would then be interpreted by the system. The test for a symbolic  
 51039 link which refers to a nonexisting file must be atomic with the creation of a new file.

51040 In addition, the *open()* function refuses to open non-directories if the O\_DIRECTORY flag is set.  
 51041 This avoids race conditions whereby a user might compromise the system by substituting a hard  
 51042 link to a sensitive file (e.g., a device or a FIFO) while a privileged application is running, where  
 51043 opening a file even for read access might have undesirable side-effects.

51044 In addition, the *open()* function does not follow symbolic links if the O\_NOFOLLOW flag is set.  
 51045 This avoids race conditions whereby a user might compromise the system by substituting a  
 51046 symbolic link to a sensitive file (e.g., a device) while a privileged application is running, where  
 51047 opening a file even for read access might have undesirable side-effects.

51048 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group  
 51049 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required  
 51050 that implementations provide a way to have the group ID be set to the group ID of the  
 51051 containing directory, but did not prohibit implementations also supporting a way to set the  
 51052 group ID to the effective group ID of the creating process. Conforming applications should not  
 51053 assume which group ID will be used. If it matters, an application can use *chown()* to set the

51054 group ID after the file is created, or determine under what conditions the implementation will  
51055 set the desired group ID.

51056 The purpose of the *openat()* function is to enable opening files in directories other than the  
51057 current working directory without exposure to race conditions. Any part of the path of a file  
51058 could be changed in parallel to a call to *open()*, resulting in unspecified behavior. By opening a  
51059 file descriptor for the target directory and using the *openat()* function it can be guaranteed that  
51060 the opened file is located relative to the desired directory. Some implementations use the  
51061 *openat()* function for other purposes as well. In some cases, if the *oflag* parameter has the  
51062 *O\_XATTR* bit set, the returned file descriptor provides access to extended attributes. This  
51063 functionality is not standardized here.

51064 Implementations are encouraged to have *open()* and *openat()* report an [EILSEQ] error if *oflag*  
51065 includes *O\_CREAT*, the file did not previously exist, and the last component of *path* contains any  
51066 bytes that have the encoded value of a <newline> character.

#### 51067 FUTURE DIRECTIONS

51068 A future version of this standard may add an *O\_NOCLOBBER* flag, specified as follows, for use  
51069 by shells when the *noclobber* option is set (see XRAT Section C.2.7.2, on page 3891):

51070 *O\_NOCLOBBER* If *O\_CREAT* and *O\_NOCLOBBER* are set, *open()* shall fail if the file exists  
51071 and is either a regular file or a symbolic link that resolves to a regular file.  
51072 The check for the existence and type of the file and the creation of the file  
51073 if it does not exist shall be atomic with respect to other threads executing  
51074 *open()* naming the same filename in the same directory with  
51075 *O\_NOCLOBBER* and *O\_CREAT* set or with *O\_EXCL* and *O\_CREAT* set.  
51076 If *O\_NOCLOBBER* and *O\_CREAT* are set, and the file exists and is either  
51077 a non-regular file or a symbolic link that resolves to a non-regular file, the  
51078 file shall be opened as if neither flag was set. If *O\_NOCLOBBER* and  
51079 *O\_CREAT* are set, and *path* names a symbolic link that does not resolve to  
51080 an existing file, an empty file shall be created such that *path* resolves to the  
51081 newly created file. If *O\_NOCLOBBER* is set and *O\_CREAT* is not set, the  
51082 result is undefined.

#### 51083 SEE ALSO

51084 *chmod()*, *close()*, *creat()*, *dirfd()*, *dup()*, *exec*, *fcntl()*, *fdopendir()*, *link()*, *lseek()*, *mkdtemp()*,  
51085 *mknod()*, *read()*, *symlink()*, *umask()*, *unlockpt()*, *write()*

51086 XBD Chapter 11 (on page 199), <fcntl.h>, <sys/stat.h>, <sys/types.h>

#### 51087 CHANGE HISTORY

51088 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 51089 Issue 5

51090 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
51091 Threads Extension.

51092 Large File Summit extensions are added.

#### 51093 Issue 6

51094 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

51095 The following new requirements on POSIX implementations derive from alignment with the  
51096 Single UNIX Specification:

- 51097 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
51098 required for conforming implementations of previous POSIX specifications, it was not  
51099 required for UNIX applications.

- 51100           • In the DESCRIPTION, O\_CREAT is amended to state that the group ID of the file is set to  
51101           the group ID of the file's parent directory or to the effective group ID of the process. This is  
51102           a FIPS requirement.
- 51103           • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open  
51104           file description. This change is to support large files.
- 51105           • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
51106           large files.
- 51107           • The [ENXIO] mandatory error condition is added.
- 51108           • The [EINVAL], [ENAMETOOLONG], and [ETXTBSY] optional error conditions are added.
- 51109           The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI  
51110           STREAMS Option Group are marked.
- 51111           The following changes were made to align with the IEEE P1003.1a draft standard:
- 51112           • An explanation is added of the effect of the O\_CREAT and O\_EXCL flags when the path  
51113           refers to a symbolic link.
- 51114           • The [ELOOP] optional error condition is added.
- 51115           The normative text is updated to avoid use of the term “must” for application requirements.
- 51116           The DESCRIPTION of O\_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.
- 51117   **Issue 7**
- 51118           Austin Group Interpretations 1003.1-2001 #113 and #143 are applied.
- 51119           Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O\_TTY\_INIT flag.
- 51120           Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the  
51121           FD\_CLOEXEC flag atomically at *open()*, and adding the F\_DUPFD\_CLOEXEC flag.
- 51122           SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 51123           This page is revised and the *openat()* function is added from The Open Group Technical  
51124           Standard, 2006, Extended API Set Part 2.
- 51125           Functionality relating to the XSI STREAMS option is marked obsolescent.
- 51126           Changes are made related to support for finegrained timestamps.
- 51127           Changes are made to allow a directory to be opened for searching.
- 51128           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0418 [292], XSH/TC1-2008/0419 [141],  
51129           XSH/TC1-2008/0420 [461], XSH/TC1-2008/0421 [390], XSH/TC1-2008/0422 [146],  
51130           XSH/TC1-2008/0423 [324], XSH/TC1-2008/0424 [292], XSH/TC1-2008/0425 [278],  
51131           XSH/TC1-2008/0426 [278], XSH/TC1-2008/0427 [291], and XSH/TC1-2008/0428 [307] are  
51132           applied.
- 51133           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0235 [873], XSH/TC2-2008/0236 [835],  
51134           XSH/TC2-2008/0237 [847], XSH/TC2-2008/0238 [817], XSH/TC2-2008/0239 [835],  
51135           XSH/TC2-2008/0240 [847], XSH/TC2-2008/0241 [822], XSH/TC2-2008/0242 [817], and  
51136           XSH/TC2-2008/0243 [943] are applied.
- 51137   **Issue 8**
- 51138           Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
51139           filenames containing any bytes that have the encoded value of a <newline> character.
- 51140           Austin Group Defect 293 is applied, adding the [EILSEQ] error.

- 51141 Austin Group Defects 658 and 1665 are applied, restricting the allowed behaviors when O\_EXEC  
51142 is used on a directory, or O\_SEARCH on a non-directory file, or O\_RDWR on a FIFO, so that the  
51143 requirements for O\_EXCL still apply.
- 51144 Austin Group Defect 1016 is applied, changing the FUTURE DIRECTIONS section.
- 51145 Austin Group Defect 1151 is applied, changing the description of O\_TTY\_INIT to include  
51146 requirements relating to the **winsize** structure.
- 51147 Austin Group Defect 1318 is applied, adding FD\_CLOFORK and O\_CLOFORK.
- 51148 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 51149 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal  
51150 devices.

51151 **NAME**

51152 open\_memstream, open\_wmemstream — open a dynamic memory buffer stream

51153 **SYNOPSIS**

```
51154 CX #include <stdio.h>
51155 FILE *open_memstream(char **bufp, size_t *sizep);
51156 #include <wchar.h>
51157 FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

51158 **DESCRIPTION**

51159 The *open\_memstream()* and *open\_wmemstream()* functions shall create an I/O stream associated  
 51160 with a dynamically allocated memory buffer. The stream shall be opened for writing and shall  
 51161 be seekable.

51162 The stream associated with a call to *open\_memstream()* shall be byte-oriented.

51163 The stream associated with a call to *open\_wmemstream()* shall be wide-oriented.

51164 The stream shall maintain a current position in the allocated buffer and a current buffer length.  
 51165 The position shall be initially set to zero (the start of the buffer). Each write to the stream shall  
 51166 start at the current position and move this position by the number of successfully written bytes  
 51167 for *open\_memstream()* or the number of successfully written wide characters for  
 51168 *open\_wmemstream()*. The length shall be initially set to zero. If a write moves the position to a  
 51169 value larger than the current length, the current length shall be set to this position. In this case a  
 51170 null character for *open\_memstream()* or a null wide character for *open\_wmemstream()* shall be  
 51171 appended to the current buffer. For both functions the terminating null is not included in the  
 51172 calculation of the buffer length.

51173 After a successful *fflush()* or *fclose()*, the pointer referenced by *bufp* shall contain the address of  
 51174 the buffer, and the variable pointed to by *sizep* shall contain the smaller of the current buffer  
 51175 length and the number of bytes for *open\_memstream()*, or the number of wide characters for  
 51176 *open\_wmemstream()*, between the beginning of the buffer and the current file position indicator.

51177 The *fseek()* and *fseeko()* functions can be used to set the file position beyond the current buffer  
 51178 length. It is implementation-defined whether this extends the buffer to the new length. If it  
 51179 extends the buffer, the added buffer contents shall be set to null bytes for *open\_memstream()*, or  
 51180 null wide characters for *open\_wmemstream()*; if it does not extend the buffer, then if data is later  
 51181 written at this point, the buffer contents in the gap shall be set to null bytes for  
 51182 *open\_memstream()*, or null wide characters for *open\_wmemstream()*. If *fseek()* or *fseeko()* is called  
 51183 with *SEEK\_END* as the *whence* argument, it is implementation-defined whether the file position  
 51184 is adjusted relative to the current buffer length or relative to the buffer size that would be set by  
 51185 an *fflush()* call made immediately before the *fseek()* or *fseeko()* call.

51186 After a successful *fflush()* the pointer referenced by *bufp* and the variable referenced by *sizep*  
 51187 remain valid only until the next write operation on the stream or a call to *fclose()*.

51188 After a successful *fclose()*, the pointer referenced by *bufp* can be passed to *free()*.

51189 **RETURN VALUE**

51190 Upon successful completion, these functions shall return a pointer to the object controlling the  
 51191 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

**51192 ERRORS**

51193 These functions shall fail if:

51194 [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

51195 These functions may fail if:

51196 [EINVAL] *bufp* or *sizep* are NULL.

51197 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

51198 [ENOMEM] Memory for the stream or the buffer could not be allocated.

**51199 EXAMPLES**

```

51200 #include <stdio.h>
51201 #include <stdlib.h>

51202 int
51203 main (void)
51204 {
51205     FILE *stream;
51206     char *buf;
51207     size_t len;
51208     off_t eob;

51209     stream = open_memstream (&buf, &len);
51210     if (stream == NULL)
51211         /* handle error */ ;
51212     fprintf (stream, "hello my world");
51213     fflush (stream);
51214     printf ("buf=%s, len=%zu\n", buf, len);
51215     eob = ftello(stream);
51216     fseeko (stream, 0, SEEK_SET);
51217     fprintf (stream, "good-bye");
51218     fseeko (stream, eob, SEEK_SET);
51219     fclose (stream);
51220     printf ("buf=%s, len=%zu\n", buf, len);
51221     free (buf);
51222     return 0;
51223 }
```

51224 This program produces the following output:

51225 buf=hello my world, len=14

51226 buf=good-bye world, len=14

**51227 APPLICATION USAGE**

51228 The buffer created by these functions should be freed by the application after closing the stream,  
51229 by means of a call to *free()*.

**51230 RATIONALE**

51231 These functions are similar to *fmemopen()* except that the memory is always allocated  
51232 dynamically by the function, and the stream is opened only for output.

51233 **FUTURE DIRECTIONS**

51234 None.

51235 **SEE ALSO**51236 *fclose()*, *fdopen()*, *fflush()*, *fmemopen()*, *fopen()*, *free()*, *freopen()*51237 XBD `<stdio.h>`, `<wchar.h>`51238 **CHANGE HISTORY**

51239 First released in Issue 7.

51240 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0244 [588] and XSH/TC2-2008/0245  
51241 [586] are applied.51242 **Issue 8**51243 Austin Group Defect 1406 is applied, clarifying the behavior of *fseek()* and *fseeko()* on streams  
51244 created by *open\_memstream()* and *open\_wmemstream()*.



51245 **NAME**

51246           openat — open file relative to directory file descriptor

51247 **SYNOPSIS**

51248           #include &lt;fcntl.h&gt;

51249           int openat(int *fd*, const char \**path*, int *oflag*, ...);51250 **DESCRIPTION**51251           Refer to *open()*.

51252 **NAME**

51253            opendir — open directory associated with file descriptor

51254 **SYNOPSIS**

51255            #include <dirent.h>

51256            DIR \*opendir(const char \**dirname*);

51257 **DESCRIPTION**

51258            Refer to *fdopendir()*.

51259 **NAME**

51260           openlog — open a connection to the logging facility

51261 **SYNOPSIS**

51262 XSI        #include &lt;syslog.h&gt;

51263        void openlog(const char \*ident, int logopt, int facility);

51264 **DESCRIPTION**51265        Refer to *closelog()*.

51266 **NAME**

51267           optarg, opterr, optind, optopt — options parsing variables

51268 **SYNOPSIS**

51269           #include <unistd.h>

51270           extern char \*optarg;

51271           extern int opterr, optind, optopt;

51272 **DESCRIPTION**

51273           Refer to [getopt\(\)](#).

51274 **NAME**

51275 pathconf — get configurable pathname variables

51276 **SYNOPSIS**

51277 #include &lt;unistd.h&gt;

51278 long pathconf(const char \*path, int name);

51279 **DESCRIPTION**51280 Refer to *fpathconf()*.

51281 **NAME**

51282 pause — suspend the thread until a signal is received

51283 **SYNOPSIS**

51284 #include &lt;unistd.h&gt;

51285 int pause(void);

51286 **DESCRIPTION**51287 The *pause()* function shall suspend the calling thread until delivery of a signal whose action is  
51288 either to execute a signal-catching function or to terminate the process.51289 If the action is to terminate the process, *pause()* shall not return.51290 If the action is to execute a signal-catching function, *pause()* shall return after the signal-catching  
51291 function returns.51292 **RETURN VALUE**51293 Since *pause()* suspends thread execution indefinitely unless interrupted by a signal, there is no  
51294 successful completion return value. A value of  $-1$  shall be returned and *errno* set to indicate the  
51295 error.51296 **ERRORS**51297 The *pause()* function shall fail if:51298 [EINTR] A signal is caught by the calling process and control is returned from the  
51299 signal-catching function.51300 **EXAMPLES**

51301 None.

51302 **APPLICATION USAGE**51303 Many common uses of *pause()* have timing windows. The scenario involves checking a  
51304 condition related to a signal and, if the signal has not occurred, calling *pause()*. When the signal  
51305 occurs between the check and the call to *pause()*, the process often blocks indefinitely. The  
51306 *sigprocmask()* and *sigsuspend()* functions can be used to avoid this type of problem.51307 **RATIONALE**

51308 None.

51309 **FUTURE DIRECTIONS**

51310 None.

51311 **SEE ALSO**51312 [\*sigsuspend\(\)\*](#)51313 XBD <[unistd.h](#)>51314 **CHANGE HISTORY**

51315 First released in Issue 1. Derived from Issue 1 of the SVID.

51316 **Issue 5**

51317 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

51318 **Issue 6**

51319 The APPLICATION USAGE section is added.

51320 **NAME**

51321 pclose — close a pipe stream to or from a process

51322 **SYNOPSIS**

```
51323 CX #include <stdio.h>
51324 int pclose(FILE *stream);
```

51325 **DESCRIPTION**

51326 The *pclose()* function shall close a stream that was opened by *popen()*, wait for the command to  
 51327 terminate, and return the termination status of the process that was running the command  
 51328 language interpreter. However, if a call caused the termination status to be unavailable to  
 51329 *pclose()*, then *pclose()* shall return  $-1$  with *errno* set to [ECHILD] to report this situation. This can  
 51330 happen if the application calls one of the following functions:

- 51331 • *wait()*
- 51332 • *waitpid()* with a *pid* argument less than or equal to 0 or equal to the process ID of the  
 51333 command line interpreter
- 51334 • Any other function not defined in this volume of POSIX.1-2024 that could do one of the  
 51335 above

51336 In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.

51337 If the command language interpreter cannot be executed, the child termination status returned  
 51338 by *pclose()* shall be as if the command language interpreter terminated using *exit(127)* or  
 51339 *\_exit(127)*.

51340 The *pclose()* function shall not affect the termination status of any child of the calling process  
 51341 other than the one created by *popen()* for the associated stream.

51342 If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of  
 51343 *pclose()* is undefined.

51344 If a thread is canceled during execution of *pclose()*, the behavior is undefined.

51345 **RETURN VALUE**

51346 Upon successful return, *pclose()* shall return the termination status of the command language  
 51347 interpreter. Otherwise, *pclose()* shall return  $-1$  and set *errno* to indicate the error.

51348 **ERRORS**

51349 The *pclose()* function shall fail if:

51350 [ECHILD] The status of the child process could not be obtained, as described above.

51351 **EXAMPLES**

51352 None.

51353 **APPLICATION USAGE**

51354 None.

51355 **RATIONALE**

51356 There is a requirement that *pclose()* not return before the child process terminates. This is  
 51357 intended to disallow implementations that return [EINTR] if a signal is received while waiting.  
 51358 If *pclose()* returned before the child terminated, there would be no way for the application to  
 51359 discover which child used to be associated with the stream, and it could not do the cleanup  
 51360 itself.

51361 If the stream pointed to by *stream* was not created by *popen()*, historical implementations of

51362 *pclose()* return `-1` without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case,  
51363 POSIX.1-2024 makes the behavior unspecified. An application should not use *pclose()* to close  
51364 any stream that was not created by *popen()*.

51365 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,  
51366 and SIGHUP while waiting for the child process to terminate. Since this behavior is not  
51367 described for the *pclose()* function in POSIX.1-2024, such implementations are not conforming.  
51368 Also, some historical implementations return [EINTR] if a signal is received, even though the  
51369 child process has not terminated. Such implementations are also considered non-conforming.

51370 Consider, for example, an application that uses:

```
51371 popen ("command", "r")
```

51372 to start *command*, which is part of the same application. The parent writes a prompt to its  
51373 standard output (presumably the terminal) and then reads from the *popen()*ed stream. The child  
51374 reads the response from the user, does some transformation on the response (pathname  
51375 expansion, perhaps) and writes the result to its standard output. The parent process reads the  
51376 result from the pipe, does something with it, and prints another prompt. The cycle repeats.  
51377 Assuming that both processes do appropriate buffer flushing, this would be expected to work.

51378 To conform to POSIX.1-2024, *pclose()* must use *waitpid()*, or some similar function, instead of  
51379 *wait()*.

51380 See the RATIONALE for *popen()* for a sample implementation of *pclose()*.

#### 51381 **FUTURE DIRECTIONS**

51382 None.

#### 51383 **SEE ALSO**

51384 [\*fork\(\)\*](#), [\*popen\(\)\*](#), [\*wait\(\)\*](#)

51385 XBD [`<stdio.h>`](#)

#### 51386 **CHANGE HISTORY**

51387 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 51388 **Issue 7**

51389 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0246 [632] is applied.

#### 51390 **Issue 8**

51391 Austin Group Defect 411 is applied, changing the RATIONALE section.



51392 **NAME**

51393 perror — write error messages to standard error

51394 **SYNOPSIS**

```
51395 #include <stdio.h>
51396 void perror(const char *s);
```

51397 **DESCRIPTION**

51398 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 51399 conflict between the requirements described here and the ISO C standard is unintentional. This  
 51400 volume of POSIX.1-2024 defers to the ISO C standard.

51401 The *perror()* function shall map the error number accessed through the symbol *errno* to a  
 51402 language-dependent error message, which shall be written to the standard error stream as  
 51403 follows:

- 51404 • First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the  
 51405 string pointed to by *s* followed by a <colon> and a <space>.
- 51406 • Then an error message string followed by a <newline>.

51407 The contents of the error message strings shall be the same as those returned by *strerror()* with  
 51408 argument *errno*.

51409 CX The *perror()* function shall mark for update the last data modification and last file status change  
 51410 timestamps of the file associated with the standard error stream at some time between its  
 51411 successful completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()* on *stderr*.

51412 The *perror()* function shall not change the orientation of the standard error stream.

51413 On error, *perror()* shall set the error indicator for the stream to which *stderr* points, and shall set  
 51414 *errno* to indicate the error.

51415 Since no value is returned, an application wishing to check for error situations should call  
 51416 *clearerr(stderr)* before calling *perror()*, then if *ferror(stderr)* returns non-zero, the value of *errno*  
 51417 indicates which error occurred.

51418 **RETURN VALUE**

51419 The *perror()* function shall not return a value.

51420 **ERRORS**

51421 CX Refer to *fputc()*.

51422 **EXAMPLES**51423 **Printing an Error Message for a Function**

51424 The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs,  
 51425 the *perror()* function prints a message and the program exits.

```
51426 #include <stdio.h>
51427 #include <stdlib.h>
51428 ...
51429 char *bufptr;
51430 size_t szbuf;
51431 ...
51432 if ((bufptr = malloc(szbuf)) == NULL) {
51433     perror("malloc"); exit(2);
51434 }
```

51435 . . .

51436 **APPLICATION USAGE**

51437 Application writers may prefer to use alternative interfaces instead of *perror()*, such as

51438 *strerror\_r()* in combination with *fprintf()*.

51439 **RATIONALE**

51440 None.

51441 **FUTURE DIRECTIONS**

51442 None.

51443 **SEE ALSO**

51444 *fprintf()*, *fputc()*, *psiginfo()*, *strerror()*

51445 XBD <**stdio.h**>

51446 **CHANGE HISTORY**

51447 First released in Issue 1. Derived from Issue 1 of the SVID.

51448 **Issue 5**

51449 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the

51450 orientation of the standard error stream.

51451 **Issue 6**

51452 Extensions beyond the ISO C standard are marked.

51453 **Issue 7**

51454 SD5-XSH-ERN-95 is applied.

51455 Changes are made related to support for finegrained timestamps.

51456 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0429 [389,401], XSH/TC1-2008/0430

51457 [389], and XSH/TC1-2008/0431 [389,401] are applied.

51458 **NAME**

51459 pipe, pipe2 — create an interprocess channel

51460 **SYNOPSIS**

```
51461 #include <unistd.h>
51462 int pipe(int fildes[2]);
51463 int pipe2(int fildes[2], int flag);
```

51464 **DESCRIPTION**

51465 The *pipe()* function shall create a pipe and place two file descriptors, one each into the  
 51466 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write  
 51467 ends of the pipe, respectively. The file descriptors shall be allocated as described in [Section 2.6](#)  
 51468 (on page 525). The FD\_CLOEXEC and FD\_CLOFORK flags shall be clear on both file  
 51469 descriptors. The O\_NONBLOCK flag shall be clear on both open file descriptions. (The *fcntl()*  
 51470 function can be used to set this flag.)

51471 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A  
 51472 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a  
 51473 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether  
 51474 *fildes*[1] is also open for reading.

51475 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open  
 51476 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

51477 The pipe's user ID shall be set to the effective user ID of the calling process.

51478 The pipe's group ID shall be set to the effective group ID of the calling process.

51479 Upon successful completion, *pipe()* shall mark for update the last data access, last data  
 51480 modification, and last file status change timestamps of the pipe.

51481 The *pipe2()* function shall be equivalent to the *pipe()* function, except that the state of  
 51482 O\_NONBLOCK on the new file descriptions and FD\_CLOEXEC and FD\_CLOFORK on the new  
 51483 file descriptors shall be determined solely by the *flag* argument, which can be constructed from a  
 51484 bitwise-inclusive OR of flags from the following list (provided by **<fcntl.h>**):

51485 O\_CLOEXEC Atomically set the FD\_CLOEXEC flag on both new file descriptors.

51486 O\_CLOFORK Atomically set the FD\_CLOFORK flag on both new file descriptors.

51487 O\_NONBLOCK Set the O\_NONBLOCK file status flag on both new file descriptions.

51488 **RETURN VALUE**

51489 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
 51490 indicate the error, no file descriptors shall be allocated and the contents of *fildes* shall be left  
 51491 unmodified.

51492 **ERRORS**

51493 The *pipe()* and *pipe2()* functions shall fail if:

51494 [EMFILE] All, or all but one, of the file descriptors available to the process are currently  
 51495 open.

51496 [ENFILE] The number of simultaneously open files in the system would exceed a  
 51497 system-imposed limit.

51498 The *pipe2()* function may fail if:

51499 [EINVAL] The value of the *flag* argument is invalid.

## 51500 EXAMPLES

### 51501 Using a Pipe to Pass Data Between a Parent Process and a Child Process

51502 The following example demonstrates the use of a pipe to transfer data between a parent process  
51503 and a child process. Error handling is excluded, but otherwise this code demonstrates good  
51504 practice when using pipes: after the *fork()* the two processes close the unused ends of the pipe  
51505 before they commence transferring data.

```
51506 #include <stdlib.h>
51507 #include <unistd.h>
51508 ...
51509 int fildes[2];
51510 const int BSIZE = 100;
51511 char buf[BSIZE];
51512 ssize_t nbytes;
51513 int status;
51514
51515 status = pipe(fildes);
51516 if (status == -1 ) {
51517     /* an error occurred */
51518     ...
51519 }
51520
51521 switch (fork()) {
51522 case -1: /* Handle error */
51523     break;
51524
51525 case 0: /* Child - reads from pipe */
51526     close(fildes[1]); /* Write end is unused */
51527     nbytes = read(fildes[0], buf, BSIZE); /* Get data from pipe */
51528     /* At this point, a further read would see end-of-file ... */
51529     close(fildes[0]); /* Finished with pipe */
51530     exit(EXIT_SUCCESS);
51531
51532 default: /* Parent - writes to pipe */
51533     close(fildes[0]); /* Read end is unused */
51534     write(fildes[1], "Hello world\n", 12); /* Write data on pipe */
51535     close(fildes[1]); /* Child will see EOF */
51536     exit(EXIT_SUCCESS);
51537 }
51538 }
```

### 51534 APPLICATION USAGE

51535 None.

### 51536 RATIONALE

51537 The wording carefully avoids using the verb “to open” in order to avoid any implication of use  
51538 of *open()*; see also *write()*.

51539 The *O\_CLOEXEC* and *O\_CLOFORK* flags of *pipe2()* are necessary to avoid a data race in multi-  
51540 threaded applications. Without *O\_CLOFORK*, a file descriptor is leaked into a child process  
51541 created by one thread in the window between another thread creating a file descriptor with  
51542 *pipe()* and then using *fcntl()* to set the *FD\_CLOFORK* flag. Without *O\_CLOEXEC*, a file  
51543 descriptor intentionally inherited by child processes is similarly leaked into an executed

- 51544 program if `FD_CLOEXEC` is not set atomically.
- 51545 Since pipes are often used for communication between a parent and child process, `O_CLOFORK`  
 51546 has to be used with care in order for the pipe to be usable. If the parent will be writing and the  
 51547 child will be reading, `O_CLOFORK` should be used when creating the pipe, and then `fcntl()`  
 51548 should be used to clear `FD_CLOFORK` for the read side of the pipe. This prevents the write side  
 51549 from leaking into other children, ensuring the child will get end-of-file when the parent closes  
 51550 the write side (although the read side can still be leaked). If the parent will be reading and the  
 51551 child will be writing, there is no way to prevent the write side being leaked (short of preventing  
 51552 other threads from creating child processes) in order to ensure the parent gets end-of-file when  
 51553 the child closes the write side, and so the two processes should use an alternative method of  
 51554 indicating the end of communications.
- 51555 Arranging for `FD_CLOEXEC` to be set appropriately is more straightforward. The parent should  
 51556 use `O_CLOEXEC` when creating the pipe and the child should clear `FD_CLOEXEC` on the side  
 51557 to be passed to the new program before calling an *exec* family function to execute it.
- 51558 The `O_NONBLOCK` flag is for convenience in avoiding additional `fcntl()` calls.
- 51559 **FUTURE DIRECTIONS**
- 51560 None.
- 51561 **SEE ALSO**
- 51562 [Section 2.6](#) (on page 525), `fcntl()`, `read()`, `write()`
- 51563 XBD `<fcntl.h>`, `<unistd.h>`
- 51564 **CHANGE HISTORY**
- 51565 First released in Issue 1. Derived from Issue 1 of the SVID.
- 51566 **Issue 6**
- 51567 The following new requirements on POSIX implementations derive from alignment with the  
 51568 Single UNIX Specification:
- 51569 • The DESCRIPTION is updated to indicate that certain dispositions of `fildes[0]` and `fildes[1]`  
 51570 are unspecified.
- 51571 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/65 is applied, adding the example to the  
 51572 EXAMPLES section.
- 51573 **Issue 7**
- 51574 SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe's user  
 51575 ID and group ID.
- 51576 Changes are made related to support for finegrained timestamps.
- 51577 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0247 [835] and XSH/TC2-2008/0248  
 51578 [467,835] are applied.
- 51579 **Issue 8**
- 51580 Austin Group Defects 411, 1318, and 1577 are applied, adding `pipe2()` and `FD_CLOFORK`.
- 51581 Austin Group Defect 1576 is applied, adding the word ``respectively''.

51582 **NAME**

51583 poll, ppoll — input/output multiplexing

51584 **SYNOPSIS**

```
51585 #include <poll.h>
51586 int poll(struct pollfd fds[], nfds_t nfds, int timeout);
51587 int ppoll(struct pollfd fds[], nfds_t nfds,
51588           const struct timespec *restrict timeout,
51589           const sigset_t *restrict sigmask);
```

51590 **DESCRIPTION**

51591 The *ppoll()* function provides applications with a mechanism for multiplexing input/output  
 51592 over a set of file descriptors. For each member of the array pointed to by *fds*, *ppoll()* shall  
 51593 examine the given file descriptor for the event(s) specified in *events*. The number of **pollfd**  
 51594 structures in the *fds* array is specified by *nfds*. The *ppoll()* function shall identify those file  
 51595 descriptors on which an application can make an attempt to read or write data without blocking,  
 51596 or on which certain events have occurred.

51597 The *poll()* function shall be equivalent to the *ppoll()* function, except as follows:

- 51598 • For the *poll()* function, the timeout period is given in milliseconds in an argument of type  
 51599 **int**, whereas for the *ppoll()* function the timeout period is given in seconds and  
 51600 nanoseconds via an argument of type pointer to **struct timespec**. A *timeout* of  $-1$  for *poll()*  
 51601 shall be equivalent to passing a null pointer for the *timeout* for *ppoll()*.
- 51602 • The *poll()* function has no *sigmask* argument; it shall behave as *ppoll()* does when *sigmask* is  
 51603 a null pointer.

51604 The *fds* argument specifies the file descriptors to be examined and the events of interest for each  
 51605 file descriptor. It is a pointer to an array with one member for each open file descriptor of  
 51606 interest. The array's members are **pollfd** structures within which *fd* specifies an open file  
 51607 descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the  
 51608 following event flags:

51609	POLLIN	The file descriptor is ready for reading data other than high-priority data.
51610	POLLRDNORM	The file descriptor is ready for reading normal data.
51611	POLLRDBAND	The file descriptor is ready for reading priority data.
51612	POLLPRI	The file descriptor is ready for reading high-priority data.
51613	POLLOUT	The file descriptor is ready for writing normal data.
51614	POLLWRNORM	Equivalent to POLLOUT.
51615	POLLWRBAND	The file descriptor is ready for writing priority data.
51616	POLLERR	An error condition is present on the file descriptor. All error conditions that 51617 arise solely from the state of the object underlying the open file description 51618 and would be diagnosed by a return of $-1$ from a <i>read()</i> or <i>write()</i> call on the 51619 file descriptor shall be reported as a POLLERR event. This flag is only valid in 51620 the <i>revents</i> bitmask; it shall be ignored in the <i>events</i> member.
51621	POLLHUP	A device has been disconnected, or a pipe or FIFO has been closed by the last 51622 process that had it open for writing. Once set, the hangup state of a FIFO shall 51623 persist until some process opens the FIFO for writing or until all read-only file 51624 descriptors for the FIFO are closed. This event and POLLOUT are mutually- 51625 exclusive. However, this event and POLLIN, POLLRDNORM,

51626 POLLRDBAND, or POLLPRI are not mutually-exclusive. This flag is only  
51627 valid in the *revents* bitmask; it shall be ignored in the *events* member.

51628 POLLNVAL The specified *fd* value is not an open file descriptor. This flag is only valid in  
51629 the *revents* member; it shall be ignored in the *events* member.

51630 A file descriptor shall be considered ready for reading when a call to an input function with  
51631 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
51632 successfully. (The function might return data, an end-of-file indication, or an error other than  
51633 one indicating that it is blocked, and in each of these cases the descriptor is considered ready for  
51634 reading.) A file descriptor shall be considered ready for writing when a call to an output  
51635 function with O\_NONBLOCK clear would not block, whether or not the function would transfer  
51636 data successfully. How much data could be written without blocking depends upon the object  
51637 underlying the open file description and its current state.

51638 The significance and semantics of normal, priority, and high-priority data are file and device-  
51639 specific. The semantics of device disconnection are device-specific.

51640 If the value of *fd* is less than 0, *events* shall be ignored, and *revents* shall be set to 0 in that entry on  
51641 return from *poll()* or *ppoll()*.

51642 In each **pollfd** structure, *poll()* or *ppoll()* shall clear the *revents* member, except that where the  
51643 application requested a report on a condition by setting one of the bits of *events* listed above,  
51644 *poll()* or *ppoll()* shall set the corresponding bit in *revents* if the requested condition is true. In  
51645 addition, *poll()* or *ppoll()* shall set the POLLHUP, POLLERR, and POLLNVAL flag in *revents* if  
51646 the condition is true, even if the application did not set the corresponding bit in *events*.

51647 The *timeout* argument controls how long the *poll()* or *ppoll()* function shall wait before timing  
51648 out. If the *timeout* argument is positive for *poll()* or not a null pointer for *ppoll()*, it specifies a  
51649 maximum interval to wait for the poll to complete. If the specified time interval expires without  
51650 any of the defined events having occurred, the function shall return. If the *timeout* argument is  
51651 -1 for *poll()* or a null pointer for *ppoll()*, then the call shall block indefinitely until at least one  
51652 descriptor meets the specified criteria or until the call is interrupted. To effect a poll, the  
51653 application shall ensure that the *timeout* argument for *poll()* is 0, or for *ppoll()* is not a null  
51654 pointer and points to a zero-valued **timespec** structure.

51655 Implementations may place limitations on the maximum timeout interval supported. All  
51656 implementations shall support a maximum timeout interval of at least 31 days for *ppoll()*. If the  
51657 *timeout* argument specifies a timeout interval greater than the implementation-defined  
51658 maximum value, the maximum value shall be used as the actual timeout value. Implementations  
51659 may also place limitations on the granularity of timeout intervals. If the requested timeout  
51660 interval requires a finer granularity than the implementation supports, the actual timeout  
51661 interval shall be rounded up to the next supported value.

51662 The *poll()* and *ppoll()* functions shall not be affected by the O\_NONBLOCK flag.

51663 The *poll()* and *ppoll()* functions shall support regular files, terminal and pseudo-terminal  
51664 devices, FIFOs, pipes, and sockets. The behavior of *poll()* and *ppoll()* on elements of *fds* that refer  
51665 to other types of file is unspecified.

51666 Regular files shall always poll TRUE for reading and writing.

51667 A file descriptor for a socket that is listening for connections shall indicate that it is ready for  
51668 reading, once connections are available. A file descriptor for a socket that is connecting  
51669 asynchronously shall indicate that it is ready for writing, once a connection has been established.

51670 Provided the application does not perform any action that results in unspecified or undefined  
51671 behavior, the value of the *fd* and *events* members of each element of *fds* shall not be modified by

51672 *poll()* or *ppoll()*.

51673 If *sigmask* is not a null pointer, the *ppoll()* function shall replace the signal mask of the caller by  
 51674 the set of signals pointed to by *sigmask* before examining the descriptors, and shall restore the  
 51675 signal mask of the calling thread before returning. If a signal is unmasked as a result of the  
 51676 signal mask being altered by *ppoll()*, and a signal-catching function is called for that signal  
 51677 during the execution of the *ppoll()* function, and SA\_RESTART is clear for the interrupting  
 51678 signal, then

- 51679 • If none of the defined events have occurred on any selected file descriptor, *ppoll()* shall  
 51680 immediately fail with the [EINTR] error after the signal-catching function returns.
- 51681 • If one or more of the defined events have occurred, it is unspecified whether *ppoll()*  
 51682 behaves the same as if none of the events had occurred (failing with [EINTR] as above) or  
 51683 behaves the same as if it was not interrupted (returning the total number of **pollfd**  
 51684 structures that have selected events).

51685 If a thread is canceled during a *ppoll()* call, it is unspecified whether the signal mask in effect  
 51686 when executing the registered cleanup functions is the original signal mask or the signal mask  
 51687 installed as part of the *ppoll()* call.

#### 51688 RETURN VALUE

51689 Upon successful completion, a non-negative value shall be returned. A positive value shall  
 51690 indicate the total number of **pollfd** structures that have selected events (that is, those for which  
 51691 the *revents* member is non-zero). A value of 0 shall indicate that the call timed out and no file  
 51692 descriptors have been selected. Upon failure, -1 shall be returned and *errno* set to indicate the  
 51693 error.

#### 51694 ERRORS

51695 The *poll()* and *ppoll()* functions shall fail if:

- |       |          |  |
|-------|----------|--|
| 51696 | [EAGAIN] | The allocation of internal data structures failed but a subsequent request may<br>51697 succeed. |
| 51698 | [EINTR]  | A signal was caught during <i>poll()</i> or <i>ppoll()</i> .                                     |
| 51699 | [EINVAL] | The <i>nfds</i> argument is greater than {OPEN_MAX}.   |

51700 The *ppoll()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 51701 | [EINVAL] | An invalid timeout interval was specified. |
|-------|----------|--|

#### 51702 EXAMPLES

51703 None.

#### 51704 APPLICATION USAGE

51705 Other than the difference in the precision of the requested timeout, the following *ppoll()* call:

```
51706 ready = ppoll(&fds, nfds, tmo_p, &sigmask);
```

51707 is equivalent to atomically executing the following calls:

```
51708 sigset_t origmask;
```

```
51709 int timeout;
```

```
51710 timeout = (tmo_p == NULL) ? -1 :
```

```
51711 (tmo_p->tv_sec * 1000 + tmo_p->tv_nsec / 1000000);
```

```
51712 pthread_sigmask(SIG_SETMASK, &sigmask, &origmask);
```

```
51713 ready = poll(&fds, nfds, timeout);
```

```
51714 pthread_sigmask(SIG_SETMASK, &origmask, NULL);
```



51715 When a *poll()* or *ppoll()* call indicates a file descriptor is ready for reading, this means that if an  
 51716 attempt to read data had been made at the time that the status of the file descriptor was checked,  
 51717 it would have returned at least one byte of data, an end-of-file indication, or an error, without  
 51718 blocking (even if *O\_NONBLOCK* is clear). When a *poll()* or *ppoll()* call indicates that a file  
 51719 descriptor is ready for writing, this means that if an attempt to write one byte of data had been  
 51720 made at the time that the status of the file descriptor was checked, it would have written that  
 51721 byte or returned an error, without blocking. However, if an attempt to write more than one byte  
 51722 had been made, it might have blocked (if *O\_NONBLOCK* is clear). In both cases, by the time the  
 51723 call returns and a subsequent I/O operation is attempted, the state of the file descriptor might  
 51724 have changed (for example, because another thread read or wrote some data) and, if  
 51725 *O\_NONBLOCK* is clear, there is no guarantee that the operation will not block (unless it would  
 51726 not block for some other reason, such as setting *MIN=0* and *TIME=0* for a terminal in non-  
 51727 canonical mode). Therefore it is recommended that applications always set *O\_NONBLOCK* on  
 51728 file descriptors whose readiness for I/O they query with *poll()* or *ppoll()*.

51729 The error conditions specified for *read()* and *write()* that are reported as *POLLERR* events are  
 51730 only those that arise solely from the state of the object underlying the open file description.  
 51731 They do not include, for example, *[EAGAIN]* as this relates to the state of the open file  
 51732 description not (solely) the object underlying it.

51733 Application writers should note that repeating a *poll()* or *ppoll()* call which indicated that I/O  
 51734 was possible on one or more of the file descriptors given, without causing some change to the  
 51735 state, either by altering the *fds* array or causing appropriate input or output to occur on at least  
 51736 one file descriptor indicated as ready, will result in “busy waiting”—a subsequent call will  
 51737 always return immediately indicating the same (or perhaps more) events as the previous one.

#### 51738 RATIONALE

51739 The *POLLHUP* event does not occur for FIFOs just because the FIFO is not open for writing. It  
 51740 only occurs when the FIFO is closed by the last writer and persists until some process opens the  
 51741 FIFO for writing or until all read-only file descriptors for the FIFO are closed.

51742 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers  
 51743 can install an additional cancellation handler which resets the signal mask to the expected value:

```
51744 void cleanup(void *arg)
51745 {
51746     sigset_t *ss = (sigset_t *) arg;
51747     pthread_sigmask(SIG_SETMASK, ss, NULL);
51748 }
51749 int call_ppoll(struct pollfd fds[], nfds_t nfds,
51750               const struct timespec *restrict timeout,
51751               const sigset_t *restrict sigmask)
51752 {
51753     sigset_t oldmask;
51754     int result;
51755     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
51756     pthread_cleanup_push(cleanup, &oldmask);
51757     result = ppoll(fds, nfds, timeout, sigmask);
51758     pthread_cleanup_pop(0);
51759     return result;
51760 }
```

51761 **FUTURE DIRECTIONS**

51762 None.

51763 **SEE ALSO**51764 *pselect()*, *read()*, *write()*

51765 XBD &lt;poll.h&gt;

51766 **CHANGE HISTORY**

51767 First released in Issue 4, Version 2.

51768 **Issue 5**

51769 Moved from X/OPEN UNIX extension to BASE.

51770 The description of POLLWRBAND is updated.

51771 **Issue 6**

51772 Text referring to sockets is added to the DESCRIPTION.

51773 Functionality relating to the XSI STREAMS Option Group is marked.

51774 The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of  
51775 POLLWRBAND.51776 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/66 is applied, correcting the spacing in  
51777 the EXAMPLES section.51778 **Issue 7**

51779 Austin Group Interpretation 1003.1-2001 #209 is applied, clarifying the POLLHUP event.

51780 The *poll()* function is moved from the XSI option to the Base.

51781 Functionality relating to the XSI STREAMS option is marked obsolescent.

51782 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0249 [623] and XSH/TC2-2008/0250  
51783 [683] are applied.51784 **Issue 8**51785 Austin Group Defect 1263 is applied, adding *ppoll()*.

51786 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

51787 Austin Group Defect 1448 is applied, aligning the wording relating to file descriptor readiness  
51788 with *pselect()* and changing the APPLICATION USAGE section.

51789 **NAME**

51790 popen — initiate pipe streams to or from a process

51791 **SYNOPSIS**

```
51792 CX #include <stdio.h>
51793 FILE *popen(const char *command, const char *mode);
```

51794 **DESCRIPTION**

51795 The `popen()` function shall execute the command specified by the string `command`. It shall create  
 51796 a pipe between the calling program and the executed command, and shall return a pointer to a  
 51797 stream that can be used to either read from or write to the pipe.

51798 The environment of the executed command shall be as if a child process were created within the  
 51799 `popen()` call using the `fork()` function, and the child invoked the `sh` utility using the call:

```
51800 execl(<shell path>, "sh", "-c", "--", command, (char *)0);
```

51801 where `<shell path>` is an unspecified pathname for the `sh` utility. It is implementation-defined  
 51802 whether the handlers registered with `pthread_atfork()` are called as part of the creation of the  
 51803 child process.

51804 The `popen()` function shall ensure that any streams from previous `popen()` calls that remain open  
 51805 in the parent process are closed in the new child process, regardless of the `FD_CLOEXEC` or  
 51806 `FD_CLOFORK` status of the file descriptor underlying those streams.

51807 The `mode` argument to `popen()` is a string that specifies I/O mode:

- 51808 1. If `mode` starts with 'r', when the child process is started, its file descriptor  
 51809 `STDOUT_FILENO` shall be the writable end of the pipe, and the file descriptor  
 51810 `fileno(stream)` in the calling process, where `stream` is the stream pointer returned by  
 51811 `popen()`, shall be the readable end of the pipe. The `FD_CLOFORK` flag shall be cleared on  
 51812 both the `STDOUT_FILENO` file descriptor passed to the child process and the file  
 51813 descriptor underlying the returned stream.
- 51814 2. If `mode` starts with 'w', when the child process is started its file descriptor  
 51815 `STDIN_FILENO` shall be the readable end of the pipe, and the file descriptor `fileno(stream)`  
 51816 in the calling process, where `stream` is the stream pointer returned by `popen()`, shall be the  
 51817 writable end of the pipe. The `FD_CLOFORK` flag shall be cleared on both the  
 51818 `STDOUT_FILENO` file descriptor passed to the child process and the file descriptor  
 51819 underlying the returned stream.
- 51820 3. If `mode` includes a second character of 'e', then the file descriptor underlying the stream  
 51821 returned to the calling process by `popen()` shall have the `FD_CLOEXEC` flag atomically  
 51822 set. Additionally, if the implementation creates the file descriptor for use by the child  
 51823 process from within the parent process, then that file descriptor shall have the  
 51824 `FD_CLOEXEC` flag atomically set within the parent process. If `mode` does not have a  
 51825 second character, the `FD_CLOEXEC` flag of the underlying file descriptor returned by  
 51826 `popen()` shall be clear.
- 51827 4. If `mode` is any other value, the result is unspecified.

51828 After `popen()`, both the parent and the child process shall be capable of executing independently  
 51829 before either terminates.

51830 Pipe streams are byte-oriented.

51831 **RETURN VALUE**

51832 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to  
 51833 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate  
 51834 the error.

51835 **ERRORS**

51836 The *popen()* function shall fail if:

51837 [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

51838 The *popen()* function may fail if:

51839 [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

51840 [EINVAL] The *mode* argument is invalid.

51841 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

51842 **EXAMPLES**51843 **Using popen() to Obtain a List of Files from the ls Utility**

51844 The following example demonstrates the use of *popen()* and *pclose()* to execute the command *ls\**  
 51845 in order to obtain a list of files in the current directory:

```
51846 #include <stdio.h>
51847 ...
51848 FILE *fp;
51849 int status;
51850 char path[PATH_MAX];
51851 fp = popen("ls *", "r");
51852 if (fp == NULL)
51853     /* Handle error */;
51854 while (fgets(path, PATH_MAX, fp) != NULL)
51855     printf("%s", path);
51856 status = pclose(fp);
51857 if (status == -1) {
51858     /* Error reported by pclose() */
51859     ...
51860 } else {
51861     /* Use macros described under wait() to inspect `status' in order
51862        to determine success/failure of command executed by popen() */
51863     ...
51864 }
```

51865 **APPLICATION USAGE**

51866 Since open files are shared, a mode 'r' command can be used as an input filter and a mode 'w'  
 51867 command as an output filter.

51868 Buffered reading before opening an input filter may leave the standard input of that filter  
 51869 mispositioned. Similar problems with an output filter may be prevented by careful buffer  
 51870 flushing; for example, with *fflush()*.

51871 A stream opened by *popen()* should be closed by *pclose()*.

51872 The behavior of *popen()* is specified for values of *mode* of "r", "w", "re", and "we". Other

51873 modes such as "rb" and "wb" might be supported by specific implementations, but these  
 51874 would not be portable features. Note that historical implementations of *popen()* only check to see  
 51875 if the first character of *mode* is 'r'. Thus, a *mode* of "robert the robot" would be treated as  
 51876 *mode* "r", and a *mode* of "anything else" would be treated as *mode* "w".

51877 If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a  
 51878 stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process  
 51879 started by *popen()*.

51880 To determine whether or not the environment specified in the Shell and Utilities volume of  
 51881 POSIX.1-2024 is present, use the function call:

```
51882 sysconf(_SC_2_VERSION)
```

51883 (See *sysconf()*).

#### 51884 RATIONALE

51885 The *popen()* function should not be used by programs that have set user (or group) ID privileges.  
 51886 The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead.  
 51887 This prevents any unforeseen manipulation of the environment of the user that could cause  
 51888 execution of commands not anticipated by the calling program.

51889 If the original and *popen()*ed processes both intend to read or write or read and write a common  
 51890 file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for  
 51891 sharing file handles must be observed (see [Section 2.5.1](#), on page 522).

51892 The 'e' mode modifier to *popen()* is necessary to avoid a data race in multi-threaded  
 51893 applications. Without it, the parent's file descriptor is leaked into a second child process created  
 51894 by one thread in the window between another thread creating the pipe via *popen()* then using  
 51895 *fileno()* and *fcntl()* on the result. Also, if the *popen()* implementation temporarily has the child's  
 51896 file descriptor open within the parent, then that file descriptor could also be leaked if it is not  
 51897 atomically FD\_CLOEXEC for the duration in which it is open in the parent.

51898 The standard only requires that the implementation atomically set FD\_CLOEXEC on file  
 51899 descriptors created in the parent process when the 'e' mode modifier is in effect;  
 51900 implementations may also do so when the 'e' modifier is not in use, provided that the  
 51901 FD\_CLOEXEC bit is eventually cleared before *popen()* completes, however, this is not required  
 51902 because any application worried about the potential file descriptor leak will already be using the  
 51903 'e' modifier.

51904 Implementations are encouraged to add support for a "wf" mode which creates the pipe as if by  
 51905 calling *pipe2()* with the O\_CLOFORK flag and then clearing FD\_CLOFORK for the read side of  
 51906 the pipe. This prevents the write side from leaking into child processes created by other threads,  
 51907 ensuring the child created by *popen()* will get end-of-file when the parent closes the write side  
 51908 (although the read side can still be leaked). Unfortunately there is no way (short of temporarily  
 51909 preventing other threads from creating child processes, or implementing an atomic create-pipe-  
 51910 and-fork system call) to implement an "rf" mode with the equivalent guarantee that the child  
 51911 created by *popen()* will be the only writer. Therefore multi-threaded applications that do not  
 51912 have complete control over process creation cannot rely on getting end-of-file on the stream and  
 51913 need to use an alternative method of indicating the end of communications.

51914 Although the standard is clear that a conforming application should not call *popen()* when file  
 51915 descriptor 0 or 1 is closed, implementations are encouraged to handle these cases correctly.

51916 The following two examples demonstrate possible implementations of *popen()* using other  
 51917 standard functions. These examples are designed to show FD\_CLOEXEC handling rather than  
 51918 all aspects of thread safety, and implementations are encouraged to improve the locking

mechanism around the state list to be more efficient, as well as to be more robust if file descriptor 0 or 1 is returned as either part of the pipe. Also, remember that other implementations are possible, including one that uses an implementation-specific means of creating a pipe between parent and child where the parent process never has access to the child's end of the pipe. Both of these examples make use of the following helper functions, documented but not implemented here, to do the bookkeeping necessary to properly close all file descriptors created by other *popen()* calls regardless of their `FD_CLOEXEC` or `FD_CLOFORK` status:

```

51926 /* Obtain mutual exclusion lock, so that no concurrent popen() or
51927      pclose() calls are simultaneously modifying the list of tracked
51928      children. */
51929 static void popen_lock(void);

51930 /* Release mutual exclusion lock, without changing errno. */
51931 static void popen_unlock(void);

51932 /* Add the pid and stream pair to the list of tracked children, prior
51933      to any code that can clear FD_CLOEXEC on the file descriptor
51934      associated with stream. To be used while holding the lock. */
51935 static void popen_add_pair(FILE *stream, pid_t pid);

51936 /* Given a stream, return the associated pid, or -1 with errno set if
51937      the stream was not created by popen(). To be used while holding
51938      the lock. */
51939 static pid_t popen_get_pid(FILE *stream);

51940 /* Remove stream and its corresponding pid from the list of tracked
51941      children. To be used while holding the lock. */
51942 static void popen_remove(FILE *stream);

51943 /* If stream is NULL, return the first tracked child; otherwise,
51944      return the next tracked child. Return NULL if all tracked children
51945      have been returned. To be used while holding the lock. */
51946 static FILE *popen_next(FILE *stream);

```

51947 The first example is based on *fork()*:

```

51948 #include <stdio.h>
51949 #include <errno.h>
51950 #include <fcntl.h>
51951 #include <sys/wait.h>
51952 #include <unistd.h>
51953 FILE *popen(const char *command, const char *mode)
51954 {
51955     int fds[2];
51956     pid_t pid;
51957     FILE *stream;
51958     int target = mode[0] == 'w'; /* index of fds used by parent */

51959     /* Validate mode */
51960     if ((mode[0] != 'w' && mode[0] != 'r') ||
51961         mode[1 + (mode[1] == 'e')]) {
51962         errno = EINVAL;
51963         return NULL;
51964     }

51965     /* Create pipe and stream with FD_CLOEXEC set */

```

```

51966     if (pipe2(fds, O_CLOEXEC) < 0)
51967         return NULL;
51968     stream = fdopen(fds[target], mode);
51969     if (!stream) {
51970         int saved = errno;
51971         close(fds[0]);
51972         close(fds[1]);
51973         errno = saved;
51974         return NULL;
51975     }

51976     /* Create child process */
51977     popen_lock();
51978     pid = fork();
51979     if (pid < 0) {
51980         int saved = errno;
51981         close(fds[!target]);
51982         fclose(stream);
51983         popen_unlock();
51984         errno = saved;
51985         return NULL;
51986     }

51987     /* Child process. */
51988     if (!pid) {
51989         FILE *tracked = popen_next(NULL);
51990         while (tracked) {
51991             int fd = fileno(tracked);
51992             if (fd < 0 || close(fd))
51993                 _exit(127);
51994             tracked = popen_next(tracked);
51995         }
51996         target = mode[0] == 'r'; /* Opposite fd in the child */
51997         /* Use dup2 or fcntl to clear FD_CLOEXEC on child's descriptor,
51998            FD_CLOEXEC will take care of the rest of fds[]. */
51999         if (fds[target] != target) {
52000             if (dup2(fds[target], target) != target)
52001                 _exit(127);
52002             } else {
52003                 int flags = fcntl(fds[target], F_GETFD);
52004                 if (flags < 0 ||
52005                     fcntl(fds[target], F_SETFD, flags & ~FD_CLOEXEC) < 0)
52006                     _exit(127);
52007             }
52008             execl("/bin/sh", "sh", "-c", "--", command, NULL);
52009             _exit(127);
52010         }

52011     /* Parent process. From here on out, the close and fcntl system
52012        calls are assumed to pass, since all inputs are valid and do not
52013        require allocating any fds or memory. Besides, excluding
52014        failures due to undefined behavior (such as another thread
52015        closing an fd it knows nothing about), cleanup from any defined

```

```

52016         failures would require stopping and reaping the child process,
52017         which may have worse consequences. */
52018     close(fds[!target]);  popen_add_pair(stream, pid);
52019     popen_unlock();
52020     if (mode[1] != 'e') {
52021         int flags = fcntl(fds[target], F_GETFD);
52022         if (flags >= 0)
52023             fcntl(fds[target], F_SETFD, flags & ~FD_CLOEXEC);
52024     }
52025     return stream;
52026 }

```

52027 The second example is based on *posix\_spawn()*:

```

52028 #include <stdio.h>
52029 #include <errno.h>
52030 #include <fcntl.h>
52031 #include <sys/wait.h>
52032 #include <unistd.h>
52033 #include <spawn.h>
52034 extern char **environ;
52035 FILE *popen(const char *command, const char *mode)
52036 {
52037     int fds[2];
52038     pid_t pid;
52039     FILE *stream;
52040     int target = mode[0] == 'w'; /* index of fds used by parent */
52041     const char *argv[] = { "sh", "-c", "--", command, NULL };
52042     posix_spawn_file_actions_t actions;
52043     int saved;
52044     FILE *tracked;
52045
52046     /* Validate mode */
52047     if ((mode[0] != 'w' && mode[0] != 'r') ||
52048         mode[1 + (mode[1] == 'e')]) {
52049         errno = EINVAL;
52050         return NULL;
52051     }
52052
52053     /* Create pipe and stream with FD_CLOEXEC set */
52054     if (pipe2(fds, O_CLOEXEC) < 0)
52055         return NULL;
52056     stream = fdopen(fds[target], mode);
52057     if (!stream) {
52058         saved = errno;
52059         close(fds[0]);
52060         close(fds[1]);
52061         errno = saved;
52062         return NULL;
52063     }
52064
52065     /* Create child process */
52066     if (posix_spawn_file_actions_init(&actions)) {
52067         saved = errno;

```



```

52065         goto spawnerr1;
52066     }
52067     popen_lock();
52068     tracked = popen_next(NULL);
52069     while (tracked) {
52070         int fd = fileno(tracked);
52071         if (fd < 0 || posix_spawn_file_actions_addclose(&actions, fd))
52072             goto spawnerr2;
52073         tracked = popen_next(tracked);
52074     }
52075     if (posix_spawn_file_actions_adddup2(&actions, fds[!target], !target))
52076         goto spawnerr2;
52077     if (posix_spawn(&pid, "/bin/sh", &actions, NULL, (char **)argv,
52078     environ)) {
52079     spawnerr2:
52080         saved = errno;
52081         posix_spawn_file_actions_destroy(&actions);
52082         popen_unlock();
52083     spawnerr1:
52084         close(fds[!target]);
52085         fclose(stream);
52086         errno = saved;
52087         return NULL;
52088     }
52089     /* From here on out, system calls are assumed to pass, since all
52090     inputs are valid and do not require allocating any fds or memory.
52091     Besides, excluding failures due to undefined behavior (such as
52092     another thread closing an fd it knows nothing about), cleanup
52093     from any defined failures would require stopping and reaping the
52094     child process, which may have worse consequences. */
52095     posix_spawn_file_actions_destroy(&actions);
52096     close(fds[!target]);
52097     popen_add_pair(stream, pid);
52098     popen_unlock();
52099     if (mode[1] != 'e') {
52100         int flags = fcntl(fds[target], F_GETFD);
52101         if (flags >= 0)
52102             fcntl(fds[target], F_SETFD, flags & ~FD_CLOEXEC);
52103     }
52104     return stream;
52105 }

```

52106 Both examples can share a common *pclose()* implementation.

```

52107 int pclose(FILE *stream)
52108 {
52109     int status;
52110     popen_lock();
52111     pid_t pid = popen_get_pid(stream);
52112     if (pid < 0) {
52113         popen_unlock();
52114         return -1;

```

```

52115     }
52116     popen_remove(stream);
52117     popen_unlock();
52118     fclose(stream); /* Ignore failure */
52119     while (waitpid(pid, &status, 0) == -1) {
52120         if (errno != EINTR) {
52121             status = -1;
52122             break;
52123         }
52124     }
52125     return status;
52126 }

```

52127 Note that, while a particular implementation of *popen()* (such as the two above) can assume a  
52128 particular path for the shell, such a path is not necessarily valid on another system. The above  
52129 examples are not portable, and are not intended to be.

52130 Earlier versions of this standard required the *command* string to be passed as the next argument  
52131 after "-c" (omitting "--"). This meant that portable applications needed to take care not to  
52132 pass a command string beginning with <hyphen-minus> ('-') or <plus-sign> ('+'), as it  
52133 would then be interpreted as containing options. Now that implementations are required to pass  
52134 the "--", applications no longer need to do this.

#### 52135 FUTURE DIRECTIONS

52136 None.

#### 52137 SEE ALSO

52138 [Section 2.5](#) (on page 521), *fork()*, *pclose()*, *pipe()*, *sysconf()*, *system()*, *wait()*, *waitid()*

52139 XBD [<stdio.h>](#)

52140 XCU *sh*

#### 52141 CHANGE HISTORY

52142 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 52143 Issue 5

52144 A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

#### 52145 Issue 6

52146 The following new requirements on POSIX implementations derive from alignment with the  
52147 Single UNIX Specification:

- 52148 • The optional [EMFILE] error condition is added.

52149 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/67 is applied, adding the example to the  
52150 EXAMPLES section.

#### 52151 Issue 7

52152 Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for *mode* in the  
52153 DESCRIPTION.

52154 SD5-XSH-ERN-149 is applied, changing the {STREAM\_MAX} [EMFILE] error condition from a  
52155 “may fail” to a “shall fail”.

52156 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0432 [14] is applied.

52157 **Issue 8**

52158 Austin Group Defects 411 and 1318 are applied, adding the 'e' mode modifier and  
52159 FD\_CLOFORK, and adding example implementations of *popen()* and *pclose()* in the  
52160 RATIONALE section.

52161 Austin Group Defect 1317 is applied, making it implementation-defined whether the handlers  
52162 registered with *pthread\_atfork()* are called.

52163 Austin Group Defect 1440 is applied, adding a "--" argument to the specified *execl()* call.

52164 Austin Group Defect 1526 is applied, making typographic changes relating to *mode* values for  
52165 consistency with *fopen()*.

52166 **NAME**

52167        posix\_close — close a file descriptor

52168 **SYNOPSIS**

52169        #include <unistd.h>

52170        int posix\_close(int *fd*, int *flag*);

52171 **DESCRIPTION**

52172        Refer to *close()*.

52173 **NAME**

52174 posix\_devctl — device control

52175 **SYNOPSIS**

```
52176 DC      #include <devctl.h>
52177
52177      int posix_devctl(int fildev, int dcmd, void *restrict dev_data_ptr,
52178                      size_t nbyte, int *restrict dev_info_ptr);
```

52179 **DESCRIPTION**

52180 The *posix\_devctl()* function shall cause the device control command *dcmd* to be passed to the  
 52181 driver identified by *fildev*. Associated data shall be passed to and/or from the driver depending  
 52182 on direction information encoded in the *dcmd* argument or as implied in the *dcmd* argument by  
 52183 the design and implementation of the driver.

52184 If the *dev\_data\_ptr* argument is not a null pointer, it shall be a pointer to a buffer that is provided  
 52185 by the caller and that contains data bytes to be passed to the driver or provides space for  
 52186 receiving data bytes to be passed back from the driver, or both.

52187 If the data bytes are to be passed to the driver, at least *nbyte* bytes of associated data shall be  
 52188 made available to the driver; if the data bytes are to be passed from the driver, no more than  
 52189 *nbyte* bytes shall be passed.

52190 The driver may be executing in an address space different from the address space of the calling  
 52191 thread. Therefore, if the data bytes passed to the driver (i.e., the contents of the memory area  
 52192 starting at *dev\_data\_ptr* and continuing for *nbyte* bytes) contain pointers to memory in the  
 52193 address space of the calling thread and the driver uses these pointers to access that memory, the  
 52194 effects are unspecified.

52195 OB If *dev\_data\_ptr* is not a null pointer and *nbyte* is zero, the amount of data passed to and/or from  
 52196 the driver is unspecified. This feature is obsolescent and is only provided for compatibility with  
 52197 existing device drivers.

52198 If *dev\_data\_ptr* is a null pointer, there shall be no data bytes passed between the caller and the  
 52199 driver other than the data specified in the rest of the arguments to *posix\_devctl()* and in its return  
 52200 value.

52201 The *dev\_info\_ptr* argument provides the opportunity to return an integer number containing  
 52202 additional device information, instead of just a success/failure indication. For implementation-  
 52203 provided *dcmd* values, it is implementation-defined whether each such value causes the **int**  
 52204 pointed to by *dev\_info\_ptr* to be set and, if set, what value it is set to.

52205 For each supported device, the set of valid *dcmd* commands, the associated data interpretation,  
 52206 and the effects of the command on the device are all defined by the driver for the device  
 52207 identified by *fildev*, and are therefore implementation-defined for implementation-provided  
 52208 device drivers.

52209 **RETURN VALUE**

52210 Upon successful completion, *posix\_devctl()* shall return zero; otherwise an error number shall be  
 52211 returned to indicate the error. The value returned in the **int** value pointed to by *dev\_info\_ptr* is  
 52212 driver dependent.

52213 **ERRORS**

52214 The *posix\_devctl()* function shall fail if:

52215 [EBADF] The *fildev* argument is not a valid open file descriptor.

52216 The *posix\_devctl()* function may fail if:

- 52217 [EINTR] The *posix\_devctl()* function was interrupted by a signal.
- 52218 [EINVAL] The *nbyte* argument exceeds an implementation-defined maximum or is less than the minimum number of bytes required for this command.
- 52219
- 52220 [EINVAL] The *dcmd* argument is not valid for this device.
- 52221 [ENOTTY] The *fildev* argument is not associated with a character special file that accepts control functions.
- 52222
- 52223 [EPERM] The requesting process does not have the appropriate privilege to request the device to perform the specified command.
- 52224
- 52225 Driver code may detect other errors, but the error numbers returned are driver dependent. See
- 52226 “Recommended Practice for Driver-Detected Errors” in RATIONALE.
- 52227 If the *posix\_devctl()* function fails, the effect of this failed function on the device is driver
- 52228 dependent. Corresponding data might be transferred, partially transferred, or not transferred at
- 52229 all.

52230 **EXAMPLES**

52231 None.

52232 **APPLICATION USAGE**

52233 None.

52234 **RATIONALE**

52235 **Background**

52236 An interface to be included in the POSIX standard should improve source code portability of  
 52237 application programs. In traditional UNIX practice, *ioctl()* was used to handle special devices.  
 52238 Therefore, a general specification of its arguments cannot be written. Based on this fact, in the  
 52239 past many people claimed that *ioctl()*, or something close to it, had no place in the POSIX  
 52240 standards.

52241 Against this perception stood the widespread use of *ioctl()* to interface to all sorts of drivers for a  
 52242 vast variety of hardware used in all areas of general-purpose, realtime, and embedded  
 52243 computing, such as analog-digital converters, counters, and video graphic devices. These  
 52244 devices provide a set of services that cannot be represented or used in terms of *read()* or *write()*  
 52245 calls.

52246 The arguments in favor of *ioctl()* standardization can be summarized as follows:

52247 Even if *ioctl()* addresses very different hardware, many of these devices either are actually the  
 52248 same, interfaced to different computer systems with different implementations of operating  
 52249 systems, or belong to classes of devices with rather high commonality in their functions, e.g.,  
 52250 analog-digital converters or digital-analog converters. Growing standardization of the control  
 52251 and status register (CSR) space of these devices allows exploitation of a growing similarity of  
 52252 control codes and data for these devices. A general mechanism is needed to control these  
 52253 devices.

52254 In all these cases, a standardized interface from the application program to drivers for these  
 52255 devices will improve source code portability.

52256 Even if control codes and device data have to be changed when porting applications from one  
 52257 system to another, the definition of *ioctl()* largely improves readability of a program handling  
 52258 special devices. Changes are confined to more clearly labeled places.

52259 A driver for a specific device normally cannot be considered portable *per se*, but an application

52260 that uses this driver can be made portable if all interfaces needed are well defined and  
 52261 standardized. Users and integrators of realtime systems often add device drivers for specific  
 52262 devices, and a standard interface simplifies this process. Also, device drivers often follow their  
 52263 special hardware from system to system.

52264 In recognition of these reasons, The Open Group included *ioctl()* in the The Single UNIX  
 52265 Specification, Version 1, and the interface was later incorporated into POSIX.1 under the XSI  
 52266 STREAMS option (although that option was subsequently removed).

52267 The *posix\_devctl()* interface defined in this standard provides an alternative to the the various  
 52268 *ioctl()* implementations with a standard interface that captures the extensibility of *ioctl()*, but  
 52269 avoids several of its deficiencies, which is mentioned in “Relationship to *ioctl()* and the  
 52270 Perceived Needs for Improvement” below.

### 52271 Existing Practice

52272 The *ioctl()* interface is widely used. It has provided the generality mentioned above. Existing  
 52273 practice encodes into the second parameter information about data size and direction in some  
 52274 systems. An example of such an encoding is the use in BSD 4.3 of two bits of the command word  
 52275 as read/write bits. However, *ioctl()* has definite problems with the way that its sometimes  
 52276 optional third parameter can be interpreted.

52277 This practice is similar to the existing POSIX *fcntl()* function, in which the third parameter can  
 52278 be optional for the F\_GETFD and F\_GETFL commands, an **int** when used with the F\_DUPFD,  
 52279 F\_SETFD, or F\_SETFL commands, or a **struct flock** when used with the F\_GETLK, F\_SETLK, or  
 52280 F\_SETLKW commands. However, the *fcntl()* interface defines two distinct and known data  
 52281 types as possible for the third parameter. This is not the case in the *ioctl()* interface, where any  
 52282 number of device driver specific structures and commands are used.

### 52283 Relationship to *ioctl()* and the Perceived Needs for Improvement

52284 [Section A.11](#) (on page 3718) briefly mentions some of the perceived deficiencies in existing  
 52285 implementations of the *ioctl()* function, in the context of those *ioctl()* commands used to  
 52286 implement terminal control. The standard developers decided that, since the set of such control  
 52287 operations was fairly well defined, suitable encapsulations such as *tcsetattr()*, *tcsendbreak()*, and  
 52288 *tcdrain()* could be standardized. These interfaces, while successfully standardizing portable  
 52289 terminal control operations, are not extensible to arbitrary user-supplied devices.

52290 There are several perceived deficiencies with the *ioctl()* function that drove the development of  
 52291 the *posix\_devctl()* interface as an alternative:

- 52292 • The major problem with *ioctl()* is that the third argument (when one is passed) varies in  
 52293 both size and type according to the second (command) argument. It is not unprecedented  
 52294 in POSIX, or standards in general, for a function to accept a generic pointer; consider the  
 52295 ISO C function *fread()*, or the POSIX functions *read()* and *mmap()*. However, in all such  
 52296 instances, the generic pointer is accompanied by a size argument that specifies the size of  
 52297 the pointed-to object. Unlike the Ada language, it is, and has always been, the C  
 52298 programmer’s responsibility to ensure that these two arguments form a consistent  
 52299 specification of the passed object. But traditional *ioctl()* implementations do not allow the  
 52300 user to specify the size of the pointed-to object; that size is instead fixed implicitly by the  
 52301 specified command (passed as another argument). The *posix\_devctl()* interface improves  
 52302 upon *ioctl()* in that it allows the user to specify the object size, thereby restoring the  
 52303 familiar C paradigm for passing a generic object by pointer/size pair.

- 52304 • A secondary problem with *ioctl()* is that the third argument is sometimes permitted to be  
52305 interpreted as an integer (**int**). The *posix\_devctl()* interface clearly requires the *dev\_data\_ptr*  
52306 argument to be a pointer.
- 52307 • A related problem with *ioctl()* is that the direction(s) in which data are transferred to or  
52308 from the pointed-to object is neither specified explicitly as an argument (as with *mmap()*),  
52309 nor implied by the *ioctl()* function (as with *read()/write()*, *fread()/fwrite()*, or  
52310 *fgets()/fputs()*). Instead, the direction is implied by the command argument. In traditional  
52311 implementations, only the device driver knows the interpretation of the commands and  
52312 whether data bytes are to be transferred to or from the pointed-to object. But in networked  
52313 implementations, generic portions of the operating system may need to know the direction  
52314 to ensure that data bytes are passed properly between a client and a server, separately from  
52315 device driver concerns. Two implementation-specific solutions to this problem are to  
52316 always assume data bytes need to be transferred in both directions, or to encode the  
52317 implied direction into the command word along with the fixed data size. The *posix\_devctl()*  
52318 interface already provides the implementation with an explicit size parameter. Since the  
52319 direction is already known implicitly to both the application and the driver and since  
52320 workable methods exist for implementations to ascertain that direction if required, this  
52321 perceived problem is strictly an implementation issue and solvable without further impact  
52322 on the interface.
- 52323 • Finally, *posix\_devctl()* improves upon *ioctl()* by adopting the new style of error return,  
52324 avoiding all the problems *errno* brings to multi-threaded applications. Because the driver-  
52325 specific information carried by the non-error return values of *ioctl()* still potentially needs  
52326 to be passed to the application, *posix\_devctl()* adds the *dev\_info\_ptr* argument to specify  
52327 where this information should be stored.

#### 52328 Which Differences Between *posix\_devctl()* and *ioctl()* Are Acceptable?

52329 Any differences between the definitions of *posix\_devctl()* and *ioctl()* have to be perceived as  
52330 a clear improvement by the community of potential users. Drivers for normal peripherals are  
52331 typically written by highly specialized professionals. Drivers for the special devices are very  
52332 often written by the application developer or by the hardware designer. Any interface definition  
52333 that can be seen as overly complicated will simply not be accepted.

52334 Nevertheless, a few simple and useful improvements to *ioctl()* are possible, specifically the  
52335 improvement of type checking, and justify the definition of a new interface.

52336 The major difference between the two interfaces is the addition of the size of the device data. For  
52337 enhanced compatibility with existing *ioctl()* implementations, this size can be specified as zero;  
52338 in this case the amount of data passed is unspecified. (This allows a macro definition of *ioctl()*  
52339 that converts it into a *posix\_devctl()* call.) In any case, the data size argument does not contradict  
52340 the general goal of being able to implement *posix\_devctl()* using the existing *ioctl()* interfaces  
52341 provided in current UNIX systems and other POSIX implementations because the standard  
52342 allows but does not require checking the size of the device data. Although the third argument of  
52343 the *ioctl()* function does not specify a size, it is implicit in the specific combination of control  
52344 command and driver and, therefore, known to the driver implementation.

52345 The method of indicating error return values differs from traditional *ioctl()* implementations, but  
52346 it does not preclude the construction of *posix\_devctl()* as a macro built upon *ioctl()*, which was  
52347 one of the original design goals.



### 52348 **Rationale for the *dev\_info\_ptr* Argument**

52349 The POSIX.26 developers felt that it was important to preserve the current *ioctl()* functionality of  
 52350 allowing a device driver to return some arbitrary piece of information instead of just a  
 52351 success/failure indication. Such information might be, for example, the number of bytes  
 52352 received, the number of bytes that would not fit into the buffer pointed at by *dev\_data\_ptr*, the  
 52353 data type indication, or the device status. Current practice for device drivers and *ioctl()* usage  
 52354 allows such a device-dependent return value. Thus, the concept of an additional output  
 52355 argument, *dev\_info\_ptr*, was born.

### 52356 **Rationale for No *direction* Argument**

52357 The initial specification for *posix\_devctl()* contained an additional argument that specified the  
 52358 direction of data flow, i.e., to the driver and/or from the driver. This argument was later  
 52359 removed for the following reasons:

- 52360 • The argument was redundant. Most (if not all) existing implementations encode the  
 52361 direction data either explicitly or implicitly in the command word.
- 52362 • The argument increased the probability of programming errors, since it must be made to  
 52363 agree with the direction information already encoded or implied in the command word or  
 52364 an error would occur.
- 52365 • The only real use of the argument would be if new drivers were written that supported  
 52366 generic commands such as TRANSFER\_CONTROL\_DATA, which was modified by the  
 52367 direction argument to indicate in which direction the data should be transferred. This is  
 52368 contrary to current practice that uses command pairs such as GET\_CONTROL\_DATA and  
 52369 PUT\_CONTROL\_DATA.
- 52370 • The primary purpose of the direction argument was to allow higher levels of the system to  
 52371 identify the direction of data transfers, particularly in the case of remote devices, without  
 52372 having to understand all the commands of all the devices on the system. Implementations  
 52373 that need to ascertain the direction of data transfer from a command word will define a  
 52374 consistent convention for encoding the direction into each command word, and all device  
 52375 drivers supplied by the user must adhere to this convention.

52376 Thus, the data direction argument was removed.

### 52377 **Rationale for Not Defining the Direction Encoding in the *dcmd* Argument**

52378 The POSIX.26 developers gave consideration to defining the direction encoding in the *dcmd*  
 52379 argument, but decided against doing so. No particular benefit was seen to a predefined  
 52380 encoding, as long as the encoding was used consistently across the entire implementation and  
 52381 was well known to the implementation.

52382 In addition, although only one encoding (BSD's) employed for *ioctl()* was known among the  
 52383 members of the small working group, it could not be ruled out that other encodings already  
 52384 existed, and no reason for precluding these encodings was seen.

52385 Finally, system or architectural constraints might make a chosen standard encoding difficult to  
 52386 use on a given implementation.

52387 Thus, this standard does not define a direction encoding. Specifying a standard encoding is  
 52388 actually a small part of a larger and more contentious objective, that of specifying a complete set  
 52389 of interfaces for portable device drivers. If a future POSIX standard specifies such interfaces, the  
 52390 issue of device control direction encoding will necessarily be addressed as part of that  
 52391 specification.

52392 **Recommended Practice for Handling Data Size Errors**

52393 In the event that the amount of data from the device is too large to fit into the specified buffer, as  
 52394 much data as will fit should be transferred, and the error posted. The retained data will aid in  
 52395 debugging, even if some of the data is lost.

52396 **Recommended Practice for *nbyte* == 0**

52397 The feature that permits an unspecified amount of control data to be transferred if *nbyte* is zero  
 52398 exists only for compatibility with existing device driver usage of *ioctl()*,  
 52399 i.e., when *ioctl()* is implemented on top of *posix\_devctl()* and the device driver transfers an  
 52400 amount of data implied by the command.

52401 Implementations in which *posix\_devctl()* is built as a library routine on top of *ioctl()* may not be  
 52402 able to make checks on the *nbyte* argument. However, newly developed applications using  
 52403 *posix\_devctl()* should always use an appropriate value for the *nbyte* argument, for portability to  
 52404 implementations directly supporting *posix\_devctl()* in which the device drivers may be able to  
 52405 honor the application's *nbyte* argument or return the error [EINVAL] if the argument is an  
 52406 unacceptable value. Device drivers designed for those systems should interpret a zero value of  
 52407 *nbyte* as no data to be transferred.

52408 **Recommended Practice for Driver-Detected Errors**

52409 If the driver detects the following error conditions, it is recommended that the *posix\_devctl()*  
 52410 function fail and return the corresponding error number:

52411	[EBUSY]	The control operation could not complete successfully because the device was
52412		in use by another process, or the driver was unable to carry out the request
52413		due to an outstanding operation in progress.
52414	[EINVAL]	The arguments <i>dev_data_ptr</i> and <i>nbyte</i> define a buffer too small to hold the
52415		amount of data expected by or to be returned by this driver.
52416	[EIO]	The control operation could not complete successfully because the driver
52417		detected a hardware error.

52418 **FUTURE DIRECTIONS**

52419 None.

52420 **SEE ALSO**

52421 XBD [Section 4.21](#) (on page 108), [<devctl.h>](#)

52422 **CHANGE HISTORY**

52423 First released in Issue 8. Derived from POSIX.26.

52424 **NAME**52425 posix\_fadvise — file advisory information (**ADVANCED REALTIME**)52426 **SYNOPSIS**

```
52427 ADV #include <fcntl.h>
52428 int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

52429 **DESCRIPTION**

52430 The *posix\_fadvise()* function shall advise the implementation on the expected behavior of the  
 52431 application with respect to the data in the file associated with the open file descriptor, *fd*, starting  
 52432 at *offset* and continuing for *len* bytes. The specified range need not currently exist in the file. If *len*  
 52433 is zero, all data from *offset* to the largest possible value of the file offset for that file shall be  
 52434 specified. The implementation may use this information to optimize handling of the specified  
 52435 data. The *posix\_fadvise()* function shall have no effect on the semantics of other operations on the  
 52436 specified data, although it may affect the performance of other operations.

52437 The advice to be applied to the data is specified by the *advice* parameter and may be one of the  
 52438 following values:

52439 **POSIX\_FADV\_NORMAL**

52440 Specifies that the application has no advice to give on its behavior with respect to the  
 52441 specified data. It is the default characteristic if no advice is given for an open file.

52442 **POSIX\_FADV\_SEQUENTIAL**

52443 Specifies that the application expects to access the specified data sequentially from lower  
 52444 offsets to higher offsets.

52445 **POSIX\_FADV\_RANDOM**

52446 Specifies that the application expects to access the specified data in a random order.

52447 **POSIX\_FADV\_WILLNEED**

52448 Specifies that the application expects to access the specified data in the near future.

52449 **POSIX\_FADV\_DONTNEED**

52450 Specifies that the application expects that it will not access the specified data in the near  
 52451 future.

52452 **POSIX\_FADV\_NOREUSE**

52453 Specifies that the application expects to access the specified data once and then not reuse it  
 52454 thereafter.

52455 These values are defined in **<fcntl.h>**.

52456 **RETURN VALUE**

52457 Upon successful completion, *posix\_fadvise()* shall return zero; otherwise, an error number shall  
 52458 be returned to indicate the error.

52459 **ERRORS**

52460 The *posix\_fadvise()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 52461 | [EBADF]  | The <i>fd</i> argument is not a valid file descriptor.                               |
| 52462 | [EINVAL] | The value of <i>advice</i> is invalid, or the value of <i>len</i> is less than zero. |
| 52463 | [ESPIPE] | The <i>fd</i> argument is associated with a pipe or FIFO.                            |

52464 **EXAMPLES**

52465 None.

52466 **APPLICATION USAGE**

52467 The *posix\_fadvise()* function is part of the Advisory Information option and need not be  
52468 provided on all implementations.

52469 **RATIONALE**

52470 None.

52471 **FUTURE DIRECTIONS**

52472 None.

52473 **SEE ALSO**52474 *posix\_madvise()*

52475 XBD &lt;fcntl.h&gt;

52476 **CHANGE HISTORY**

52477 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

52478 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

52479 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/68 is applied, changing the function  
52480 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the  
52481 standard developers felt it acceptable to make this change before implementations of this  
52482 function become widespread.

52483 **Issue 7**

52484 Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the  
52485 [EINVAL] error.

52486 **Issue 8**52487 Austin Group Defect 1135 is applied, clarifying the requirements when *len* is zero.

52488 **NAME**52489 posix\_fallocate — file space control (**ADVANCED REALTIME**)52490 **SYNOPSIS**

```
52491 ADV #include <fcntl.h>
52492 int posix_fallocate(int fd, off_t offset, off_t len);
```

52493 **DESCRIPTION**

52494 The *posix\_fallocate()* function shall ensure that any required storage for regular file data starting  
 52495 at *offset* and continuing for *len* bytes is allocated on the file system storage media. If  
 52496 *posix\_fallocate()* returns successfully, subsequent writes to the specified file data shall not fail due  
 52497 to the lack of free space on the file system storage media.

52498 If the *offset+len* is beyond the current file size, then *posix\_fallocate()* shall adjust the file size to  
 52499 *offset+len*. Otherwise, the file size shall not be changed.

52500 It is implementation-defined whether a previous *posix\_fadvise()* call influences allocation  
 52501 strategy.

52502 Space allocated via *posix\_fallocate()* shall be freed by a successful call to *creat()* or *open()* that  
 52503 truncates the size of the file. Space allocated via *posix\_fallocate()* may be freed by a successful call  
 52504 to *truncate()* that reduces the file size to a size smaller than *offset+len*.

52505 **RETURN VALUE**

52506 Upon successful completion, *posix\_fallocate()* shall return zero; otherwise, an error number shall  
 52507 be returned to indicate the error.

52508 **ERRORS**

52509 The *posix\_fallocate()* function shall fail if:

- |           |           |   |
|-----------|-----------|---|
| 52510     | [EBADF]   | The <i>fd</i> argument is not a valid file descriptor.  |
| 52511     | [EBADF]   | The <i>fd</i> argument references a file that was opened without write permission.  |
| 52512     | [EFBIG]   | The value of <i>offset+len</i> is greater than the maximum file size.   |
| 52513 XSI | [EFBIG]   | The value of <i>offset+len</i> exceeds the file size limit of the process. A SIGXFSZ signal shall also be generated for the thread. |
| 52514     |           |   |
| 52515     | [EINTR]   | A signal was caught during execution.   |
| 52516     | [EINVAL]  | The <i>len</i> argument is less than zero, or the <i>offset</i> argument is less than zero.   |
| 52517     | [EIO]     | An I/O error occurred while reading from or writing to a file system.   |
| 52518     | [ENODEV]  | The <i>fd</i> argument does not refer to a regular file.  |
| 52519     | [ENOSPC]  | There is insufficient free space remaining on the file system storage media.  |
| 52520     | [ENOTSUP] | The underlying file system does not support this operation.   |
| 52521     | [ESPIPE]  | The <i>fd</i> argument is associated with a pipe or FIFO.   |
| 52522     |           | The <i>posix_fallocate()</i> function may fail if:  |
| 52523     | [EINVAL]  | The <i>len</i> argument is zero.  |

52524 **EXAMPLES**

52525 None.

52526 **APPLICATION USAGE**52527 The *posix\_fallocate()* function is part of the Advisory Information option and need not be  
52528 provided on all implementations.52529 Not all file systems are capable of supporting *posix\_fallocate()*, in which case the function will  
52530 return [ENOTSUP]. However, if a system supports the Advisory Information option, there must  
52531 be at least one file system that is capable of supporting this function and is available for use in  
52532 conforming environments. The *pathconf()* and *fpathconf()* functions can be used to determine  
52533 whether a file in a particular file system supports *posix\_fallocate()*.52534 **RATIONALE**

52535 None.

52536 **FUTURE DIRECTIONS**

52537 None.

52538 **SEE ALSO**52539 *creat()*, *fruncate()*, *open()*, *unlink()*

52540 XBD &lt;fcntl.h&gt;

52541 **CHANGE HISTORY**

52542 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

52543 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

52544 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/69 is applied, changing the function  
52545 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the  
52546 standard developers felt it acceptable to make this change before implementations of this  
52547 function become widespread.52548 **Issue 7**52549 Austin Group Interpretations 1003.1-2001 #022, #024, and #162 are applied, changing the  
52550 definition of the [EINVAL] error.52551 **Issue 8**

52552 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.

52553 Austin Group Defect 684 is applied, requiring an [ENOTSUP] error instead of [EINVAL] when  
52554 the underlying file system does not support *posix\_fallocate()*.

52555 Austin Group Defect 687 is applied, changing the APPLICATION USAGE section.

52556 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
52557 relating to the file size limit for the process.

52558 **NAME**

52559 posix\_getdents — read directory entries

52560 **SYNOPSIS**

52561 #include &lt;dirent.h&gt;

52562 ssize\_t posix\_getdents(int *fildes*, void \**buf*, size\_t *nbyte*, int *flags*);52563 **DESCRIPTION**

52564 The *posix\_getdents()* function shall attempt to read directory entries from the directory associated  
 52565 with the open file descriptor *fildes* and shall place information about the directory entries and the  
 52566 files they refer to in **posix\_dent** structures in the buffer pointed to by *buf*, up to a maximum of  
 52567 *nbyte* bytes. The number of **posix\_dent** structures populated in *buf* may be fewer than the  
 52568 number that will fit in *nbyte* bytes, but shall be at least one if *nbyte* is greater than the size of the  
 52569 **posix\_dent** structure plus [NAME\_MAX] and *fildes* is not currently at end-of-file.

52570 The application shall ensure that *buf* is aligned suitably to point to a **posix\_dent** structure. The  
 52571 alignment needed shall not be more restrictive than the alignment provided by *malloc()*. Strictly  
 52572 conforming applications shall ensure that the value of *flags* is zero; other applications can set it to  
 52573 a value constructed by a bitwise-inclusive OR of implementation-defined bitwise-distinct flag  
 52574 values.

52575 Each **posix\_dent** structure returned in *buf* shall be located at an address that satisfies the  
 52576 implementation's alignment requirements for the **posix\_dent** structure and shall be populated  
 52577 as follows:

- 52578 • The value of the *d\_ino* member shall be set to the file serial number of the file named by the  
 52579 *d\_name* member.
- 52580 • The value of the *d\_reclen* member shall be set to the number of bytes occupied by this entry  
 52581 in *buf*, including any padding bytes needed before the next entry, if any. If this is the last  
 52582 entry in *buf*, *d\_reclen* shall include any padding bytes needed to make the address of this  
 52583 entry plus *d\_reclen* bytes satisfy the alignment requirements for the **posix\_dent** structure.
- 52584 • The value of the *d\_type* member shall be set to indicate the file type of the named file, if the  
 52585 file type can be determined without needing to use the file serial number to obtain the  
 52586 file's metadata; otherwise it may be set to DT\_UNKNOWN. If the file type is determined  
 52587 and it is one of the file types defined in this standard, the value of *d\_type* shall be DT\_BLK,  
 52588 DT\_CHR, DT\_DIR, DT\_FIFO, DT\_LNK, DT\_REG, DT\_SOCK, DT\_MQ, DT\_SEM,  
 52589 DT\_SHM, or DT\_TMO (see <dirent.h>). If it is determined but is not a standard file type,  
 52590 the value of *d\_type* shall not equal any of those listed here.
- 52591 • The *d\_name* member shall be a filename string, and (if not dot or dot-dot) shall contain the  
 52592 same byte sequence as the last pathname component of the string used to create the  
 52593 directory entry, plus the terminating NUL byte.

52594 If the *d\_name* member names a symbolic link, the values of the *d\_ino* and *d\_type* members shall  
 52595 be set to the values for the symbolic link itself.

52596 The *posix\_getdents()* function shall start reading at the current file offset in the open file  
 52597 description associated with *fildes*. On successful return, the file offset shall be incremented to  
 52598 point to the directory entry immediately following the last entry whose information was  
 52599 returned in *buf*, or to point to end-of-file if there are no more directory entries. On failure, the  
 52600 value of the file offset is unspecified. The current file offset can be set and retrieved using *lseek()*  
 52601 on the open file description associated with *fildes*. The behavior is unspecified if *lseek()* is used  
 52602 to set the file offset to a value other than zero or a value returned by a previous call to *lseek()* on  
 52603 the same open file description.

52604 The *posix\_getdents()* function shall not return directory entries containing empty names. If  
 52605 entries for dot or dot-dot exist, a sequence of calls that reads from offset zero to end-of-file shall  
 52606 return one entry for dot and one entry for dot-dot; otherwise, they shall not be returned.

52607 Upon successful completion, *posix\_getdents()* shall mark for update the last data access  
 52608 timestamp of the directory.

52609 If *fildev* is a file descriptor associated with a directory stream opened using *fdopendir()* or  
 52610 *opendir()*, the behavior is unspecified.

52611 If *posix\_getdents()* is called concurrently with an operation that adds, deletes, or modifies a  
 52612 directory entry, the results from *posix\_getdents()* shall reflect either all of the effects of the  
 52613 concurrent operation or none of them. If a sequence of calls to *posix\_getdents()* is made that reads  
 52614 from offset zero to end-of-file and a file is removed from or added to the directory between the  
 52615 first and last of those calls, whether the sequence of calls returns an entry for that file is  
 52616 unspecified.

52617 **RETURN VALUE**

52618 Upon successful completion, either a non-negative integer shall be returned indicating the  
 52619 number of bytes occupied by the **posix\_dent** structures placed in *buf* or 0 shall be returned  
 52620 indicating the end of the directory was reached without any directory entries being placed in *buf*.  
 52621 Otherwise, -1 shall be returned and *errno* shall be set to indicate the error.

52622 **ERRORS**

52623 The *posix\_getdents()* function shall fail if:

- 52624 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.
- 52625 [EINVAL] The *nbyte* argument is not large enough to contain the information to be  
 52626 returned about the directory entry located at the current file offset.
- 52627 [ENOENT] The current file offset is not located at a valid directory entry.
- 52628 [ENOTDIR] The *fildev* argument is associated with a non-directory file.
- 52629 [EOVERFLOW] One of the values in a structure to be placed in *buf* cannot be represented  
 52630 correctly.

52631 The *posix\_getdents()* function may fail if:

- 52632 [EIO] A physical I/O error has occurred.
- 52633 [ENOMEM] Insufficient memory was available to fulfill the request.

52634 **EXAMPLES**

52635 This example function lists the files in a specified directory with their file serial number and file  
 52636 type. If the file type is not available from *posix\_getdents()*, it is obtained using *fstatat()*.

```
52637 #include <dirent.h>
52638 #include <fcntl.h>
52639 #include <stdio.h>
52640 #include <stdlib.h>
52641 #include <sys/stat.h>
52642 #include <unistd.h>
52643 #define ENTBUSIZ 10240
52644 int list_dir(const char *dirnam)
52645 {
52646     int fd = open(dirnam, O_RDONLY | O_DIRECTORY);
```



```

52647     if (fd == -1)
52648         return -1;

52649     char *buf = malloc(ENTBUFSIZ);
52650     if (buf == NULL)
52651     {
52652         close(fd);
52653         return -1;
52654     }

52655     ssize_t bytesinbuf;
52656     for(;;)
52657     {
52658         ssize_t nextent = 0;

52659         bytesinbuf = posix_getdents(fd, buf, ENTBUFSIZ, 0);
52660         if (bytesinbuf <= 0)
52661             break;

52662         do {
52663             const char *ftype;
52664             struct posix_dent *entp = (void *)&buf[nextent];
52665             if (entp->d_type == DT_UNKNOWN)
52666             {
52667                 struct stat stbuf;
52668                 if (fstatat(fd, entp->d_name, &stbuf,
52669                     AT_SYMLINK_NOFOLLOW) == -1)
52670                     ftype = "?";
52671                 else
52672                     ftype = S_ISBLK(stbuf.st_mode) ? "b" :
52673                         S_ISCHR(stbuf.st_mode) ? "c" :
52674                         S_ISDIR(stbuf.st_mode) ? "d" :
52675                         S_ISFIFO(stbuf.st_mode) ? "p" :
52676                         S_ISLNK(stbuf.st_mode) ? "l" :
52677                         S_ISREG(stbuf.st_mode) ? "r" :
52678                         S_ISSOCK(stbuf.st_mode) ? "s" :
52679                         S_TYPEISMQ(&stbuf) ? "mq" :
52680                         S_TYPEISSEM(&stbuf) ? "sem" :
52681                         S_TYPEISSHM(&stbuf) ? "shm" :
52682             #ifdef S_TYPEISTMO
52683                 S_TYPEISTMO(&stbuf) ? "tmo" :
52684             #endif
52685                 "?";
52686             }
52687             else
52688             {
52689                 ftype = entp->d_type == DT_BLK ? "b" :
52690                     entp->d_type == DT_CHR ? "c" :
52691                     entp->d_type == DT_DIR ? "d" :
52692                     entp->d_type == DT_FIFO ? "p" :
52693                     entp->d_type == DT_LNK ? "l" :
52694                     entp->d_type == DT_REG ? "r" :
52695                     entp->d_type == DT_SOCKET ? "s" :
52696                     entp->d_type == DT_MQ ? "mq" :

```

```

52697             entp->d_type == DT_SEM ? "sem" :
52698             entp->d_type == DT_SHM ? "shm" :
52699     #ifdef DT_TMO
52700             entp->d_type == DT_TMO ? "tmo" :
52701     #endif
52702             "?";
52703     }
52704     printf("%ld\t%s\t%s\n", (long)entp->d_ino, ftype,
52705           entp->d_name);
52706     nextent += entp->d_reclen;
52707     } while (nextent < bytesinbuf);
52708 }
52709
52709     close(fd);
52710     free(buf);
52711     return bytesinbuf;
52712 }

```

### APPLICATION USAGE

If an array of **posix\_dent** structures (which is only possible on implementations where *d\_name* is not a flexible array member) is used to provide the storage for *buf* in order to satisfy the alignment requirement, it should be noted that the number of array elements used to size the array may bear little or no relation to the number of directory entries that can be stored in it. It is recommended that the number of elements is calculated from the desired size in bytes, for example:

```

52720 #define DESIREDSIZE 10240
52721 struct posix_dent buf[DESIREDSIZE / sizeof(struct posix_dent) + 1];
52722 size_t nbyte = sizeof buf;

```

When *posix\_getdents()* is called with a *buf* that is not type **char \***, it is important to note that *d\_reclen* is a byte count and therefore any pointer arithmetic involved in calculating the start of the next entry needs to use a **char \*** pointer.

On implementations where directory entries in a directory take up more space than the corresponding **posix\_dent** structures in *buf*, a call to *posix\_getdents()* may read *nbyte* bytes from the directory, resulting (in most cases) in the actual number of bytes placed in *buf* being less than *nbyte*.

One advantage of *posix\_getdents()* is that it provides the file type of each directory entry (if available), whereas *readdir()* only does so on implementations that have the file type as a non-standard additional member of the **dirent** structure. Knowing the file type can greatly reduce the number of *fstatat()* calls that need to be made when traversing the file hierarchy.

Whether or not a file's type can be determined without needing to use the file serial number to obtain the file's metadata may vary across the different file system types supported by an implementation. Therefore applications should not assume that if *d\_type* contains known file types (i.e. not DT\_UNKNOWN) for entries in a given directory then it will also contain known file types for entries in subdirectories of that directory or in its parent.

Since the *d\_reclen* value for the last entry in *buf* includes padding to satisfy alignment requirements, applications can grow the buffer and call *posix\_getdents()* again to append to it without needing to perform an alignment calculation.

52742 **RATIONALE**

52743 The `posix_getdents()` function was derived from existing `getdents()` functions but the name was  
52744 changed because the existing `getdents()` functions differed in various ways, in particular the type  
52745 of the second argument (structure pointer or `void *`), the members of the populated structures,  
52746 and the error numbers used for some conditions. The name change also provided an  
52747 opportunity to add a `flags` argument to provide for future extensibility.

52748 Implementations are encouraged to include support for a `DT_FORCE_TYPE` flag which, when  
52749 that bit is set in `flags`, causes `posix_getdents()` to look up the file type if it can not be obtained from  
52750 the directory entry. This will allow applications that need to know the file type of every directory  
52751 entry to keep the cost of these lookups to the minimum needed to obtain the type at the file  
52752 system level, without the additional overhead of making a call to `fstatat()` for every file (that has  
52753 `d_type` equal to `DT_UNKNOWN`).

52754 Some existing `getdents()` or similar functions return directory entry structures for deleted  
52755 directory entries in `buf`, marked with a special value of one of the structure members to  
52756 distinguish them from non-deleted entries. This behavior is not allowed for `posix_getdents()`,  
52757 although the data from a deleted directory entry may be present in `buf` in the form of extra  
52758 padding on the end of the previous entry.

52759 **FUTURE DIRECTIONS**

52760 A future version of this standard may add a `DT_FORCE_TYPE` flag as described in  
52761 RATIONALE.

52762 **SEE ALSO**

52763 [\*fdopendir\(\)\*](#), [\*fstatat\(\)\*](#), [\*lseek\(\)\*](#), [\*readdir\(\)\*](#)

52764 XBD <[\*\*dirent.h\*\*](#)>

52765 **CHANGE HISTORY**

52766 First released in Issue 8.

52767 **NAME**

52768 posix\_madvise — memory advisory information and alignment control (**ADVANCED**  
52769 **REALTIME**)

52770 **SYNOPSIS**

```
52771 ADV #include <sys/mman.h>
52772 int posix_madvise(void *addr, size_t len, int advice);
```

52773 **DESCRIPTION**

52774 The *posix\_madvise()* function shall advise the implementation on the expected behavior of the  
52775 application with respect to the data in the memory starting at address *addr*, and continuing for  
52776 *len* bytes. The implementation may use this information to optimize handling of the specified  
52777 data. The *posix\_madvise()* function shall have no effect on the semantics of access to memory in  
52778 the specified range, although it may affect the performance of access.

52779 The implementation may require that *addr* be a multiple of the page size, which is the value  
52780 returned by *sysconf()* when the name value `_SC_PAGESIZE` is used.

52781 The advice to be applied to the memory range is specified by the *advice* parameter and may be  
52782 one of the following values:

52783 `POSIX_MADV_NORMAL`

52784 Specifies that the application has no advice to give on its behavior with respect to the  
52785 specified range. It is the default characteristic if no advice is given for a range of memory.

52786 `POSIX_MADV_SEQUENTIAL`

52787 Specifies that the application expects to access the specified range sequentially from lower  
52788 addresses to higher addresses.

52789 `POSIX_MADV_RANDOM`

52790 Specifies that the application expects to access the specified range in a random order.

52791 `POSIX_MADV_WILLNEED`

52792 Specifies that the application expects to access the specified range in the near future.

52793 `POSIX_MADV_DONTNEED`

52794 Specifies that the application expects that it will not access the specified range in the near  
52795 future.

52796 These values are defined in the `<sys/mman.h>` header.

52797 **RETURN VALUE**

52798 Upon successful completion, *posix\_madvise()* shall return zero; otherwise, an error number shall  
52799 be returned to indicate the error.

52800 **ERRORS**

52801 The *posix\_madvise()* function shall fail if:

52802 [EINVAL] The value of *advice* is invalid.

52803 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly  
52804 or completely outside the range allowed for the address space of the calling  
52805 process.

- 52806 The *posix\_madvise()* function may fail if:
- 52807 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the  
52808 name value *\_SC\_PAGESIZE* is used.
- 52809 [EINVAL] The value of *len* is zero.
- 52810 **EXAMPLES**
- 52811 None.
- 52812 **APPLICATION USAGE**
- 52813 The *posix\_madvise()* function is part of the Advisory Information option and need not be  
52814 provided on all implementations.
- 52815 **RATIONALE**
- 52816 None.
- 52817 **FUTURE DIRECTIONS**
- 52818 None.
- 52819 **SEE ALSO**
- 52820 *mmap()*, *posix\_fadvise()*, *sysconf()*
- 52821 XBD <[sys/mman.h](#)>
- 52822 **CHANGE HISTORY**
- 52823 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
- 52824 IEEE PASC Interpretation 1003.1 #102 is applied.

52825 **NAME**

52826        posix\_mem\_offset — find offset and length of a mapped typed memory block (**ADVANCED**  
52827        **REALTIME**)

52828 **SYNOPSIS**

```
52829 TYM     #include <sys/mman.h>
52830     int posix_mem_offset(const void *restrict addr, size_t len,
52831         off_t *restrict off, size_t *restrict contig_len,
52832         int *restrict fildes);
```

52833 **DESCRIPTION**

52834        The *posix\_mem\_offset()* function shall return in the variable pointed to by *off* a value that  
52835        identifies the offset (or location), within a memory object, of the memory block currently  
52836        mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used  
52837        (via *mmap()*) to establish the mapping which contains *addr*. If that descriptor was closed since  
52838        the mapping was established, the returned value of *fildes* shall be  $-1$ . The *len* argument specifies  
52839        the length of the block of the memory object the user wishes the offset for; upon return, the  
52840        value pointed to by *contig\_len* shall equal either *len*, or the length of the largest contiguous block  
52841        of the memory object that is currently mapped to the calling process starting at *addr*, whichever  
52842        is smaller.

52843        If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig\_len*  
52844        values obtained by calling *posix\_mem\_offset()* are used in a call to *mmap()* with a file descriptor  
52845        that refers to the same memory pool as *fildes* (either through the same port or through a different  
52846        port), and that was opened with neither the `POSIX_TYPED_MEM_ALLOCATE` nor the  
52847        `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag, the typed memory area that is mapped shall  
52848        be exactly the same area that was mapped at *addr* in the address space of the process that called  
52849        *posix\_mem\_offset()*.

52850        If the memory object specified by *fildes* is not a typed memory object, then the behavior of this  
52851        function is implementation-defined.

52852 **RETURN VALUE**

52853        Upon successful completion, the *posix\_mem\_offset()* function shall return zero; otherwise, the  
52854        corresponding error status value shall be returned.

52855 **ERRORS**

52856        The *posix\_mem\_offset()* function shall fail if:

52857        [EACCES]        The process has not mapped a memory object supported by this function at  
52858        the given address *addr*.

52859        This function shall not return an error code of [EINTR].

52860 **EXAMPLES**

52861        None.

52862 **APPLICATION USAGE**

52863        None.

52864 **RATIONALE**

52865        None.

52866 **FUTURE DIRECTIONS**

52867 None.

52868 **SEE ALSO**

52869 *mmap()*, *posix\_typed\_mem\_open()*

52870 XBD <[sys/mman.h](#)>

52871 **CHANGE HISTORY**

52872 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

52873 **NAME**52874 posix\_memalign — aligned memory allocation (**ADVANCED REALTIME**)52875 **SYNOPSIS**

```
52876 ADV #include <stdlib.h>
52877 int posix_memalign(void **memptr, size_t alignment, size_t size);
```

52878 **DESCRIPTION**

52879 The *posix\_memalign()* function shall allocate *size* bytes aligned on a boundary specified by  
 52880 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*  
 52881 shall be a power of two multiple of *sizeof(void \*)*.

52882 Upon successful completion, the value pointed to by *memptr* shall be a multiple of *alignment*.

52883 If the size of the space requested is 0, the behavior is implementation-defined: either a null  
 52884 pointer shall be returned in *memptr*, or the behavior shall be as if the size were some non-zero  
 52885 value, except that the behavior is undefined if the the value returned in *memptr* is used to access  
 52886 an object.

52887 CX The *free()* function shall deallocate memory that has previously been allocated by  
 52888 *posix\_memalign()*.

52889 For purposes of determining the existence of a data race, *posix\_memalign()* shall behave as  
 52890 though it accessed only memory locations accessible through its argument and not other static  
 52891 duration storage. The function may, however, visibly modify the storage that it allocates. Calls  
 52892 to *aligned\_alloc()*, *calloc()*, *free()*, *malloc()*, *posix\_memalign()*, *reallocarray()*, and *realloc()* that  
 52893 allocate or deallocate a particular region of memory shall occur in a single total order (see  
 52894 [Section 4.15.1](#), on page 100), and each such deallocation call shall synchronize with the next  
 52895 allocation (if any) in this order.

52896 **RETURN VALUE**

52897 Upon successful completion, *posix\_memalign()* shall return zero; otherwise, an error number  
 52898 shall be returned to indicate the error and the contents of *memptr* shall either be left unmodified  
 52899 or be set to a null pointer.

52900 If *size* is 0, either:

- 52901 • *posix\_memalign()* shall not attempt to allocate any space, in which case either an  
 52902 implementation-defined error number shall be returned, or zero shall be returned with a  
 52903 null pointer returned in *memptr*, or
- 52904 • *posix\_memalign()* shall attempt to allocate some space and, if the allocation succeeds, zero  
 52905 shall be returned and a pointer to the allocated space shall be returned in *memptr*. The  
 52906 application shall ensure that the pointer is not used to access an object.

52907 **ERRORS**

52908 The *posix\_memalign()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 52909 | [EINVAL] | The value of the alignment parameter is not a power of two multiple of |
| 52910 |          | <i>sizeof(void *)</i> .  |
| 52911 | [ENOMEM] | There is insufficient memory available with the requested alignment.   |



**52912 EXAMPLES**

52913 The following example shows how applications can obtain consistent behavior on error by  
52914 setting *\*memptr* to be a null pointer before calling *posix\_memalign()*.

```
52915 void *ptr = NULL;
52916 ...
52917 //do some work, which might goto error
52918 if (posix_memalign(&ptr, align, size))
52919     goto error;
52920
52921 //do some more work, which might goto error
52922 ...
52923 error:
52924     free(ptr);
52925     //more cleanup;
```

**52925 APPLICATION USAGE**

52926 The *posix\_memalign()* function is part of the Advisory Information option and need not be  
52927 provided on all implementations.

**52928 RATIONALE**

52929 None.

**52930 FUTURE DIRECTIONS**

52931 None.

**52932 SEE ALSO**

52933 [aligned\\_alloc\(\)](#), [free\(\)](#), [malloc\(\)](#)

52934 XBD <[stdlib.h](#)>

**52935 CHANGE HISTORY**

52936 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

52937 In the SYNOPSIS, the inclusion of <[sys/types.h](#)> is no longer required.

**52938 Issue 7**

52939 Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment*  
52940 argument in the DESCRIPTION.

52941 Austin Group Interpretation 1003.1-2001 #152 is applied, clarifying the behavior when the size of  
52942 the space requested is 0.

52943 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0251 [526] and XSH/TC2-2008/0252  
52944 [520,526] are applied.

**52945 Issue 8**

52946 Austin Group Defect 1302 is applied, aligning this function with the requirements of the  
52947 ISO/IEC 9899:2018 standard on other memory allocation functions.

52948 **NAME**

52949        posix\_openpt — open a pseudo-terminal device

52950 **SYNOPSIS**

```
52951 XSI      #include <stdlib.h>
52952         int posix_openpt(int oflag);
```

52953 **DESCRIPTION**

52954        The *posix\_openpt()* function shall establish a connection between a manager device for a pseudo-terminal and a file descriptor. The file descriptor shall be allocated as described in [Section 2.6](#) (on page 525) and can be used by other I/O functions that refer to that pseudo-terminal.

52957        The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

52959        Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list:

52960	O_RDWR	Open for reading and writing.
52961	O_CLOEXEC	Atomically set the FD_CLOEXEC flag on the file descriptor.
52962	O_CLOFORK	Atomically set the FD_CLOFORK flag on the file descriptor.
52963	O_NOCTTY	If set <i>posix_openpt()</i> shall not cause the terminal device to become the controlling terminal for the process.

52965        The behavior of other values for the *oflag* argument is unspecified.

52966 **RETURN VALUE**

52967        Upon successful completion, the *posix\_openpt()* function shall open a file descriptor for a manager pseudo-terminal device and return a non-negative integer representing the file descriptor. Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

52970 **ERRORS**

52971        The *posix\_openpt()* function shall fail if:

52972	[EMFILE]	All file descriptors available to the process are currently open.
52973	[ENFILE]	The maximum allowable number of files is currently open in the system.

52974        The *posix\_openpt()* function may fail if:

52975	[EINVAL]	The value of <i>oflag</i> is not valid.
52976	[EAGAIN]	Out of pseudo-terminal resources.

52977 **EXAMPLES**

52978        **Opening a Pseudo-Terminal and Returning the Name of the Subsidiary Device and a File Descriptor**

```
52980     #include <fcntl.h>
52981     #include <stdio.h>
52982     #include <stdlib.h>
52983
52983     int managerfd, subsidiaryfd;
52984     char *subsidiarydevice;
52985
52985     managerfd = posix_openpt (O_RDWR | O_NOCTTY);
52986
52986     if (managerfd == -1
52987         || grantpt (managerfd) == -1
```

```

52988     || unlockpt (managerfd) == -1
52989     || (subsidiarydevice = ptsname (managerfd)) == NULL)
52990     return -1;

52991     printf("subsidiary device is: %s\n", subsidiarydevice);

52992     subsidiaryfd = open(subsidiarydevice, O_RDWR|O_NOCTTY);
52993     if (subsidiaryfd < 0)
52994         return -1;

```

#### 52995 APPLICATION USAGE

52996 This function is a method for portably obtaining a file descriptor of a manager terminal device  
52997 for a pseudo-terminal. The *grantpt()* function and the *ptsname()* and *ptsname\_r()* functions can  
52998 be used to manipulate mode and ownership permissions, and to obtain the name of the  
52999 subsidiary device, respectively.

#### 53000 RATIONALE

53001 The standard developers considered the matter of adding a special device for cloning manager  
53002 pseudo-terminal devices: the */dev/ptmx* device. However, consensus could not be reached, and  
53003 it was felt that adding a new function would permit other implementations. The *posix\_openpt()*  
53004 function is designed to complement the *grantpt()*, *ptsname()*, *ptsname\_r()*, and *unlockpt()*  
53005 functions.

53006 On implementations supporting the */dev/ptmx* clone device, opening the manager device of a  
53007 pseudo-terminal is simply:

```

53008 mfdp = open("/dev/ptmx", oflag );
53009 if (mfdp < 0)
53010     return -1;

```

53011 The *O\_CLOEXEC* and *O\_CLOFORK* flags are necessary to avoid a data race in multi-threaded  
53012 applications. Without *O\_CLOFORK*, a file descriptor is leaked into a child process created by  
53013 one thread in the window between another thread creating a file descriptor with *posix\_openpt()*  
53014 and then using *fcntl()* to set the *FD\_CLOFORK* flag. Without *O\_CLOEXEC*, a file descriptor  
53015 intentionally inherited by child processes is similarly leaked into an executed program if  
53016 *FD\_CLOEXEC* is not set atomically.

#### 53017 FUTURE DIRECTIONS

53018 None.

#### 53019 SEE ALSO

53020 [Section 2.6](#) (on page 525), *grantpt()*, *open()*, *ptsname()*, *unlockpt()*

53021 XBD [<fcntl.h>](#), [<stdlib.h>](#)

#### 53022 CHANGE HISTORY

53023 First released in Issue 6.

#### 53024 Issue 7

53025 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

53026 SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.

53027 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0253 [835] and XSH/TC2-2008/0254  
53028 [835] are applied.

53029 **Issue 8**

- 53030 Austin Group Defects 411 and 1318 are applied, adding O\_CLOEXEC and O\_CLOFORK.
- 53031 Austin Group Defect 508 is applied, changing the APPLICATION USAGE and RATIONALE  
53032 sections to refer to *ptsname\_r()* as well as *ptsname()*.
- 53033 Austin Group Defect 593 is applied, removing `#include <fcntl.h>` from the SYNOPSIS and  
53034 a reference to `<fcntl.h>` from the DESCRIPTION section.
- 53035 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 53036 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal  
53037 devices.

53038 **NAME**53039 posix\_spawn, posix\_spawnnp — spawn a process (**ADVANCED REALTIME**)53040 **SYNOPSIS**

```
53041 SPN      #include <spawn.h>
53042
53043 int posix_spawn(pid_t *restrict pid, const char *restrict path,
53044               const posix_spawn_file_actions_t *restrict file_actions,
53045               const posix_spawnattr_t *restrict attrp,
53046               char *const argv[restrict], char *const envp[restrict]);
53047 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
53048                  const posix_spawn_file_actions_t *restrict file_actions,
53049                  const posix_spawnattr_t *restrict attrp,
53050                  char *const argv[restrict], char *const envp[restrict]);
```

53050 **DESCRIPTION**

53051 The *posix\_spawn()* and *posix\_spawnnp()* functions shall create a new process (child process) from  
 53052 the specified process image. The new process image shall be constructed from a regular  
 53053 executable file called the new process image file.

53054 When a C program is executed as the result of this call, it shall be entered as a C-language  
 53055 function call as follows:

```
53056 int main(int argc, char *argv[]);
```

53057 where *argc* is the argument count and *argv* is an array of character pointers to the arguments  
 53058 themselves. In addition, the following variable:

```
53059 extern char **environ;
```

53060 shall be initialized as a pointer to an array of character pointers to the environment strings.

53061 The argument *argv* is an array of character pointers to null-terminated strings. The last member  
 53062 of this array shall be a null pointer and is not counted in *argc*. These strings constitute the  
 53063 argument list available to the new process image. The value in *argv*[0] should point to a filename  
 53064 string that is associated with the process image being started by the *posix\_spawn()* or  
 53065 *posix\_spawnnp()* function.

53066 The argument *envp* is an array of character pointers to null-terminated strings. These strings  
 53067 constitute the environment for the new process image. The environment array is terminated by a  
 53068 null pointer.

53069 The number of bytes available for the combined argument and environment lists of the child  
 53070 process is {ARG\_MAX}. The implementation shall specify in the system documentation (see  
 53071 XBD Chapter 2, on page 15) whether any list overhead, such as length words, null terminators,  
 53072 pointers, or alignment bytes, is included in this total.

53073 The *path* argument to *posix\_spawn()* is a pathname that identifies the new process image file to  
 53074 execute; if the pathname does not start with a <slash>, it shall be interpreted relative to the  
 53075 working directory of the child process after all *file\_actions* have been performed.

53076 The *file* parameter to *posix\_spawnnp()* shall be used to construct a pathname that identifies the  
 53077 new process image file. If the *file* parameter contains a <slash> character, the *file* parameter shall  
 53078 be used as the pathname for the new process image file. Otherwise, the path prefix for this file  
 53079 shall be obtained by a search of the directories passed as the environment variable *PATH* (see  
 53080 XBD Chapter 8, on page 167), using the working directory of the child process after all  
 53081 *file\_actions* have been performed. If this environment variable is not defined, the results of the  
 53082 search are implementation-defined. However, if at least one of the *exec* family of functions

53083 would fail with [ENOEXEC] because the process image contents are not executable, this shall  
 53084 cause *posix\_spawnnp()* to fail rather than attempting a fallback to invoking the process as a shell  
 53085 script passed to *sh*.

53086 If *file\_actions* is a null pointer, then file descriptors open in the calling process shall remain open  
 53087 in the child process, except for those whose FD\_CLOEXEC or FD\_CLOFORK flag is set (see  
 53088 *fcntl()*), and except for file descriptors that are closed by a fork handler (if fork handlers are  
 53089 called). For those file descriptors that remain open, the child process shall not inherit any  
 53090 process-owned file locks, but all remaining attributes of the corresponding open file descriptions  
 53091 (see *fcntl()*), shall remain unchanged. The current working directory of the child process shall be  
 53092 the same as it is in the parent process.

53093 If *file\_actions* is not a null pointer, then the file descriptors open in the child process shall be those  
 53094 open in the calling process as modified by FD\_CLOFORK file descriptor flags, fork handlers (if  
 53095 they are called), the spawn file actions object pointed to by *file\_actions*, and the FD\_CLOEXEC  
 53096 flag of each remaining open file descriptor after the spawn file actions have been processed. The  
 53097 effective order of processing the spawn file actions shall be:

- 53098 1. The set of open file descriptors for the child process shall initially be the same set as is  
 53099 open for the calling process, except for those that have the FD\_CLOFORK flag set and any  
 53100 that are closed by fork handlers (if they are called). The child process shall not inherit any  
 53101 file locks, but all remaining attributes of the corresponding open file descriptions (see  
 53102 *fcntl()*), shall remain unchanged.
- 53103 2. The signal mask, signal default actions, and the effective user and group IDs for the child  
 53104 process shall be changed as specified in the attributes object referenced by *attrp*.
- 53105 3. The file actions specified by the spawn file actions object shall be performed in the order  
 53106 in which they were added to the spawn file actions object, and relative pathnames in a  
 53107 given action shall be interpreted in relation to the tracked working directory. The tracked  
 53108 working directory shall begin with the current working directory of the parent process,  
 53109 and can be altered according to *chdir* or *fchdir* file actions; the current working directory of  
 53110 the child process shall be the final state of the tracked working directory after all file  
 53111 actions have been applied.
- 53112 4. Any file descriptor that has its FD\_CLOEXEC flag set shall be closed.

53113 If file descriptor 0, 1, or 2 would otherwise be closed in the new process image created by  
 53114 *posix\_spawn()* or *posix\_spawnnp()*, implementations may open an unspecified file for the file  
 53115 descriptor in the new process image. If a standard utility or a conforming application is executed  
 53116 with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the  
 53117 environment in which the utility or application is executed shall be deemed non-conforming,  
 53118 and consequently the utility or application might not behave as described in this standard.

53119 The **posix\_spawnattr\_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at  
 53120 least the attributes defined below.

53121 If the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn-flags* attribute of the object referenced  
 53122 by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the child's process  
 53123 group shall be as specified in the *spawn-pgroup* attribute of the object referenced by *attrp*.

53124 As a special case, if the POSIX\_SPAWN\_SETPGROUP flag is set in the *spawn-flags* attribute of  
 53125 the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero,  
 53126 then the child shall be in a new process group with a process group ID equal to its process ID.

53127 If the POSIX\_SPAWN\_SETSID flag is set in the *spawn-flags* attribute of the object referenced by  
 53128 *attrp*, the child process shall be the session leader of a new session, shall be the process group

53129 leader of a new process group, and shall have no controlling terminal. The process group ID of  
 53130 the child process shall be set equal to the process ID of the child process. The child process shall  
 53131 be the only process in the new process group and the only process in the new session.

53132 If both the POSIX\_SPAWN\_SETPGROUP flag and the POSIX\_SPAWN\_SETSID flag are set in the  
 53133 *spawn-flags* attribute of the object referenced by *attrp*, the behavior is unspecified.

53134 If neither the POSIX\_SPAWN\_SETPGROUP flag nor the POSIX\_SPAWN\_SETSID flag is set in  
 53135 the *spawn-flags* attribute of the object referenced by *attrp*, the new child process shall inherit the  
 53136 parent's process group.

53137 PS If the POSIX\_SPAWN\_SETSCHEDPARAM flag is set in the *spawn-flags* attribute of the object  
 53138 referenced by *attrp*, but POSIX\_SPAWN\_SETSCHEDULER is not set, the new process image  
 53139 shall initially have the scheduling policy of the calling process with the scheduling parameters  
 53140 specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

53141 If the POSIX\_SPAWN\_SETSCHEDULER flag is set in the *spawn-flags* attribute of the object  
 53142 referenced by *attrp* (regardless of the setting of the POSIX\_SPAWN\_SETSCHEDPARAM flag),  
 53143 the new process image shall initially have the scheduling policy specified in the *spawn-*  
 53144 *schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in  
 53145 the *spawn-schedparam* attribute of the same object.

53146 The POSIX\_SPAWN\_RESETPID flag in the *spawn-flags* attribute of the object referenced by *attrp*  
 53147 governs the effective user ID of the child process. If this flag is not set, the child process shall  
 53148 inherit the effective user ID of the parent process. If this flag is set, the effective user ID of the  
 53149 child process shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit  
 53150 of the new process image file is set, the effective user ID of the child process shall become that  
 53151 file's owner ID before the new process image begins execution.

53152 The POSIX\_SPAWN\_RESETPID flag in the *spawn-flags* attribute of the object referenced by *attrp*  
 53153 also governs the effective group ID of the child process. If this flag is not set, the child process  
 53154 shall inherit the effective group ID of the parent process. If this flag is set, the effective group ID  
 53155 of the child process shall be reset to the parent's real group ID. In either case, if the set-group-ID  
 53156 mode bit of the new process image file is set, the effective group ID of the child process shall  
 53157 become that file's group ID before the new process image begins execution.

53158 If the POSIX\_SPAWN\_SETSIGMASK flag is set in the *spawn-flags* attribute of the object  
 53159 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*  
 53160 *sigmask* attribute of the object referenced by *attrp*.

53161 If the POSIX\_SPAWN\_SETSIGDEF flag is set in the *spawn-flags* attribute of the object referenced  
 53162 by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to  
 53163 their default actions in the child process. Signals set to the default action in the parent process  
 53164 shall be set to the default action in the child process.

53165 Signals set to be caught by the calling process shall be set to the default action in the child  
 53166 process.

53167 Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be  
 53168 ignored by the child process, unless otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag  
 53169 being set in the *spawn-flags* attribute of the object referenced by *attrp* and the signals being  
 53170 indicated in the *spawn-sigdefault* attribute of the object referenced by *attrp*.

53171 If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the  
 53172 SIGCHLD signal is set to be ignored or to the default action in the child process, unless  
 53173 otherwise specified by the POSIX\_SPAWN\_SETSIGDEF flag being set in the *spawn-flags*  
 53174 attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the

53175 *spawn\_sigdefault* attribute of the object referenced by *attrp*.

53176 If the value of the *attrp* pointer is a null pointer, then the default values are used.

53177 All process attributes, other than those influenced by the attributes set in the object referenced  
 53178 by *attrp* as specified above or by the file descriptor manipulations specified in *file\_actions*, shall  
 53179 appear in the new process image as though *fork()* had been called to create a child process and  
 53180 then a member of the *exec* family of functions had been called by the child process to execute the  
 53181 new process image.

53182 It is implementation-defined whether the fork handlers are run when *posix\_spawn()* or  
 53183 *posix\_spawnp()* is called.

53184 **RETURN VALUE**

53185 Upon successful completion, *posix\_spawn()* and *posix\_spawnp()* shall return the process ID of the  
 53186 child process to the parent process, in the variable pointed to by a non-null *pid* argument, and  
 53187 shall return zero as the function return value. Otherwise, no child process shall be created, the  
 53188 value stored into the variable pointed to by a non-null *pid* is unspecified, and an error number  
 53189 shall be returned as the function return value to indicate the error. If the *pid* argument is a null  
 53190 pointer, the process ID of the child is not returned to the caller.

53191 **ERRORS**

53192 These functions may fail if:

53193 [EINVAL] The value specified by *file\_actions* or *attrp* is invalid.

53194 If this error occurs after the calling process successfully returns from the *posix\_spawn()* or  
 53195 *posix\_spawnp()* function, the child process may exit with exit status 127.

53196 If *posix\_spawn()* or *posix\_spawnp()* fail for any of the reasons that would cause *fork()* or one of  
 53197 the *exec* family of functions to fail, including when the corresponding *exec* function would  
 53198 attempt a fallback to *sh* instead of failing with [ENOEXEC], an error value shall be returned as  
 53199 described by *fork()* and *exec*, respectively; or, if the error occurs after the calling process  
 53200 successfully returns, the child process shall exit with exit status 127.

53201 If POSIX\_SPAWN\_SETPGROUP is set in the *spawn\_flags* attribute of the object referenced by  
 53202 *attrp*, and *posix\_spawn()* or *posix\_spawnp()* fails while changing the child's process group, an  
 53203 error value shall be returned as described by *setpgid()*; or, if the error occurs after the calling  
 53204 process successfully returns, the child process shall exit with exit status 127.

53205 If POSIX\_SPAWN\_SETSID is set in the *spawn\_flags* attribute of the object referenced by *attrp*, and  
 53206 *posix\_spawn()* or *posix\_spawnp()* fails while creating the new session, changing the child's  
 53207 session ID, or changing the child's process group, an error value shall be returned as described  
 53208 by *setsid()*; or, if the error occurs after the calling process successfully returns, the child process  
 53209 shall exit with exit status 127.

53210 PS If POSIX\_SPAWN\_SETSCHEDPARAM is set and POSIX\_SPAWN\_SETSCHEDULER is not set in  
 53211 the *spawn\_flags* attribute of the object referenced by *attrp*, then if *posix\_spawn()* or *posix\_spawnp()*  
 53212 fails for any of the reasons that would cause *sched\_setparam()* to fail, an error value shall be  
 53213 returned as described by *sched\_setparam()*; or, if the error occurs after the calling process  
 53214 successfully returns, the child process shall exit with exit status 127.

53215 If POSIX\_SPAWN\_SETSCHEDULER is set in the *spawn\_flags* attribute of the object referenced by  
 53216 *attrp*, and if *posix\_spawn()* or *posix\_spawnp()* fails for any of the reasons that would cause  
 53217 *sched\_setscheduler()* to fail, an error value shall be returned as described by *sched\_setscheduler()*;  
 53218 or, if the error occurs after the calling process successfully returns, the child process shall exit  
 53219 with exit status 127.



53220 If the *file\_actions* argument is not a null pointer, and specifies any *chdir*, *close*, *dup2*, *fchdir*, or *open*  
 53221 actions to be performed, and if *posix\_spawn()* or *posix\_spawnnp()* fails for any of the reasons that  
 53222 would cause *chdir()*, *close()*, *dup2()*, *fchdir()*, or *open()* to fail, other than attempting a *close()* on  
 53223 a file descriptor that is in range but already closed, an error value shall be returned as described  
 53224 by *chdir()*, *close()*, *dup2()*, *fchdir()*, and *open()*, respectively; or, if the error occurs after the  
 53225 calling process successfully returns, the child process shall exit with exit status 127. An open file  
 53226 action may, by itself, result in any of the errors described by *close()* or *dup2()*, in addition to  
 53227 those described by *open()*.

#### 53228 EXAMPLES

53229 None.

#### 53230 APPLICATION USAGE

53231 These functions are part of the Spawn option and need not be provided on all implementations.

53232 See also the APPLICATION USAGE section for *exec*.

#### 53233 RATIONALE

53234 The *posix\_spawn()* function and its close relation *posix\_spawnnp()* have been introduced to  
 53235 overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or  
 53236 impossible to implement without swapping or dynamic address translation.

- 53237 • Swapping is generally too slow for a realtime environment.
- 53238 • Dynamic address translation is not available everywhere that POSIX might be useful.
- 53239 • Processes are too useful to simply option out of POSIX whenever it must run without  
 53240 address translation or other MMU services.

53241 Thus, POSIX needs process creation and file execution primitives that can be efficiently  
 53242 implemented without address translation or other MMU services.

53243 The *posix\_spawn()* function is implementable as a library routine, but both *posix\_spawn()* and  
 53244 *posix\_spawnnp()* are designed as kernel operations. Also, although they may be an efficient  
 53245 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation  
 53246 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for  
 53247 *fork()/exec*.

53248 This view of the role of *posix\_spawn()* and *posix\_spawnnp()* influenced the design of their API. It  
 53249 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified  
 53250 operations of any sort are permitted between the creation of the child process and the execution  
 53251 of the new process image; any attempt to reach that level would need to provide a programming  
 53252 language as parameters. Instead, *posix\_spawn()* and *posix\_spawnnp()* are process creation  
 53253 primitives like the *Start\_Process* and *Start\_Process\_Search* Ada language bindings package  
 53254 *POSIX\_Process\_Primitives* and also like those in many operating systems that are not UNIX  
 53255 systems, but with some POSIX-specific additions.

53256 To achieve its coverage goals, *posix\_spawn()* and *posix\_spawnnp()* have control of seven types of  
 53257 inheritance: file descriptors, current working directory, process group ID, user and group ID,  
 53258 signal mask, scheduling, and whether each signal ignored in the parent will remain ignored in  
 53259 the child, or be reset to its default action in the child.

53260 Control of file descriptors is required to allow an independently written child process image to  
 53261 access data streams opened by and even generated or read by the parent process without being  
 53262 specifically coded to know which parent files and file descriptors are to be used. Control of the  
 53263 current working directory is required because the parent process may want to constrain the  
 53264 resources that the child process can reach from its current working directory or affect how  
 53265 relative pathnames are interpreted, while recognizing that a multi-threaded parent process

53266 would require a lot of overhead to safely change its own working directory prior to creating the  
 53267 child process. Control of the process group ID is required to control how the job control of the  
 53268 child process relates to that of the parent.

53269 Control of the signal mask and signal defaulting is sufficient to support the implementation of  
 53270 *system()*. Although support for *system()* is not explicitly one of the goals for *posix\_spawn()* and  
 53271 *posix\_spawnnp()*, it is covered under the “at least 50%” coverage goal.

53272 The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of  
 53273 the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec*  
 53274 family of functions should fully specify open file inheritance. The implementation need make no  
 53275 decisions regarding the set of open file descriptors when the child process image begins  
 53276 execution, those decisions having already been made by the caller and expressed as the set of  
 53277 open file descriptors and their FD\_CLOEXEC and FD\_CLOFORK flags at the time of the call, the  
 53278 actions of fork handlers (if they are called), and the spawn file actions object specified in the call.  
 53279 We have been assured that in cases where the POSIX *Start\_Process* Ada primitives have been  
 53280 implemented in a library, this method of controlling file descriptor inheritance may be  
 53281 implemented very easily.

53282 We can identify several problems with *posix\_spawn()* and *posix\_spawnnp()*, but there does not  
 53283 appear to be a solution that introduces fewer problems. Environment modification for child  
 53284 process attributes not specifiable via the *attrp* or *file\_actions* arguments must be done in the  
 53285 parent process, and since the parent generally wants to save its context, it is more costly than  
 53286 similar functionality with *fork()/exec*. It is also complicated to modify the environment of a  
 53287 multi-threaded process temporarily, since all threads must agree when it is safe for the  
 53288 environment to be changed. However, this cost is only borne by those invocations of  
 53289 *posix\_spawn()* and *posix\_spawnnp()* that use the additional functionality. Since extensive  
 53290 modifications are not the usual case, and are particularly unlikely in time-critical code, keeping  
 53291 much of the environment control out of *posix\_spawn()* and *posix\_spawnnp()* is appropriate design.

53292 The *posix\_spawn()* and *posix\_spawnnp()* functions do not have all the power of *fork()/exec*. This is  
 53293 to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to  
 53294 duplicate its functionality in a simple, fast function with no special hardware requirements. It is  
 53295 worth noting that *posix\_spawn()* and *posix\_spawnnp()* are very similar to the process creation  
 53296 operations on many operating systems that are not UNIX systems.

## 53297 Requirements

53298 The requirements for *posix\_spawn()* and *posix\_spawnnp()* are:

- 53299 • They must be implementable without an MMU or unusual hardware.
- 53300 • They must be compatible with existing POSIX standards.

53301 Additional goals are:

- 53302 • They should be efficiently implementable.
- 53303 • They should be able to replace at least 50% of typical executions of *fork()*.
- 53304 • A system with *posix\_spawn()* and *posix\_spawnnp()* and without *fork()* should be useful, at  
 53305 least for realtime applications.
- 53306 • A system with *fork()* and the *exec* family should be able to implement *posix\_spawn()* and  
 53307 *posix\_spawnnp()* as library routines.

53308 **Two-Syntax**

53309 POSIX *exec* has several calling sequences with approximately the same functionality. These  
 53310 appear to be required for compatibility with existing practice. Since the existing practice for the  
 53311 *posix\_spawn\**(*)* functions is otherwise substantially unlike POSIX, we feel that simplicity  
 53312 outweighs compatibility. There are, therefore, only two names for the *posix\_spawn\**(*)* functions.

53313 The parameter list does not differ between *posix\_spawn()* and *posix\_spawnp()*; *posix\_spawnp()*  
 53314 interprets the second parameter more elaborately than *posix\_spawn()*.

53315 **Compatibility with POSIX.5 (Ada)**

53316 The *Start\_Process* and *Start\_Process\_Search* procedures from the *POSIX\_Process\_Primitives*  
 53317 package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a  
 53318 manner similar to that of *posix\_spawn()* and *posix\_spawnp()*. Originally, in keeping with our  
 53319 simplicity goal, the standard developers had limited the capabilities of *posix\_spawn()* and  
 53320 *posix\_spawnp()* to a subset of the capabilities of *Start\_Process* and *Start\_Process\_Search*; certain  
 53321 non-default capabilities were not supported. However, based on suggestions by the ballot group  
 53322 to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings  
 53323 working group member, the standard developers decided that *posix\_spawn()* and *posix\_spawnp()*  
 53324 should be sufficiently powerful to implement *Start\_Process* and *Start\_Process\_Search*. The  
 53325 rationale is that if the Ada language binding to such a primitive had already been approved as  
 53326 an IEEE standard, there can be little justification for not approving the functionally-equivalent  
 53327 parts of a C binding. The only three capabilities provided by *posix\_spawn()* and *posix\_spawnp()*  
 53328 that are not provided by *Start\_Process* and *Start\_Process\_Search* are optionally specifying the  
 53329 child's process group ID, the set of signals to be reset to default signal handling in the child  
 53330 process, and the child's scheduling policy and parameters.

53331 For the Ada language binding for *Start\_Process* to be implemented with *posix\_spawn()*, that  
 53332 binding would need to explicitly pass an empty signal mask and the parent's environment to  
 53333 *posix\_spawn()* whenever the caller of *Start\_Process* allowed these arguments to default, since  
 53334 *posix\_spawn()* does not provide such defaults. The ability of *Start\_Process* to mask user-specified  
 53335 signals during its execution is functionally unique to the Ada language binding and must be  
 53336 dealt with in the binding separately from the call to *posix\_spawn()*.

53337 **Process Group**

53338 The process group inheritance field can be used to join the child process with an existing process  
 53339 group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*,  
 53340 the *setpgid()* mechanism will place the child process in a new process group.

53341 **Threads**

53342 Without the *posix\_spawn()* and *posix\_spawnp()* functions, systems without address translation  
 53343 can still use threads to give an abstraction of concurrency. In many cases, thread creation  
 53344 suffices, but it is not always a good substitute. The *posix\_spawn()* and *posix\_spawnp()* functions  
 53345 are considerably "heavier" than thread creation. Processes have several important attributes that  
 53346 threads do not. Even without address translation, a process may have base-and-bound memory  
 53347 protection. Each process has a process environment including security attributes and file  
 53348 capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-  
 53349 uniform-memory-architecture multi-processors better than threads, and they are more  
 53350 convenient to use for activities that are not closely linked.

53351 The *posix\_spawn()* and *posix\_spawnp()* functions may not bring support for multiple processes to  
 53352 every configuration. Process creation is not the only piece of operating system support required

53353 to support multiple processes. The total cost of support for multiple processes may be quite high  
53354 in some circumstances. Existing practice shows that support for multiple processes is  
53355 uncommon and threads are common among “tiny kernels”. There should, therefore, probably  
53356 continue to be AEPs for operating systems with only one process.

#### 53357 **Asynchronous Error Notification**

53358 A library implementation of *posix\_spawn()* or *posix\_spawnp()* may not be able to detect all  
53359 possible errors before it forks the child process. POSIX.1-2024 provides for an error indication  
53360 returned from a child process which could not successfully complete the spawn operation via a  
53361 special exit status which may be detected using the status value returned by *wait()*, *waitid()*, and  
53362 *waitpid()*.

53363 The *stat\_val* interface and the macros used to interpret it are not well suited to the purpose of  
53364 returning API errors, but they are the only path available to a library implementation. Thus, an  
53365 implementation may cause the child process to exit with exit status 127 for any error detected  
53366 during the spawn process after the *posix\_spawn()* or *posix\_spawnp()* function has successfully  
53367 returned.

53368 The standard developers had proposed using two additional macros to interpret *stat\_val*. The  
53369 first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because  
53370 of an error detected during the *posix\_spawn()* or *posix\_spawnp()* operations rather than during  
53371 actual execution of the child process image; the second, WSPAWNERRNO, would have  
53372 extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group  
53373 strongly opposed this because it would make a library implementation of *posix\_spawn()* or  
53374 *posix\_spawnp()* dependent on kernel modifications to *waitpid()* to be able to embed special  
53375 information in *stat\_val* to indicate a spawn failure.

53376 The 8 bits of child process exit status that are guaranteed by POSIX.1-2024 to be accessible to the  
53377 waiting parent process are insufficient to disambiguate a spawn error from any other kind of  
53378 error that may be returned by an arbitrary process image. No other bits of the exit status are  
53379 required to be visible in *stat\_val*, so these macros could not be strictly implemented at the library  
53380 level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this  
53381 value by *system()* and *popen()* to signal failures in these operations that occur after the function  
53382 has returned but before a shell is able to execute. The exit status of 127 does not uniquely  
53383 identify this class of error, nor does it provide any detailed information on the nature of the  
53384 failure. Note that a kernel implementation of *posix\_spawn()* or *posix\_spawnp()* is permitted (and  
53385 encouraged) to return any possible error as the function value, thus providing more detailed  
53386 failure information to the parent process.

53387 Thus, no special macros are available to isolate asynchronous *posix\_spawn()* or *posix\_spawnp()*  
53388 errors. Instead, errors detected by the *posix\_spawn()* or *posix\_spawnp()* operations in the context  
53389 of the child process before the new process image executes are reported by setting the child’s exit  
53390 status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the  
53391 *stat\_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that  
53392 other status values with which the child process image may exit (before the parent can  
53393 conclusively determine that the child process image has begun execution) are distinct from exit  
53394 status 127.

#### 53395 **FUTURE DIRECTIONS**

53396 None.

53397 **SEE ALSO**

53398 *alarm()*, *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *fstatat()*, *kill()*, *open()*,  
 53399 *posix\_spawn\_file\_actions\_addchdir()*, *posix\_spawn\_file\_actions\_addclose()*,  
 53400 *posix\_spawn\_file\_actions\_adddup2()*, *posix\_spawn\_file\_actions\_destroy()*, *posix\_spawnattr\_destroy()*,  
 53401 *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*,  
 53402 *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*,  
 53403 *sched\_setparam()*, *sched\_setscheduler()*, *setpgid()*, *setsid()*, *setuid()*, *times()*, *wait()*, *waitid()*

53404 XBD Chapter 8 (on page 167), [<spawn.h>](#)

53405 **CHANGE HISTORY**

53406 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53407 IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are  
 53408 changed as well as the signal mask in step 2.

53409 IEEE PASC Interpretation 1003.1 #132 is applied.

53410 **Issue 7**

53411 Functionality relating to the Threads option is moved to the Base.

53412 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0433 [291], XSH/TC1-2008/0434 [173],  
 53413 and XSH/TC1-2008/0435 [173] are applied.

53414 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0255 [824] is applied.

53415 **Issue 8**

53416 Austin Group Defect 370 is applied, requiring that attempting to close a file descriptor that is in  
 53417 range but already closed is not treated as an error.

53418 Austin Group Defect 768 is applied, adding OFD-owned file locks.

53419 Austin Group Defect 1044 is applied, adding POSIX\_SPAWN\_SETSID.

53420 Austin Group Defect 1208 is applied, adding *posix\_spawn\_file\_actions\_addchdir()* and  
 53421 *posix\_spawn\_file\_actions\_addfchdir()*.

53422 Austin Group Defect 1318 is applied, adding FD\_CLOFORK and clarifying that the inherited set  
 53423 of file descriptors is affected by the actions of fork handlers (if they are called).

53424 Austin Group Defect 1328 is applied, adding the **restrict** keyword to the third parameter of  
 53425 *posix\_spawn()* and *posix\_spawnnp()*.

53426 Austin Group Defect 1362 is applied, removing parentheses around some text intended to be  
 53427 normative.

53428 Austin Group Defect 1674 is applied, adding a requirement that *posix\_spawnnp()* does not fallback  
 53429 to executing *sh* when the corresponding *exec* function would do so instead of failing with  
 53430 [ENOEXEC].

53431 **NAME**

53432 posix\_spawn\_file\_actions\_addchdir, posix\_spawn\_file\_actions\_addfchdir — add chdir or fchdir  
53433 action to spawn file actions object (**ADVANCED REALTIME**)

53434 **SYNOPSIS**

```
53435 SPN #include <spawn.h>
53436 int posix_spawn_file_actions_addchdir(posix_spawn_file_actions_t
53437     *restrict file_actions, const char *restrict path);
53438 int posix_spawn_file_actions_addfchdir(posix_spawn_file_actions_t
53439     *file_actions, int fildes);
```

53440 **DESCRIPTION**

53441 The *posix\_spawn\_file\_actions\_addchdir()* function shall add a *chdir* action to the object referenced  
53442 by *file\_actions* that shall cause the working directory to be set to *path* (as if *chdir(path)* had been  
53443 called) when a new process is spawned using this file actions object. A relative *path* shall be  
53444 interpreted in relation to the working directory determined by any prior actions. The string  
53445 pointed to by *path* shall be copied by the *posix\_spawn\_file\_actions\_addchdir()* function.

53446 The *posix\_spawn\_file\_actions\_addfchdir()* function shall add an *fchdir* action to the object  
53447 referenced by *file\_actions* that shall cause the working directory to be set to *fildes* (as if  
53448 *fchdir(fildes)* had been called) when a new process is spawned using this file actions object.

53449 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*.

53450 **RETURN VALUE**

53451 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
53452 be returned to indicate the error.

53453 **ERRORS**

53454 The *posix\_spawn\_file\_actions\_addfchdir()* function shall fail if:  
53455 [EBADF] The value specified by *fildes* is negative.

53456 These functions shall fail if:

53457 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

53458 These functions may fail if:

53459 [EINVAL] The value specified by *file\_actions* is invalid.

53460 It shall not be considered an error for the *path* or *fildes* argument passed to these functions to  
53461 specify a pathname or file descriptor for which the specified operation could not be performed  
53462 at the time of the call. Any such error shall be detected when the associated file actions object is  
53463 later used during a *posix\_spawn()* or *posix\_spawnnp()* operation.

53464 **EXAMPLES**

53465 None.

53466 **APPLICATION USAGE**

53467 The *posix\_spawn\_file\_actions\_addchdir()* and *posix\_spawn\_file\_actions\_addfchdir()* functions are  
53468 part of the Spawn option and need not be provided on all implementations.

53469 Changing the working directory of a child process can be useful when invoking utilities such as  
53470 *pax*. Furthermore, the ability to add *fchdir* actions to *posix\_spawn()* gives the caller as much  
53471 control over relative pathnames processed in the context of the child as it would otherwise have  
53472 using *openat()*, since all file actions are processed in sequence in the context of the child at a  
53473 point where the child process is still single-threaded. Without *chdir* or *fchdir* actions, changing  
53474 the working directory of the child would require a shim utility (some implementations provide

53475 `env -C /new/path program args...`

53476 as an extension, but the standard does not require this extension), or else temporarily changing  
53477 the working directory in the parent process prior to calling `posix_spawn()` (but this requires  
53478 locking in a multi-threaded process, to ensure that no other thread is impacted by the temporary  
53479 change to global state).

53480 File actions are performed in a new process created by `posix_spawn()` or `posix_spawnp()` in the  
53481 same order that they were added to the file actions object. Thus, the execution of an *open* action  
53482 that was created by a call to `posix_spawn_file_actions_addopen()` that specifies a relative path will  
53483 be affected by the execution of a *chdir* or *fchdir* action that was created by a previous call to  
53484 `posix_spawn_file_actions_addchdir()` or `posix_spawn_file_actions_addfchdir()`. Likewise, a relative  
53485 path passed to `posix_spawn()` will be affected by the last *chdir* or *fchdir* action in the file action  
53486 list.

#### 53487 RATIONALE

53488 Refer to the RATIONALE section in `posix_spawn_file_actions_addclose()`.

#### 53489 FUTURE DIRECTIONS

53490 None.

#### 53491 SEE ALSO

53492 `chdir()`, `fchdir()`, `posix_spawn()`, `posix_spawn_file_actions_addclose()`,  
53493 `posix_spawn_file_actions_destroy()`

53494 XBD <spawn.h>

#### 53495 CHANGE HISTORY

53496 First released in Issue 8. Derived from Solaris `posix_spawn_file_actions_addchdir_np`.

53497 **NAME**

53498 posix\_spawn\_file\_actions\_addclose, posix\_spawn\_file\_actions\_addopen — add close or open  
53499 action to spawn file actions object (**ADVANCED REALTIME**)

53500 **SYNOPSIS**

```
53501 SPN #include <spawn.h>
53502 int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
53503     *file_actions, int fildes);
53504 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
53505     *restrict file_actions, int fildes,
53506     const char *restrict path, int oflag, mode_t mode);
```

53507 **DESCRIPTION**

53508 These functions shall add a *close* or *open* action to a spawn file actions object.

53509 A spawn file actions object is of type **posix\_spawn\_file\_actions\_t** (defined in **<spawn.h>**) and is  
53510 used to specify a series of actions to be performed by a *posix\_spawn()* or *posix\_spawnp()*  
53511 operation in order to arrive at the set of open file descriptors for the child process given the set  
53512 of open file descriptors of the parent. POSIX.1-2024 does not define comparison or assignment  
53513 operators for the type **posix\_spawn\_file\_actions\_t**.

53514 A spawn file actions object, when passed to *posix\_spawn()* or *posix\_spawnp()*, shall specify how  
53515 the set of open file descriptors in the calling process is transformed into a set of potentially open  
53516 file descriptors for the spawned process. This transformation shall be as if the specified sequence  
53517 of actions was performed exactly once, in the context of the spawned process (prior to execution  
53518 of the new process image), in the order in which the actions were added to the object;  
53519 additionally, when the new process image is executed, any file descriptor (from this new set)  
53520 which has its FD\_CLOEXEC flag set shall be closed (see *posix\_spawn()*).

53521 The *posix\_spawn\_file\_actions\_addclose()* function shall add a *close* action to the object referenced  
53522 by *file\_actions* that shall cause the file descriptor *fildes* to be closed (as if *close(fildes)* had been  
53523 called) when a new process is spawned using this file actions object, except that a non-negative  
53524 *fildes* less than {OPEN\_MAX} that is already closed at the time when the new process is spawned  
53525 shall be ignored rather than failing with [EBADF].

53526 The *posix\_spawn\_file\_actions\_addopen()* function shall add an *open* action to the object referenced  
53527 by *file\_actions* that shall cause the file named by *path* to be opened (as if *open(path, oflag, mode)*  
53528 had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a  
53529 new process is spawned using this file actions object. If *fildes* was already an open file descriptor,  
53530 it shall be closed before the new file is opened. A relative *path* shall be interpreted in relation to  
53531 the working directory determined by any prior actions.

53532 The string pointed to by *path* shall be copied by the *posix\_spawn\_file\_actions\_addopen()* function.

53533 **RETURN VALUE**

53534 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
53535 be returned to indicate the error.

53536 **ERRORS**

53537 The *posix\_spawn\_file\_actions\_addopen()* function shall fail if:

53538 [EBADF] The value specified by *fildes* is negative or greater than or equal to  
53539 {OPEN\_MAX}.

53540 The *posix\_spawn\_file\_actions\_addclose()* function shall fail if:



53541 [EBADF] The value specified by *fdes* is negative.

53542 These functions shall fail if:

53543 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

53544 These functions may fail if:

53545 [EINVAL] The value specified by *file\_actions* is invalid.

53546 It shall not be considered an error for the *fdes* argument passed to these functions to specify a  
 53547 file descriptor for which the specified operation could not be performed at the time of the call.  
 53548 Any such error shall be detected when the associated file actions object is later used during a  
 53549 *posix\_spawn()* or *posix\_spawnnp()* operation.

#### 53550 EXAMPLES

53551 None.

#### 53552 APPLICATION USAGE

53553 These functions are part of the Spawn option and need not be provided on all implementations.

53554 Implementations may use file descriptors that must be inherited into child processes for the  
 53555 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,  
 53556 an application that calls *posix\_spawn\_file\_actions\_addclose()* with an arbitrary integer risks non-  
 53557 conforming behavior, and this function can only portably be used to close file descriptor values  
 53558 that the application has obtained through explicit actions, or for the three file descriptors  
 53559 corresponding to the standard file streams. In order to avoid a race condition of leaking an  
 53560 unintended file descriptor into a child process or executed program, an application should  
 53561 consider opening all file descriptors with the FD\_CLOFORK or FD\_CLOEXEC flag, or both  
 53562 flags, set unless the file descriptor is intended to be inherited by child processes or executed  
 53563 programs, respectively.

#### 53564 RATIONALE

53565 A spawn file actions object may be initialized to contain an ordered sequence of *chdir()*, *close()*,  
 53566 *dup2()*, *fchdir()*, and *open()* operations to be used by *posix\_spawn()* or *posix\_spawnnp()* to arrive at  
 53567 the set of open file descriptors and current working directory inherited by the spawned process  
 53568 from the set of open file descriptors and current working directory in the parent at the time of  
 53569 the *posix\_spawn()* or *posix\_spawnnp()* call. It had been suggested that the *close()* and *dup2()*  
 53570 operations alone are sufficient to rearrange file descriptors, and that files which need to be  
 53571 opened for use by the spawned process can be handled either by having the calling process open  
 53572 them before the *posix\_spawn()* or *posix\_spawnnp()* call (and close them after), or by passing  
 53573 pathnames to the spawned process (in *argv*) so that it may open them itself. The standard  
 53574 developers recommend that applications use one of these two methods when practical, since  
 53575 detailed error status on a failed open operation is always available to the application this way.  
 53576 However, the standard developers feel that allowing a spawn file actions object to specify open  
 53577 operations is still appropriate because:

- 53578 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 53579 2. It supports the I/O redirection paradigm commonly employed by POSIX programs  
 53580 designed to be invoked from a shell. When such a program is the child process, it may not  
 53581 be designed to open files on its own.
- 53582 3. It allows file opens that might otherwise fail or violate file ownership/access rights if  
 53583 executed by the parent process.

53584 Regarding 2. above, note that the spawn open file action provides to *posix\_spawn()* and  
 53585 *posix\_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only

53586 without the intervening execution of a shell; for example:

53587 `system ("myprog <file1 3<file2");`

53588 Regarding 3. above, note that if the calling process needs to open one or more files for access by  
 53589 the spawned process, but has insufficient spare file descriptors, then the *open* action is necessary  
 53590 to allow the *open()* to occur in the context of the child process after other file descriptors have  
 53591 been closed (that must remain open in the parent).

53592 Additionally, if a parent is executed from a file having a "set-user-id" mode bit set and the  
 53593 POSIX\_SPAWN\_RESETEUIDS flag is set in the spawn attributes, a file created within the parent  
 53594 process will (possibly incorrectly) have the parent's effective user ID as its owner, whereas a file  
 53595 created via an *open()* action during *posix\_spawn()* or *posix\_spawnnp()* will have the parent's real  
 53596 ID as its owner; and an open by the parent process may successfully open a file to which the real  
 53597 user should not have access or fail to open a file to which the real user should have access.

53598 **File Descriptor Mapping**

53599 The standard developers had originally proposed using an array which specified the mapping of  
 53600 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is  
 53601 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix\_spawn()*  
 53602 or *posix\_spawnnp()* without provision for one or more spare file descriptor entries (which simply  
 53603 may not be available). Such an array requires that an implementation develop a complex  
 53604 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor  
 53605 at the wrong time.

53606 It was noted by a member of the Ada Language Bindings working group that the approved Ada  
 53607 Language *Start\_Process* family of POSIX process primitives use a caller-specified set of file  
 53608 actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very  
 53609 flexible way, yet no such problems exist because the burden of determining how to achieve the  
 53610 final file descriptor mapping is completely on the application. Furthermore, although the file  
 53611 actions interface appears frightening at first glance, it is actually quite simple to implement in  
 53612 either a library or the kernel.

53613 The *posix\_spawn\_file\_actions\_addclose()* function is not required to check whether the file  
 53614 descriptor is less than {OPEN\_MAX} because on some implementations {OPEN\_MAX} reflects  
 53615 the RLIMIT\_NOFILE soft limit and therefore calling *setrlimit()* to reduce this limit can result in  
 53616 an {OPEN\_MAX} value less than or equal to an already open file descriptor. Applications need  
 53617 to be able to close such file descriptors on spawn. On implementations where {OPEN\_MAX}  
 53618 does not change, it is recommended that *posix\_spawn\_file\_actions\_addclose()* should return  
 53619 [EBADF] if *fd* is greater than or equal to {OPEN\_MAX}.

53620 **FUTURE DIRECTIONS**

53621 None.

53622 **SEE ALSO**

53623 *close()*, *dup()*, *open()*, *posix\_spawn()*, *posix\_spawn\_file\_actions\_addchdir()*,  
 53624 *posix\_spawn\_file\_actions\_adddup2()*, *posix\_spawn\_file\_actions\_destroy()*

53625 XBD <spawn.h>

53626 **CHANGE HISTORY**

53627 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53628 IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the  
 53629 string pointed to by *path* is copied by the *posix\_spawn\_file\_actions\_addopen()* function.

53630 **Issue 7**

53631 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0436 [418], XSH/TC1-2008/0437 [149],  
53632 XSH/TC1-2008/0438 [291], and XSH/TC1-2008/0439 [418] are applied.

53633 **Issue 8**

53634 Austin Group Defect 370 is applied, requiring that attempting to close a file descriptor that is in  
53635 range, but already closed at the time when the new process is spawned, is not treated as an error.

53636 Austin Group Defect 1208 is applied, adding *posix\_spawn\_file\_actions\_addchdir()* and  
53637 *posix\_spawn\_file\_actions\_addfchdir()*.

53638 Austin Group Defect 1318 is applied, adding FD\_CLOFORK.

53639 **NAME**

53640 posix\_spawn\_file\_actions\_adddup2 — add dup2 action to spawn file actions object  
53641 (ADVANCED REALTIME)

53642 **SYNOPSIS**

```
53643 SPN #include <spawn.h>
53644 int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t
53645 *file_actions, int fildes, int newfildes);
```

53646 **DESCRIPTION**

53647 The *posix\_spawn\_file\_actions\_adddup2()* function shall add a *dup2()* action to the object  
53648 referenced by *file\_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as  
53649 if *dup2(fildes, newfildes)* had been called) when a new process is spawned using this file actions  
53650 object.

53651 If *fildes* and *newfildes* are equal, then the action shall ensure that *newfildes* is inherited by the new  
53652 process with *FD\_CLOEXEC* clear, even if the *FD\_CLOEXEC* flag of *fildes* is set at the time the  
53653 new process is spawned, and even though *dup2()* would not make such a change.

53654 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*.

53655 **RETURN VALUE**

53656 Upon successful completion, the *posix\_spawn\_file\_actions\_adddup2()* function shall return zero;  
53657 otherwise, an error number shall be returned to indicate the error.

53658 **ERRORS**

53659 The *posix\_spawn\_file\_actions\_adddup2()* function shall fail if:

53660 [EBADF] The value specified by *fildes* or *newfildes* is negative or greater than or equal to  
53661 {OPEN\_MAX}.

53662 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

53663 The *posix\_spawn\_file\_actions\_adddup2()* function may fail if:

53664 [EINVAL] The value specified by *file\_actions* is invalid.

53665 It shall not be considered an error for the *fildes* argument passed to the  
53666 *posix\_spawn\_file\_actions\_adddup2()* function to specify a file descriptor for which the specified  
53667 operation could not be performed at the time of the call. Any such error shall be detected when  
53668 the associated file actions object is later used during a *posix\_spawn()* or *posix\_spawnnp()*  
53669 operation.

53670 **EXAMPLES**

53671 None.

53672 **APPLICATION USAGE**

53673 The *posix\_spawn\_file\_actions\_adddup2()* function is part of the Spawn option and need not be  
53674 provided on all implementations.

53675 Implementations may use file descriptors that must be inherited into child processes for the  
53676 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,  
53677 an application that calls *posix\_spawn\_file\_actions\_adddup2()* with an arbitrary integer for *newfildes*  
53678 risks non-conforming behavior, and this function can only portably be used to overwrite file  
53679 descriptor values that the application has obtained through explicit actions, or for the three file  
53680 descriptors corresponding to the standard file streams. In order to avoid a race condition of  
53681 leaking an unintended file descriptor into a child process or executed program, an application  
53682 should consider opening all file descriptors with the *FD\_CLOFORK* or *FD\_CLOEXEC* flag, or

53683 both flags, set unless the file descriptor is intended to be inherited by child processes or executed  
53684 programs, respectively.

#### 53685 RATIONALE

53686 Refer to the RATIONALE section in [posix\\_spawn\\_file\\_actions\\_addclose\(\)](#).

53687 Although `dup2()` is required to do nothing when `fildes` and `newfildes` are equal and `fildes` is an  
53688 open descriptor, the use of `posix_spawn_file_actions_adddup2()` is required to cause `fildes` to be  
53689 accessible in the child with `FD_CLOEXEC` clear. This is because there is no counterpart  
53690 `posix_spawn_file_actions_fcntl()` that could be used for clearing the flag as an independent file  
53691 action. It would also be possible to achieve this effect by using two calls to  
53692 `posix_spawn_file_actions_adddup2()` and a temporary `fildes` value known not to conflict with any  
53693 other file descriptors, coupled with a `posix_spawn_file_actions_close()` to avoid leaking the  
53694 temporary, but this approach is complex, and risks [EMFILE] or [ENFILE] failure that can be  
53695 avoided with the in-place removal of `FD_CLOEXEC`.

53696 There is no need for `posix_spawn_file_actions_adddup3()`, since it makes no sense to create a file  
53697 descriptor with `FD_CLOEXEC` set before spawning the child process, where that file descriptor  
53698 would immediately be closed again.

#### 53699 FUTURE DIRECTIONS

53700 None.

#### 53701 SEE ALSO

53702 [dup\(\)](#), [posix\\_spawn\(\)](#), [posix\\_spawn\\_file\\_actions\\_addclose\(\)](#), [posix\\_spawn\\_file\\_actions\\_destroy\(\)](#)

53703 XBD <[spawn.h](#)>

#### 53704 CHANGE HISTORY

53705 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53706 IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the  
53707 `newfildes` argument in addition to `fildes`.

#### 53708 Issue 7

53709 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0440 [149] is applied.

#### 53710 Issue 8

53711 Austin Group Defect 411 is applied, changing requirements relating to the `FD_CLOEXEC` flag  
53712 when `fildes` and `newfildes` are equal.

53713 Austin Group Defect 1318 is applied, adding `FD_CLOFORK`.

53714 **NAME**

53715         posix\_spawn\_file\_actions\_addfchdir — add fchdir action to spawn file actions object  
53716         (ADVANCED REALTIME)

53717 **SYNOPSIS**

```
53718 SPN         #include <spawn.h>  
53719         int posix_spawn_file_actions_addfchdir(posix_spawn_file_actions_t  
53720                 *file_actions, int fildes);
```

53721 **DESCRIPTION**

53722         Refer to [posix\\_spawn\\_file\\_actions\\_addchdir\(\)](#).

53723 **NAME**

53724        posix\_spawn\_file\_actions\_addopen — add open action to spawn file actions object  
53725        (ADVANCED REALTIME)

53726 **SYNOPSIS**

```
53727 SPN     #include <spawn.h>  
53728        int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t  
53729                    *restrict file_actions, int fildes,  
53730                    const char *restrict path, int oflag, mode_t mode);
```

53731 **DESCRIPTION**

53732        Refer to [posix\\_spawn\\_file\\_actions\\_addclose\(\)](#).

53733 **NAME**

53734 posix\_spawn\_file\_actions\_destroy, posix\_spawn\_file\_actions\_init — destroy and initialize  
53735 spawn file actions object (**ADVANCED REALTIME**)

53736 **SYNOPSIS**

```
53737 SPN #include <spawn.h>
53738
53738 int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t
53739     *file_actions);
53740 int posix_spawn_file_actions_init(posix_spawn_file_actions_t
53741     *file_actions);
```

53742 **DESCRIPTION**

53743 The *posix\_spawn\_file\_actions\_destroy()* function shall destroy the object referenced by *file\_actions*;  
53744 the object becomes, in effect, uninitialized. An implementation may cause  
53745 *posix\_spawn\_file\_actions\_destroy()* to set the object referenced by *file\_actions* to an invalid value. A  
53746 destroyed spawn file actions object can be reinitialized using *posix\_spawn\_file\_actions\_init()*; the  
53747 results of otherwise referencing the object after it has been destroyed are undefined.

53748 The *posix\_spawn\_file\_actions\_init()* function shall initialize the object referenced by *file\_actions* to  
53749 contain no file actions for *posix\_spawn()* or *posix\_spawnnp()* to perform.

53750 A spawn file actions object is as defined in *posix\_spawn\_file\_actions\_addclose()*.

53751 The effect of initializing an already initialized spawn file actions object is undefined.

53752 **RETURN VALUE**

53753 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
53754 be returned to indicate the error.

53755 **ERRORS**

53756 The *posix\_spawn\_file\_actions\_init()* function shall fail if:

53757 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

53758 The *posix\_spawn\_file\_actions\_destroy()* function may fail if:

53759 [EINVAL] The value specified by *file\_actions* is invalid.

53760 **EXAMPLES**

53761 None.

53762 **APPLICATION USAGE**

53763 These functions are part of the Spawn option and need not be provided on all implementations.

53764 **RATIONALE**

53765 Refer to the RATIONALE section in *posix\_spawn\_file\_actions\_addclose()*.

53766 **FUTURE DIRECTIONS**

53767 None.

53768 **SEE ALSO**

53769 *posix\_spawn()*, *posix\_spawn\_file\_actions\_addclose()*

53770 XBD <spawn.h>



53771 **CHANGE HISTORY**

53772 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53773 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

53774 **NAME**

53775 posix\_spawnattr\_destroy, posix\_spawnattr\_init — destroy and initialize spawn attributes object  
 53776 (ADVANCED REALTIME)

53777 **SYNOPSIS**

```
53778 SPN #include <spawn.h>
53779
53779 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
53780 int posix_spawnattr_init(posix_spawnattr_t *attr);
```

53781 **DESCRIPTION**

53782 The *posix\_spawnattr\_destroy()* function shall destroy a spawn attributes object. A destroyed *attr*  
 53783 attributes object can be reinitialized using *posix\_spawnattr\_init()*; the results of otherwise  
 53784 referencing the object after it has been destroyed are undefined. An implementation may cause  
 53785 *posix\_spawnattr\_destroy()* to set the object referenced by *attr* to an invalid value.

53786 The *posix\_spawnattr\_init()* function shall initialize a spawn attributes object *attr* with the default  
 53787 value for all of the individual attributes used by the implementation. Results are undefined if  
 53788 *posix\_spawnattr\_init()* is called specifying an already initialized *attr* attributes object.

53789 A spawn attributes object is of type **posix\_spawnattr\_t** (defined in **<spawn.h>**) and is used to  
 53790 specify the inheritance of process attributes across a spawn operation. POSIX.1-2024 does not  
 53791 define comparison or assignment operators for the type **posix\_spawnattr\_t**.

53792 Each implementation shall document the individual attributes it uses and their default values  
 53793 unless these values are defined by POSIX.1-2024. Attributes not defined by POSIX.1-2024, their  
 53794 default values, and the names of the associated functions to get and set those attribute values are  
 53795 implementation-defined.

53796 The resulting spawn attributes object (possibly modified by setting individual attribute values),  
 53797 is used to modify the behavior of *posix\_spawn()* or *posix\_spawnnp()*. After a spawn attributes  
 53798 object has been used to spawn a process by a call to a *posix\_spawn()* or *posix\_spawnnp()*, any  
 53799 function affecting the attributes object (including destruction) shall not affect any process that  
 53800 has been spawned in this way.

53801 **RETURN VALUE**

53802 Upon successful completion, *posix\_spawnattr\_destroy()* and *posix\_spawnattr\_init()* shall return  
 53803 zero; otherwise, an error number shall be returned to indicate the error.

53804 **ERRORS**

- 53805 The *posix\_spawnattr\_init()* function shall fail if:
- 53806 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.
- 53807 The *posix\_spawnattr\_destroy()* function may fail if:
- 53808 [EINVAL] The value specified by *attr* is invalid.

53809 **EXAMPLES**

53810 None.

53811 **APPLICATION USAGE**

53812 These functions are part of the Spawn option and need not be provided on all implementations.

53813 **RATIONALE**

53814 The original spawn interface proposed in POSIX.1-2024 defined the attributes that specify the  
 53815 inheritance of process attributes across a spawn operation as a structure. In order to be able to  
 53816 separate optional individual attributes under their appropriate options (that is, the *spawn-*  
 53817 *schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and

53818 also for extensibility and consistency with the newer POSIX interfaces, the attributes interface  
53819 has been changed to an opaque data type. This interface now consists of the type  
53820 **posix\_spawnattr\_t**, representing a spawn attributes object, together with associated functions to  
53821 initialize or destroy the attributes object, and to set or get each individual attribute. Although the  
53822 new object-oriented interface is more verbose than the original structure, it is simple to use,  
53823 more extensible, and easy to implement.

#### 53824 **FUTURE DIRECTIONS**

53825 None.

#### 53826 **SEE ALSO**

53827 *posix\_spawn()*, *posix\_spawnattr\_getsigdefault()*, *posix\_spawnattr\_getflags()*,  
53828 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
53829 *posix\_spawnattr\_getsigmask()*

53830 XBD <spawn.h>

#### 53831 **CHANGE HISTORY**

53832 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53833 IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already  
53834 initialized spawn attributes option is undefined.

53835 **NAME**

53836 posix\_spawnattr\_getflags, posix\_spawnattr\_setflags — get and set the spawn-flags attribute of a  
53837 spawn attributes object (**ADVANCED REALTIME**)

53838 **SYNOPSIS**

```
53839 SPN #include <spawn.h>
53840 int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
53841 short *restrict flags);
53842 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

53843 **DESCRIPTION**

53844 The *posix\_spawnattr\_getflags()* function shall obtain the value of the *spawn-flags* attribute from the  
53845 attributes object referenced by *attr*.

53846 The *posix\_spawnattr\_setflags()* function shall set the *spawn-flags* attribute in an initialized  
53847 attributes object referenced by *attr*.

53848 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the  
53849 new process image when invoking *posix\_spawn()* or *posix\_spawnnp()*. It is the bitwise-inclusive  
53850 OR of zero or more of the following flags:

- 53851 POSIX\_SPAWN\_RESETIDS
- 53852 POSIX\_SPAWN\_SETPGROUP
- 53853 PS POSIX\_SPAWN\_SETSCHEDPARAM
- 53854 POSIX\_SPAWN\_SETSCHEDULER
- 53855 POSIX\_SPAWN\_SETSID
- 53856 POSIX\_SPAWN\_SETSIGDEF
- 53857 POSIX\_SPAWN\_SETSIGMASK

53858 These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags  
53859 were set.

53860 **RETURN VALUE**

53861 Upon successful completion, *posix\_spawnattr\_getflags()* shall return zero and store the value of  
53862 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an  
53863 error number shall be returned to indicate the error.

53864 Upon successful completion, *posix\_spawnattr\_setflags()* shall return zero; otherwise, an error  
53865 number shall be returned to indicate the error.

53866 **ERRORS**

53867 These functions may fail if:

- 53868 [EINVAL] The value specified by *attr* is invalid.
- 53869 The *posix\_spawnattr\_setflags()* function may fail if:
  - 53870 [EINVAL] The value of the attribute being set is not valid.

53871 **EXAMPLES**

53872 None.

53873 **APPLICATION USAGE**

53874 These functions are part of the Spawn option and need not be provided on all implementations.

53875 **RATIONALE**

53876 None.

53877 **FUTURE DIRECTIONS**

53878 None.

53879 **SEE ALSO**

53880 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
53881 *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
53882 *posix\_spawnattr\_getsigmask()*

53883 XBD &lt;spawn.h&gt;

53884 **CHANGE HISTORY**

53885 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53886 **Issue 8**

53887 Austin Group Defect 1044 is applied, adding POSIX\_SPAWN\_SETSID.

53888 **NAME**

53889 posix\_spawnattr\_getpgroup, posix\_spawnattr\_setpgroup — get and set the spawn-pgroup  
53890 attribute of a spawn attributes object (**ADVANCED REALTIME**)

53891 **SYNOPSIS**

```
53892 SPN #include <spawn.h>
53893 int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
53894 pid_t *restrict pgroup);
53895 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

53896 **DESCRIPTION**

53897 The *posix\_spawnattr\_getpgroup()* function shall obtain the value of the *spawn-pgroup* attribute  
53898 from the attributes object referenced by *attr*.

53899 The *posix\_spawnattr\_setpgroup()* function shall set the *spawn-pgroup* attribute in an initialized  
53900 attributes object referenced by *attr*.

53901 The *spawn-pgroup* attribute represents the process group to be joined by the new process image  
53902 in a spawn operation (if POSIX\_SPAWN\_SETPGROUP is set in the *spawn-flags* attribute). The  
53903 default value of this attribute shall be zero.

53904 **RETURN VALUE**

53905 Upon successful completion, *posix\_spawnattr\_getpgroup()* shall return zero and store the value of  
53906 the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise,  
53907 an error number shall be returned to indicate the error.

53908 Upon successful completion, *posix\_spawnattr\_setpgroup()* shall return zero; otherwise, an error  
53909 number shall be returned to indicate the error.

53910 **ERRORS**

53911 These functions may fail if:

53912 [EINVAL] The value specified by *attr* is invalid.

53913 The *posix\_spawnattr\_setpgroup()* function may fail if:

53914 [EINVAL] The value of the attribute being set is not valid.

53915 **EXAMPLES**

53916 None.

53917 **APPLICATION USAGE**

53918 These functions are part of the Spawn option and need not be provided on all implementations.

53919 **RATIONALE**

53920 None.

53921 **FUTURE DIRECTIONS**

53922 None.

53923 **SEE ALSO**

53924 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
53925 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*,  
53926 *posix\_spawnattr\_getsigmask()*

53927 XBD [<spawn.h>](#)

53928 **CHANGE HISTORY**

53929 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53930 **NAME**

53931 posix\_spawnattr\_getschedparam, posix\_spawnattr\_setschedparam — get and set the spawn-  
53932 schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

53933 **SYNOPSIS**

```
53934 SPN PS #include <spawn.h>
53935 #include <sched.h>

53936 int posix_spawnattr_getschedparam(const posix_spawnattr_t
53937     *restrict attr, struct sched_param *restrict schedparam);
53938 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
53939     const struct sched_param *restrict schedparam);
```

53940 **DESCRIPTION**

53941 The *posix\_spawnattr\_getschedparam()* function shall obtain the value of the *spawn-schedparam*  
53942 attribute from the attributes object referenced by *attr*.

53943 The *posix\_spawnattr\_setschedparam()* function shall set the *spawn-schedparam* attribute in an  
53944 initialized attributes object referenced by *attr*.

53945 The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new  
53946 process image in a spawn operation (if POSIX\_SPAWN\_SETSCHEDULER or  
53947 POSIX\_SPAWN\_SETSCHEDPARAM is set in the *spawn-flags* attribute). The default value of this  
53948 attribute is unspecified.

53949 **RETURN VALUE**

53950 Upon successful completion, *posix\_spawnattr\_getschedparam()* shall return zero and store the  
53951 value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam*  
53952 parameter; otherwise, an error number shall be returned to indicate the error.

53953 Upon successful completion, *posix\_spawnattr\_setschedparam()* shall return zero; otherwise, an  
53954 error number shall be returned to indicate the error.

53955 **ERRORS**

53956 These functions may fail if:

53957 [EINVAL] The value specified by *attr* is invalid.

53958 The *posix\_spawnattr\_setschedparam()* function may fail if:

53959 [EINVAL] The value of the attribute being set is not valid.

53960 **EXAMPLES**

53961 None.

53962 **APPLICATION USAGE**

53963 These functions are part of the Spawn and Process Scheduling options and need not be provided  
53964 on all implementations.

53965 **RATIONALE**

53966 None.

53967 **FUTURE DIRECTIONS**

53968 None.

53969 **SEE ALSO**

53970 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
53971 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedpolicy()*,  
53972 *posix\_spawnattr\_getsigmask()*



53973 XBD <sched.h>, <spawn.h>

53974 **CHANGE HISTORY**

53975 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53976 **NAME**

53977 posix\_spawnattr\_getschedpolicy, posix\_spawnattr\_setschedpolicy — get and set the spawn-  
53978 schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

53979 **SYNOPSIS**

```
53980 SPN PS #include <spawn.h>
53981 #include <sched.h>

53982 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
53983     *restrict attr, int *restrict schedpolicy);
53984 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
53985     int schedpolicy);
```

53986 **DESCRIPTION**

53987 The *posix\_spawnattr\_getschedpolicy()* function shall obtain the value of the *spawn-schedpolicy*  
53988 attribute from the attributes object referenced by *attr*.

53989 The *posix\_spawnattr\_setschedpolicy()* function shall set the *spawn-schedpolicy* attribute in an  
53990 initialized attributes object referenced by *attr*.

53991 The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new  
53992 process image in a spawn operation (if POSIX\_SPAWN\_SETSCHEDULER is set in the *spawn-*  
53993 *flags* attribute). The default value of this attribute is unspecified.

53994 **RETURN VALUE**

53995 Upon successful completion, *posix\_spawnattr\_getschedpolicy()* shall return zero and store the  
53996 value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy*  
53997 parameter; otherwise, an error number shall be returned to indicate the error.

53998 Upon successful completion, *posix\_spawnattr\_setschedpolicy()* shall return zero; otherwise, an  
53999 error number shall be returned to indicate the error.

54000 **ERRORS**

54001 These functions may fail if:

54002 [EINVAL] The value specified by *attr* is invalid.

54003 The *posix\_spawnattr\_setschedpolicy()* function may fail if:

54004 [EINVAL] The value of the attribute being set is not valid.

54005 **EXAMPLES**

54006 None.

54007 **APPLICATION USAGE**

54008 These functions are part of the Spawn and Process Scheduling options and need not be provided  
54009 on all implementations.

54010 **RATIONALE**

54011 None.

54012 **FUTURE DIRECTIONS**

54013 None.

54014 **SEE ALSO**

54015 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
54016 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
54017 *posix\_spawnattr\_getsigmask()*

54018 XBD <sched.h>, <spawn.h>

54019 **CHANGE HISTORY**

54020 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

54021 **NAME**

54022 posix\_spawnattr\_getsigdefault, posix\_spawnattr\_setsigdefault — get and set the spawn-  
54023 sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

54024 **SYNOPSIS**

```
54025 SPN #include <signal.h>
54026 #include <spawn.h>
54027 int posix_spawnattr_getsigdefault(const posix_spawnattr_t
54028     *restrict attr, sigset_t *restrict sigdefault);
54029 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
54030     const sigset_t *restrict sigdefault);
```

54031 **DESCRIPTION**

54032 The *posix\_spawnattr\_getsigdefault()* function shall obtain the value of the *spawn-sigdefault*  
54033 attribute from the attributes object referenced by *attr*.

54034 The *posix\_spawnattr\_setsigdefault()* function shall set the *spawn-sigdefault* attribute in an  
54035 initialized attributes object referenced by *attr*.

54036 The *spawn-sigdefault* attribute represents the set of signals to be forced to default signal handling  
54037 in the new process image (if POSIX\_SPAWN\_SETSIGDEF is set in the *spawn-flags* attribute) by a  
54038 spawn operation. The default value of this attribute shall be an empty signal set.

54039 **RETURN VALUE**

54040 Upon successful completion, *posix\_spawnattr\_getsigdefault()* shall return zero and store the value  
54041 of the *spawn-sigdefault* attribute of *attr* into the object referenced by the *sigdefault* parameter;  
54042 otherwise, an error number shall be returned to indicate the error.

54043 Upon successful completion, *posix\_spawnattr\_setsigdefault()* shall return zero; otherwise, an error  
54044 number shall be returned to indicate the error.

54045 **ERRORS**

54046 These functions may fail if:

54047 [EINVAL] The value specified by *attr* is invalid.

54048 The *posix\_spawnattr\_setsigdefault()* function may fail if:

54049 [EINVAL] The value of the attribute being set is not valid.

54050 **EXAMPLES**

54051 None.

54052 **APPLICATION USAGE**

54053 These functions are part of the Spawn option and need not be provided on all implementations.

54054 **RATIONALE**

54055 None.

54056 **FUTURE DIRECTIONS**

54057 None.

54058 **SEE ALSO**

54059 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*,  
54060 *posix\_spawnattr\_getschedparam()*, *posix\_spawnattr\_getschedpolicy()*, *posix\_spawnattr\_getsigmask()*

54061 XBD [<signal.h>](#), [<spawn.h>](#)

54062 **CHANGE HISTORY**

54063 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

54064 **NAME**

54065 posix\_spawnattr\_getsigmask, posix\_spawnattr\_setsigmask — get and set the spawn-sigmask  
54066 attribute of a spawn attributes object (**ADVANCED REALTIME**)

54067 **SYNOPSIS**

```
54068 SPN #include <signal.h>
54069 #include <spawn.h>
54070 int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
54071 sigset_t *restrict sigmask);
54072 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
54073 const sigset_t *restrict sigmask);
```

54074 **DESCRIPTION**

54075 The *posix\_spawnattr\_getsigmask()* function shall obtain the value of the *spawn-sigmask* attribute  
54076 from the attributes object referenced by *attr*.

54077 The *posix\_spawnattr\_setsigmask()* function shall set the *spawn-sigmask* attribute in an initialized  
54078 attributes object referenced by *attr*.

54079 The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a  
54080 spawn operation (if *POSIX\_SPAWN\_SETSIGMASK* is set in the *spawn-flags* attribute). The  
54081 default value of this attribute is unspecified.

54082 **RETURN VALUE**

54083 Upon successful completion, *posix\_spawnattr\_getsigmask()* shall return zero and store the value  
54084 of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter;  
54085 otherwise, an error number shall be returned to indicate the error.

54086 Upon successful completion, *posix\_spawnattr\_setsigmask()* shall return zero; otherwise, an error  
54087 number shall be returned to indicate the error.

54088 **ERRORS**

54089 These functions may fail if:

54090 [EINVAL] The value specified by *attr* is invalid.

54091 The *posix\_spawnattr\_setsigmask()* function may fail if:

54092 [EINVAL] The value of the attribute being set is not valid.

54093 **EXAMPLES**

54094 None.

54095 **APPLICATION USAGE**

54096 These functions are part of the Spawn option and need not be provided on all implementations.

54097 **RATIONALE**

54098 None.

54099 **FUTURE DIRECTIONS**

54100 None.

54101 **SEE ALSO**

54102 *posix\_spawn()*, *posix\_spawnattr\_destroy()*, *posix\_spawnattr\_getsigdefault()*,  
54103 *posix\_spawnattr\_getflags()*, *posix\_spawnattr\_getpgroup()*, *posix\_spawnattr\_getschedparam()*,  
54104 *posix\_spawnattr\_getschedpolicy()*

54105 XBD [<signal.h>](#), [<spawn.h>](#)

54106 **CHANGE HISTORY**

54107 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

54108 **NAME**

54109        posix\_spawnattr\_init — initialize the spawn attributes object (**ADVANCED REALTIME**)

54110 **SYNOPSIS**

```
54111 SPN    #include <spawn.h>
54112        int posix_spawnattr_init(posix_spawnattr_t *attr);
```

54113 **DESCRIPTION**

54114        Refer to [posix\\_spawnattr\\_destroy\(\)](#).



54115 **NAME**

54116        posix\_spawnattr\_setflags — set the spawn-flags attribute of a spawn attributes object  
54117        (ADVANCED REALTIME)

54118 **SYNOPSIS**

```
54119 SPN     #include <spawn.h>  
54120     int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

54121 **DESCRIPTION**

54122        Refer to *posix\_spawnattr\_getflags()*.

54123 **NAME**

54124        `posix_spawnattr_setpgroup` — set the spawn-pgroup attribute of a spawn attributes object  
54125        (**ADVANCED REALTIME**)

54126 **SYNOPSIS**

```
54127 SPN    #include <spawn.h>  
54128        int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

54129 **DESCRIPTION**

54130        Refer to *posix\_spawnattr\_getpgroup()*.

54131 **NAME**

54132        posix\_spawnattr\_setschedparam — set the spawn-schedparam attribute of a spawn attributes  
54133        object (**ADVANCED REALTIME**)

54134 **SYNOPSIS**

```
54135 SPN PS #include <sched.h>  
54136         #include <spawn.h>  
  
54137         int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,  
54138         const struct sched_param *restrict schedparam);
```

54139 **DESCRIPTION**

54140        Refer to [posix\\_spawnattr\\_getschedparam\(\)](#).

54141 **NAME**

54142        `posix_spawnattr_setschedpolicy` — set the spawn-schedpolicy attribute of a spawn attributes  
54143        object (**ADVANCED REALTIME**)

54144 **SYNOPSIS**

```
54145 SPN PS #include <sched.h>  
54146         #include <spawn.h>  
  
54147         int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,  
54148         int schedpolicy);
```

54149 **DESCRIPTION**

54150        Refer to [posix\\_spawnattr\\_getschedpolicy\(\)](#).

54151 **NAME**

54152        posix\_spawnattr\_setsigdefault — set the spawn-sigdefault attribute of a spawn attributes object  
54153        (ADVANCED REALTIME)

54154 **SYNOPSIS**

```
54155 SPN     #include <signal.h>  
54156           #include <spawn.h>  
  
54157     int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,  
54158                                    const sigset_t *restrict sigdefault);
```

54159 **DESCRIPTION**

54160        Refer to [posix\\_spawnattr\\_getsigdefault\(\)](#).

54161 **NAME**

54162        posix\_spawnattr\_setsigmask — set the spawn-sigmask attribute of a spawn attributes object  
54163        (ADVANCED REALTIME)

54164 **SYNOPSIS**

```
54165 SPN     #include <signal.h>  
54166         #include <spawn.h>  
  
54167         int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,  
54168                                        const sigset_t *restrict sigmask);
```

54169 **DESCRIPTION**

54170        Refer to [posix\\_spawnattr\\_getsigmask\(\)](#).

54171 **NAME**54172 posix\_spawn — spawn a process (**ADVANCED REALTIME**)54173 **SYNOPSIS**

```
54174 SPN #include <spawn.h>
54175 int posix_spawn(pid_t *restrict pid, const char *restrict file,
54176 const posix_spawn_file_actions_t *file_actions,
54177 const posix_spawnattr_t *restrict attrp,
54178 char *const argv[restrict], char *const envp[restrict]);
```

54179 **DESCRIPTION**54180 Refer to *posix\_spawn()*.

54181 **NAME**54182 posix\_typed\_mem\_get\_info — query typed memory information (**ADVANCED REALTIME**)54183 **SYNOPSIS**

```
54184 TYM #include <sys/mman.h>
54185 int posix_typed_mem_get_info(int fildev,
54186 struct posix_typed_mem_info *info);
```

54187 **DESCRIPTION**

54188 The *posix\_typed\_mem\_get\_info()* function shall return, in the *posix\_tmi\_length* field of the  
 54189 **posix\_typed\_mem\_info** structure pointed to by *info*, the maximum length which may be  
 54190 successfully allocated by the typed memory object designated by *fildev*. This maximum length  
 54191 shall take into account the flag `POSIX_TYPED_MEM_ALLOCATE` or  
 54192 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified when the typed memory object  
 54193 represented by *fildev* was opened. The maximum length is dynamic; therefore, the value  
 54194 returned is valid only while the current mapping of the corresponding typed memory pool  
 54195 remains unchanged.

54196 If *fildev* represents a typed memory object opened with neither the  
 54197 `POSIX_TYPED_MEM_ALLOCATE` flag nor the `POSIX_TYPED_MEM_ALLOCATE_CONTIG`  
 54198 flag specified, the returned value of *info->posix\_tmi\_length* is unspecified.

54199 The *posix\_typed\_mem\_get\_info()* function may return additional implementation-defined  
 54200 information in other fields of the **posix\_typed\_mem\_info** structure pointed to by *info*.

54201 If the memory object specified by *fildev* is not a typed memory object, then the behavior of this  
 54202 function is undefined.

54203 **RETURN VALUE**

54204 Upon successful completion, the *posix\_typed\_mem\_get\_info()* function shall return zero;  
 54205 otherwise, the corresponding error status value shall be returned.

54206 **ERRORS**

54207 The *posix\_typed\_mem\_get\_info()* function shall fail if:

54208 [EBADF] The *fildev* argument is not a valid open file descriptor.

54209 [ENODEV] The *fildev* argument is not connected to a memory object supported by this  
 54210 function.

54211 This function shall not return an error code of [EINTR].

54212 **EXAMPLES**

54213 None.

54214 **APPLICATION USAGE**

54215 None.

54216 **RATIONALE**

54217 An application that needs to allocate a block of typed memory with length dependent upon the  
 54218 amount of memory currently available must either query the typed memory object to obtain the  
 54219 amount available, or repeatedly invoke *mmap()* attempting to guess an appropriate length.  
 54220 While the latter method is existing practice with *malloc()*, it is awkward and imprecise. The  
 54221 *posix\_typed\_mem\_get\_info()* function allows an application to immediately determine available  
 54222 memory. This is particularly important for typed memory objects that may in some cases be  
 54223 scarce resources. Note that when a typed memory pool is a shared resource, some form of  
 54224 mutual-exclusion or synchronization may be required while typed memory is being queried and



54225 allocated to prevent race conditions.

54226 The existing *fstat()* function is not suitable for this purpose. We realize that implementations  
54227 may wish to provide other attributes of typed memory objects (for example, alignment  
54228 requirements, page size, and so on). The *fstat()* function returns a structure which is not  
54229 extensible and, furthermore, contains substantial information that is inappropriate for typed  
54230 memory objects.

54231 **FUTURE DIRECTIONS**

54232 None.

54233 **SEE ALSO**

54234 *fstat()*, *mmap()*, *posix\_typed\_mem\_open()*

54235 XBD <sys/mman.h>

54236 **CHANGE HISTORY**

54237 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

54238 **NAME**54239 posix\_typed\_mem\_open — open a typed memory object (**ADVANCED REALTIME**)54240 **SYNOPSIS**

```
54241 TYM #include <sys/mman.h>
54242 int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

54243 **DESCRIPTION**

54244 The `posix_typed_mem_open()` function shall establish a connection between the typed memory  
 54245 object specified by the string pointed to by `name` and a file descriptor. It shall create an open file  
 54246 description that refers to the typed memory object and a file descriptor that refers to that open  
 54247 file description. The file descriptor shall be allocated as described in [Section 2.6](#) (on page 525)  
 54248 and can be used by other functions to refer to that typed memory object. It is unspecified  
 54249 whether the name appears in the file system and is visible to other functions that take  
 54250 pathnames as arguments. The `name` argument conforms to the construction rules for a pathname,  
 54251 except that the interpretation of `<slash>` characters other than the leading `<slash>` character in  
 54252 `name` is implementation-defined, and that the length limits for the `name` argument are  
 54253 implementation-defined and need not be the same as the pathname limits `{PATH_MAX}` and  
 54254 `{NAME_MAX}`. If `name` begins with the `<slash>` character, then processes calling  
 54255 `posix_typed_mem_open()` with the same value of `name` shall refer to the same typed memory  
 54256 object. If `name` does not begin with the `<slash>` character, the effect is implementation-defined.

54257 Each typed memory object supported in a system shall be identified by a name which specifies  
 54258 not only its associated typed memory pool, but also the path or port by which it is accessed. That  
 54259 is, the same typed memory pool accessed via several different ports shall have several different  
 54260 corresponding names. The binding between names and typed memory objects is established in  
 54261 an implementation-defined manner. Unlike shared memory objects, there is no way within  
 54262 POSIX.1-2024 for a program to create a typed memory object.

54263 The value of `tflag` shall determine how the typed memory object behaves when subsequently  
 54264 mapped by calls to `mmap()`. At most, one of the following flags defined in `<sys/mman.h>` may  
 54265 be specified:

54266 POSIX\_TYPED\_MEM\_ALLOCATE

54267 Allocate on `mmap()`.

54268 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG

54269 Allocate contiguously on `mmap()`.

54270 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE

54271 Map on `mmap()`, without affecting allocatability.

54272 If `tflag` has the flag `POSIX_TYPED_MEM_ALLOCATE` specified, any subsequent call to `mmap()`  
 54273 using the returned file descriptor shall result in allocation and mapping of typed memory from  
 54274 the specified typed memory pool. The allocated memory may be a contiguous previously  
 54275 unallocated area of the typed memory pool or several non-contiguous previously unallocated  
 54276 areas (mapped to a contiguous portion of the process address space). If `tflag` has the flag  
 54277 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified, any subsequent call to `mmap()` using the  
 54278 returned file descriptor shall result in allocation and mapping of a single contiguous previously  
 54279 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process  
 54280 address space). If `tflag` has none of the flags `POSIX_TYPED_MEM_ALLOCATE` or  
 54281 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified, any subsequent call to `mmap()` using the  
 54282 returned file descriptor shall map an application-chosen area from the specified typed memory  
 54283 pool such that this mapped area becomes unavailable for allocation until unmapped by all  
 54284 processes. If `tflag` has the flag `POSIX_TYPED_MEM_MAP_ALLOCATABLE` specified, any

54285 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen  
 54286 area from the specified typed memory pool without an effect on the availability of that area for  
 54287 allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it  
 54288 was unallocated prior to the mapping or allocated if it was allocated prior to the mapping.  
 54289 Appropriate privileges to specify the POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag are  
 54290 implementation-defined.

54291 If successful, *posix\_typed\_mem\_open()* shall return a file descriptor for the typed memory object.  
 54292 The open file description is new, and therefore the file descriptor shall not share it with any other  
 54293 processes. It is unspecified whether the file offset is set. The FD\_CLOEXEC file descriptor flag  
 54294 associated with the new file descriptor shall be cleared unless *oflag* includes O\_CLOEXEC. The  
 54295 FD\_CLOFORK file descriptor flag associated with the new file descriptor shall be cleared unless  
 54296 *oflag* includes O\_CLOFORK.

54297 The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*,  
 54298 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *fstat()*, *dup()*, *dup2()*, *dup3()*, and *close()*, is  
 54299 unspecified when passed a file descriptor connected to a typed memory object by this function.

54300 The file status flags of the open file description shall be set according to the value of *oflag*.  
 54301 Applications shall specify exactly one of the three access mode values described below as the  
 54302 value of *oflag*.

54303 O\_RDONLY Open for read access only.

54304 O\_WRONLY Open for write access only.

54305 O\_RDWR Open for read or write access.

54306 Additionally, the value of *oflag* can include the following flags:

54307 O\_CLOEXEC Set the FD\_CLOEXEC file descriptor flag.

54308 O\_CLOFORK Set the FD\_CLOFORK file descriptor flag.

#### 54309 RETURN VALUE

54310 Upon successful completion, the *posix\_typed\_mem\_open()* function shall return a non-negative  
 54311 integer representing the file descriptor. Otherwise, it shall return  $-1$  and set *errno* to indicate the  
 54312 error.

#### 54313 ERRORS

54314 The *posix\_typed\_mem\_open()* function shall fail if:

54315 [EACCES] The typed memory object exists and the permissions specified by *oflag* are  
 54316 denied.

54317 [EINTR] The *posix\_typed\_mem\_open()* operation was interrupted by a signal.

54318 [EINVAL] The flags specified in *tflag* are invalid (more than one of  
 54319 POSIX\_TYPED\_MEM\_ALLOCATE,  
 54320 POSIX\_TYPED\_MEM\_ALLOCATE\_CONTIG, or  
 54321 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE is specified).

54322 [EMFILE] All file descriptors available to the process are currently open.

54323 [ENFILE] Too many file descriptors are currently open in the system.

54324 [ENOENT] The named typed memory object does not exist.

54325 [EPERM] The caller lacks appropriate privileges to specify the  
 54326 POSIX\_TYPED\_MEM\_MAP\_ALLOCATABLE flag in the *tflag* argument.

54327 The *posix\_typed\_mem\_open()* function may fail if:  
 54328 [ENAMETOOLONG]  
 54329 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems  
 54330 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI  
 54331 systems, or has a pathname component that is longer than  
 54332 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or  
 54333 longer than `{_XOPEN_NAME_MAX}` on XSI systems.

54334 **EXAMPLES**  
 54335 None.

54336 **APPLICATION USAGE**  
 54337 None.

54338 **RATIONALE**  
 54339 The use of the `O_CLOEXEC` and `O_CLOFORK` flags to *posix\_typed\_mem\_open()* is necessary to  
 54340 avoid leaking typed memory file descriptors to child processes, since *fcntl()* has unspecified  
 54341 results on typed memory objects and therefore cannot be used to set `FD_CLOEXEC` or  
 54342 `FD_CLOFORK` after the file descriptor has been opened. The *exec* family of functions already  
 54343 unmaps all memory associated with a typed memory object, but does not close the file  
 54344 descriptor unless `FD_CLOEXEC` is also set.

54345 **FUTURE DIRECTIONS**  
 54346 None.

54347 **SEE ALSO**  
 54348 [Section 2.6](#) (on page 525), *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *ftruncate()*, *mmap()*, *msync()*,  
 54349 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *umask()*  
 54350 XBD [<fcntl.h>](#), [<sys/mman.h>](#)

54351 **CHANGE HISTORY**  
 54352 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

54353 **Issue 7**  
 54354 Austin Group Interpretation 1003.1-2001 #143 is applied.  
 54355 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0442 [119,428] is applied.  
 54356 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0256 [835], XSH/TC2-2008/0257 [835],  
 54357 and XSH/TC2-2008/0258 [835] are applied.

54358 **Issue 8**  
 54359 Austin Group Defects 411 and 1318 are applied, adding `O_CLOEXEC`, `O_CLOFORK`, and  
 54360 *dup3()*.  
 54361 Austin Group Defect 593 is applied, removing a reference to [<fcntl.h>](#) from the DESCRIPTION  
 54362 section.

54363 **NAME**

54364 pow, powf, powl — power function

54365 **SYNOPSIS**

54366 #include &lt;math.h&gt;

54367 double pow(double x, double y);

54368 float powf(float x, float y);

54369 long double powl(long double x, long double y);

54370 **DESCRIPTION**

54371 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 54372 conflict between the requirements described here and the ISO C standard is unintentional. This  
 54373 volume of POSIX.1-2024 defers to the ISO C standard.

54374 These functions shall compute the value of  $x$  raised to the power  $y$ ,  $x^y$ . If  $x$  is negative, the  
 54375 application shall ensure that  $y$  is an integer value.

54376 An application wishing to check for error situations should set *errno* to zero and call  
 54377 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 54378 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 54379 zero, an error has occurred.

54380 **RETURN VALUE**54381 Upon successful completion, these functions shall return the value of  $x$  raised to the power  $y$ .

54382 MX For finite values of  $x < 0$ , and finite non-integer values of  $y$ , a domain error shall occur and  
 54383 either a NaN (if representable), or an implementation-defined value shall be returned.

54384 If the correct value would cause overflow, a range error shall occur and *pow()*, *powf()*, and  
 54385 *powl()* shall return  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively, with the  
 54386 same sign as the correct value of the function.

54387 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
 54388 MXX and *pow()*, *powf()*, and *powl()* shall return 0.0, or (if IEC 60559 Floating-Point is not supported)  
 54389 an implementation-defined value no greater in magnitude than DBL\_MIN, FLT\_MIN, and  
 54390 LDBL\_MIN, respectively.

54391 CX For  $y < 0$ , if  $x$  is zero, a pole error may occur and *pow()*, *powf()*, and *powl()* shall return  
 54392 MX  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively. On systems that support the  
 54393 IEC 60559 Floating-Point option, if  $x$  is  $\pm 0$ :

- 54394 • If  $y$  is an odd integer, a pole error shall occur and *pow()*, *powf()*, and *powl()* shall return  
 54395  $\pm$ HUGE\_VAL,  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL, respectively.
- 54396 • If  $y$  is finite and is not an odd integer, a pole error shall occur and *pow()*, *powf()*, and *powl()*  
 54397 shall return HUGE\_VAL, HUGE\_VALF, and HUGE\_VALL, respectively.
- 54398 • If  $y$  is  $-\text{Inf}$ , a pole error may occur and *pow()*, *powf()*, and *powl()* shall return HUGE\_VAL,  
 54399 HUGE\_VALF, and HUGE\_VALL, respectively.

54400 MX If  $x$  or  $y$  is a NaN, a NaN shall be returned (unless specified elsewhere in this description).

54401 For any value of  $y$  (including NaN), if  $x$  is +1, 1.0 shall be returned.

54402 For any value of  $x$  (including NaN), if  $y$  is  $\pm 0$ , 1.0 shall be returned.

54403 For any odd integer value of  $y > 0$ , if  $x$  is  $\pm 0$ ,  $\pm 0$  shall be returned.

54404 For  $y > 0$  and not an odd integer, if  $x$  is  $\pm 0$ , +0 shall be returned.

54405 If  $x$  is  $-1$ , and  $y$  is  $\pm\text{Inf}$ ,  $1.0$  shall be returned.

54406 For  $|x| < 1$ , if  $y$  is  $-\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

54407 For  $|x| > 1$ , if  $y$  is  $-\text{Inf}$ ,  $+0$  shall be returned.

54408 For  $|x| < 1$ , if  $y$  is  $+\text{Inf}$ ,  $+0$  shall be returned.

54409 For  $|x| > 1$ , if  $y$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

54410 For  $y$  an odd integer  $< 0$ , if  $x$  is  $-\text{Inf}$ ,  $-0$  shall be returned.

54411 For  $y < 0$  and not an odd integer, if  $x$  is  $-\text{Inf}$ ,  $+0$  shall be returned.

54412 For  $y$  an odd integer  $> 0$ , if  $x$  is  $-\text{Inf}$ ,  $-\text{Inf}$  shall be returned.

54413 For  $y > 0$  and not an odd integer, if  $x$  is  $-\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

54414 For  $y < 0$ , if  $x$  is  $+\text{Inf}$ ,  $+0$  shall be returned.

54415 For  $y > 0$ , if  $x$  is  $+\text{Inf}$ ,  $+\text{Inf}$  shall be returned.

54416 MXX If the correct value would cause underflow, and is representable, a range error may occur and  
54417 the correct value shall be returned.

#### 54418 ERRORS

54419 These functions shall fail if:

54420 Domain Error The value of  $x$  is negative and  $y$  is a finite non-integer.

54421 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
54422 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
54423 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
54424 shall be raised.

54425 MX Pole Error The value of  $x$  is zero and  $y$  is negative.

54426 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
54427 then *errno* shall be set to [ERANGE]. If the integer expression  
54428 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
54429 floating-point exception shall be raised.

54430 Range Error The result overflows.

54431 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
54432 then *errno* shall be set to [ERANGE]. If the integer expression  
54433 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
54434 floating-point exception shall be raised.

54435 These functions may fail if:

54436 Pole Error The value of  $x$  is zero and  $y$  is negative.

54437 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
54438 then *errno* shall be set to [ERANGE]. If the integer expression  
54439 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the divide-by-zero  
54440 floating-point exception shall be raised.

54441 Range Error The result underflows.

54442 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
54443 then *errno* shall be set to [ERANGE]. If the integer expression  
54444 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow

54445 floating-point exception shall be raised.

54446 **EXAMPLES**

54447 None.

54448 **APPLICATION USAGE**

54449 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
54450 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

54451 **RATIONALE**

54452 None.

54453 **FUTURE DIRECTIONS**

54454 None.

54455 **SEE ALSO**

54456 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

54457 XBD Section 4.23 (on page 109), <math.h>

54458 **CHANGE HISTORY**

54459 First released in Issue 1. Derived from Issue 1 of the SVID.

54460 **Issue 5**

54461 The DESCRIPTION is updated to indicate how an application should check for an error. This  
54462 text was previously published in the APPLICATION USAGE section.

54463 **Issue 6**

54464 The normative text is updated to avoid use of the term “must” for application requirements.

54465 The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

54466 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
54467 revised to align with the ISO/IEC 9899:1999 standard.

54468 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
54469 marked.

54470 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third  
54471 paragraph in the RETURN VALUE section.

54472 **Issue 7**

54473 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

54474 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0443 [68], XSH/TC1-2008/0444 [148],  
54475 and XSH/TC1-2008/0445 [68] are applied.

54476 **Issue 8**

54477 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
54478 standard.

54479 **NAME**

54480 ppoll — input/output multiplexing

54481 **SYNOPSIS**

54482 #include <poll.h>

```
54483 int ppoll(struct pollfd fds[], nfd_t nfds,  
54484           const struct timespec *restrict timeout,  
54485           const sigset_t *restrict sigmask);
```

54486 **DESCRIPTION**

54487 Refer to *poll()*.



54488 **NAME**

54489        pread — read from a file

54490 **SYNOPSIS**

54491        #include &lt;unistd.h&gt;

54492        ssize\_t pread(int *fd*, void \**buf*, size\_t *nbyte*, off\_t *offset*);54493 **DESCRIPTION**54494        Refer to *read()*.

54495 **NAME**

54496           printf — print formatted output

54497 **SYNOPSIS**

54498           #include <stdio.h>

54499           int printf(const char \*restrict *format*, ...);

54500 **DESCRIPTION**

54501           Refer to *fprintf()*.

54502 **NAME**

54503 pselect, select — synchronous I/O multiplexing

54504 **SYNOPSIS**

```
54505 #include <sys/select.h>
54506 int pselect(int nfds, fd_set *restrict readfds,
54507             fd_set *restrict writefds, fd_set *restrict errorfds,
54508             const struct timespec *restrict timeout,
54509             const sigset_t *restrict sigmask);
54510 int select(int nfds, fd_set *restrict readfds,
54511            fd_set *restrict writefds, fd_set *restrict errorfds,
54512            struct timeval *restrict timeout);
54513 void FD_CLR(int fd, fd_set *fdset);
54514 int FD_ISSET(int fd, const fd_set *fdset);
54515 void FD_SET(int fd, fd_set *fdset);
54516 void FD_ZERO(fd_set *fdset);
```

54517 **DESCRIPTION**

54518 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the  
 54519 *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for  
 54520 reading, are ready for writing, or have an exceptional condition pending, respectively.

54521 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- 54522 • For the *select()* function, the timeout period is given in seconds and microseconds in an  
 54523 argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is  
 54524 given in seconds and nanoseconds in an argument of type **struct timespec**.
- 54525 • The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when  
 54526 *sigmask* is a null pointer.
- 54527 • Upon successful completion, the *select()* function may modify the object pointed to by the  
 54528 *timeout* argument.

54529 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal  
 54530 devices, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()* on file descriptors that  
 54531 refer to other types of file is unspecified.

54532 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall  
 54533 be checked in each set; that is, the descriptors from zero through *nfds*-1 in the descriptor sets  
 54534 shall be examined.

54535 If the *readfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 54536 specifies the file descriptors to be checked for being ready to read, and on output indicates  
 54537 which file descriptors are ready to read.

54538 If the *writefds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 54539 specifies the file descriptors to be checked for being ready to write, and on output indicates  
 54540 which file descriptors are ready to write.

54541 If the *errorfds* argument is not a null pointer, it points to an object of type **fd\_set** that on input  
 54542 specifies the file descriptors to be checked for error conditions pending, and on output indicates  
 54543 which file descriptors have error conditions pending.

54544 Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to  
 54545 by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for  
 54546 reading, ready for writing, or have an error condition pending, respectively, and shall return the  
 54547 total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the

54548 corresponding bit shall be set upon successful completion if it was set on input and the  
54549 associated condition is true for that file descriptor.

54550 If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()*  
54551 function shall block until at least one of the requested operations becomes ready, until the  
54552 *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the  
54553 *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null  
54554 pointer, it specifies a maximum interval to wait for the selection to complete. If the specified  
54555 time interval expires without any requested operation becoming ready, the function shall return.  
54556 If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block  
54557 indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout*  
54558 parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

54559 The use of a timeout does not affect any pending timers set up by *alarm()*.

54560 Implementations may place limitations on the maximum timeout interval supported. All  
54561 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*  
54562 argument specifies a timeout interval greater than the implementation-defined maximum value,  
54563 the maximum value shall be used as the actual timeout value. Implementations may also place  
54564 limitations on the granularity of timeout intervals. If the requested timeout interval requires a  
54565 finer granularity than the implementation supports, the actual timeout interval shall be rounded  
54566 up to the next supported value.

54567 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the  
54568 caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall  
54569 restore the signal mask of the calling thread before returning. If a signal is unmasked as a result  
54570 of the signal mask being altered by *pselect()*, and a signal-catching function is called for that  
54571 signal during the execution of the *pselect()* function, and SA\_RESTART is clear for the  
54572 interrupting signal, then

- 54573 • If none of the selected file descriptors are ready, *pselect()* shall immediately fail with the  
54574 [EINTR] error after the signal-catching function returns.
- 54575 • If one or more of the selected file descriptors are ready, it is unspecified whether *pselect()*  
54576 behaves the same as if none of the descriptors were ready (failing with [EINTR] as above)  
54577 or behaves the same as if it was not interrupted (returning the total number of ready  
54578 descriptors).

54579 A descriptor shall be considered ready for reading when a call to an input function with  
54580 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
54581 successfully. (The function might return data, an end-of-file indication, or an error other than  
54582 one indicating that it is blocked, and in each of these cases the descriptor shall be considered  
54583 ready for reading.)

54584 A descriptor shall be considered ready for writing when a call to an output function with  
54585 O\_NONBLOCK clear would not block, whether or not the function would transfer data  
54586 successfully.

54587 If a socket has a pending error, it shall be considered to have an exceptional condition pending.  
54588 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for  
54589 use with a socket, it is protocol-specific except as noted below. For other file types it is  
54590 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or  
54591 *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate  
54592 that the descriptor has no exceptional condition pending.

54593 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with  
54594 parameters requesting normal and ancillary data, such that the presence of either type shall

54595 cause the socket to be marked as readable. The presence of out-of-band data shall be checked if  
 54596 the socket option SO\_OOBINLINE has been enabled, as out-of-band data is enqueued with  
 54597 normal data. If the socket is currently listening, then it shall be marked as readable if an  
 54598 incoming connection request has been received, and a call to the *accept()* or *accept4()* function  
 54599 shall complete without blocking.

54600 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying  
 54601 an amount of normal data equal to the current value of the SO\_SNDLOWAT option for the  
 54602 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the  
 54603 connection attempt has either succeeded or failed leaving a pending error, the socket shall be  
 54604 marked as writable.

54605 A socket shall be considered to have an exceptional condition pending if a receive operation  
 54606 with O\_NONBLOCK clear for the open file description and with the MSG\_OOB flag set would  
 54607 return out-of-band data without blocking. (It is protocol-specific whether the MSG\_OOB flag  
 54608 would be used to read out-of-band data.) A socket shall also be considered to have an  
 54609 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other  
 54610 circumstances under which a socket may be considered to have an exceptional condition  
 54611 pending are protocol-specific and implementation-defined.

54612 If the *readfds*, *writfds*, and *errorfds* arguments are all null pointers and the *timeout* argument is  
 54613 not a null pointer, the *pselect()* or *select()* function shall block for the time specified, or until  
 54614 interrupted by a signal. If the *readfds*, *writfds*, and *errorfds* arguments are all null pointers and  
 54615 the *timeout* argument is a null pointer, the *pselect()* or *select()* function shall block until  
 54616 interrupted by a signal.

54617 File descriptors associated with regular files shall always select true for ready to read, ready to  
 54618 write, and error conditions.

54619 On failure, the objects pointed to by the *readfds*, *writfds*, and *errorfds* arguments shall not be  
 54620 modified. If the timeout interval expires without the specified condition being true for any of the  
 54621 specified file descriptors, the objects pointed to by the *readfds*, *writfds*, and *errorfds* arguments  
 54622 shall have all bits set to 0.

54623 File descriptor masks of type **fd\_set** can be initialized and tested with *FD\_CLR()*, *FD\_ISSET()*,  
 54624 *FD\_SET()*, and *FD\_ZERO()*. It is unspecified whether each of these is a macro or a function. If a  
 54625 macro definition is suppressed in order to access an actual function, or a program defines an  
 54626 external identifier with any of these names, the behavior is undefined.

54627 *FD\_CLR(fd, fdsetp)* shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not  
 54628 a member of this set, there shall be no effect on the set, and this shall not be treated as an error.

54629 *FD\_ISSET(fd, fdsetp)* shall evaluate to non-zero if the file descriptor *fd* is a member of the set  
 54630 pointed to by *fdsetp*, and shall evaluate to zero otherwise.

54631 *FD\_SET(fd, fdsetp)* shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file  
 54632 descriptor *fd* is already in this set, there shall be no effect on the set, and this shall not be treated  
 54633 as an error.

54634 *FD\_ZERO(fdsetp)* shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is  
 54635 returned if the set is not empty at the time *FD\_ZERO()* is invoked.

54636 The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or  
 54637 equal to FD\_SETSIZE, or if *fd* is not a valid file descriptor, or if any of the arguments are  
 54638 expressions with side-effects.

54639 If a thread gets canceled during a *pselect()* call, the signal mask in effect when executing the  
 54640 registered cleanup functions is either the original signal mask or the signal mask installed as part

54641 of the *pselect()* call.

#### 54642 RETURN VALUE

54643 Upon successful completion, the *pselect()* and *select()* functions shall return the total number of  
54644 bits set in the bit masks. Otherwise,  $-1$  shall be returned, and *errno* shall be set to indicate the  
54645 error.

54646 *FD\_CLR()*, *FD\_SET()*, and *FD\_ZERO()* do not return a value. *FD\_ISSET()* shall return a non-  
54647 zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and  
54648 0 otherwise.

#### 54649 ERRORS

54650 Under the following conditions, *pselect()* and *select()* shall fail and set *errno* to:

54651 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a  
54652 valid open file descriptor.

54653 [EINTR] The function was interrupted by a signal.

54654 If SA\_RESTART has been set for the interrupting signal, it is implementation-  
54655 defined whether the function restarts or returns with [EINTR].

54656 [EINVAL] An invalid timeout interval was specified.

54657 [EINVAL] The *nfds* argument is less than 0 or greater than FD\_SETSIZE.

#### 54658 EXAMPLES

54659 None.

#### 54660 APPLICATION USAGE

54661 The use of *select()* and *pselect()* requires that the application construct the set of file descriptors  
54662 to work on each time through a polling loop, and is inherently limited from operating on file  
54663 descriptors larger than FD\_SETSIZE. Also, the amount of work to perform scales as *nfds*  
54664 increases, even if the number of file descriptors selected within the larger set remains the same.  
54665 Thus, applications may wish to consider using *poll()* and *ppoll()* instead, for better scaling.

54666 When a *pselect()* or *select()* call indicates a file descriptor is ready for reading, this means that if  
54667 an attempt to read data had been made at the time that the status of the file descriptor was  
54668 checked, it would have returned at least one byte of data, an end-of-file indication, or an error,  
54669 without blocking (even if O\_NONBLOCK is clear). When a *pselect()* or *select()* call indicates that  
54670 a file descriptor is ready for writing, this means that if an attempt to write one byte of data had  
54671 been made at the time that the status of the file descriptor was checked, it would have written  
54672 that byte or returned an error, without blocking. However, if an attempt to write more than one  
54673 byte had been made, it might have blocked (if O\_NONBLOCK is clear). In both cases, by the  
54674 time the call returns and a subsequent I/O operation is attempted, the state of the file descriptor  
54675 might have changed (for example, because another thread read or wrote some data) and, if  
54676 O\_NONBLOCK is clear, there is no guarantee that the operation will not block (unless it would  
54677 not block for some other reason, such as setting MIN=0 and TIME=0 for a terminal in non-  
54678 canonical mode). Therefore it is recommended that applications always set O\_NONBLOCK on  
54679 file descriptors whose readiness for I/O they query with *pselect()* or *select()*.

#### 54680 RATIONALE

54681 In earlier versions of the Single UNIX Specification, the *select()* function was defined in the  
54682 `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was  
54683 as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the  
54684 `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions.  
54685 Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>`  
54686 to include `<sys/select.h>`.

54687 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers  
 54688 can install an additional cancellation handler which resets the signal mask to the expected value.

```
54689 void cleanup(void *arg)
54690 {
54691     sigset_t *ss = (sigset_t *) arg;
54692     pthread_sigmask(SIG_SETMASK, ss, NULL);
54693 }
54694 int call_pselect(int nfd, fd_set *readfds, fd_set *writefds,
54695                fd_set errorfds, const struct timespec *timeout,
54696                const sigset_t *sigmask)
54697 {
54698     sigset_t oldmask;
54699     int result;
54700     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
54701     pthread_cleanup_push(cleanup, &oldmask);
54702     result = pselect(nfd, readfds, writefds, errorfds, timeout, sigmask);
54703     pthread_cleanup_pop(0);
54704     return result;
54705 }
```

#### 54706 FUTURE DIRECTIONS

54707 None.

#### 54708 SEE ALSO

54709 [accept\(\)](#), [alarm\(\)](#), [connect\(\)](#), [fcntl\(\)](#), [poll\(\)](#), [read\(\)](#), [recvmsg\(\)](#), [sendmsg\(\)](#), [write\(\)](#)

54710 XBD [<sys/select.h>](#), [<sys/time.h>](#)

#### 54711 CHANGE HISTORY

54712 First released in Issue 4, Version 2.

#### 54713 Issue 5

54714 Moved from X/OPEN UNIX extension to BASE.

54715 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when  
 54716 *nfd*s is less than 0 or greater than FD\_SETSIZE. It previously stated less than 0, or greater than or  
 54717 equal to FD\_SETSIZE.

54718 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.

#### 54719 Issue 6

54720 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs*  
 54721 in the *select()* DESCRIPTION to be *readfds* and *writefds*.

54722 Text referring to sockets is added to the DESCRIPTION.

54723 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
 54724 marked as part of the XSI STREAMS Option Group.

54725 The following new requirements on POSIX implementations derive from alignment with the  
 54726 Single UNIX Specification:

- 54727 • These functions are now mandatory.

54728 The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail  
 54729 related to sockets semantics is added to the DESCRIPTION.

54730 The *select()* function now requires inclusion of [<sys/select.h>](#).

- 54731 The **restrict** keyword is added to the *select()* prototype for alignment with the  
54732 ISO/IEC 9899:1999 standard.
- 54733 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/70 is applied, updating the  
54734 DESCRIPTION to reference the signal mask in terms of the calling thread rather than the  
54735 process.
- 54736 **Issue 7**
- 54737 SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled  
54738 during a call to *pselect()*, and adding example code to the RATIONALE.
- 54739 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 54740 Functionality relating to the Threads option is moved to the Base.
- 54741 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0446 [372] is applied.
- 54742 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0259 [680] is applied.
- 54743 **Issue 8**
- 54744 Austin Group Defect 220 is applied, adding `const` to the second parameter of *FD\_ISSET()*.
- 54745 Austin Group Defect 411 is applied, adding *accept4()*.
- 54746 Austin Group Defect 1186 is applied, clarifying the behavior when the *pselect()* function is  
54747 interrupted by a signal.
- 54748 Austin Group Defect 1263 is applied, changing the APPLICATION USAGE section.
- 54749 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 54750 Austin Group Defect 1448 is applied, changing the APPLICATION USAGE section.



54751 **NAME**

54752 psiginfo, psignal — write signal information to standard error

54753 **SYNOPSIS**

```
54754 CX #include <signal.h>
54755 void psiginfo(const siginfo_t *pinfo, const char *message);
54756 void psignal(int signum, const char *message);
```

54757 **DESCRIPTION**

54758 The *psiginfo()* and *psignal()* functions shall write a language-dependent message associated with  
 54759 a signal number to the standard error stream as follows:

- 54760 • First, if *message* is not a null pointer and is not the empty string, the string pointed to by the  
 54761 *message* argument shall be written, followed by a <colon> and a <space>.
- 54762 • Then the signal description string associated with *signum* or with the signal indicated by  
 54763 *pinfo* shall be written, followed by a <newline>.

54764 For *psiginfo()*, the application shall ensure that the argument *pinfo* references a valid **siginfo\_t**  
 54765 structure. For *psignal()*, if *signum* is not a valid signal number, the behavior is implementation-  
 54766 defined.

54767 The *psiginfo()* and *psignal()* functions shall not change the orientation of the standard error  
 54768 stream.

54769 The *psiginfo()* and *psignal()* functions shall mark for update the last data modification and last  
 54770 file status change timestamps of the file associated with the standard error stream at some time  
 54771 between their successful completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()*  
 54772 on *stderr*.

54773 The *psiginfo()* and *psignal()* functions shall not change the setting of *errno* if successful.

54774 On error, the *psiginfo()* and *psignal()* functions shall set the error indicator for the stream to  
 54775 which *stderr* points, and shall set *errno* to indicate the error.

54776 Since no value is returned, an application wishing to check for error situations should set *errno*  
 54777 to 0, then call *psiginfo()* or *psignal()*, then check *errno*.

54778 **RETURN VALUE**

54779 These functions shall not return a value.

54780 **ERRORS**

54781 Refer to *fputc()*.

54782 **EXAMPLES**

54783 None.

54784 **APPLICATION USAGE**

54785 As an alternative to setting *errno* to zero before the call and checking if it is non-zero afterwards,  
 54786 applications can use *ferror()* to detect whether *psiginfo()* or *psignal()* encountered an error.

54787 An application wishing to use this method to check for error situations should call  
 54788 *clearerr(stderr)* before calling *psiginfo()* or *psignal()*, then if *ferror(stderr)* returns non-zero, the  
 54789 value of *errno* indicates which error occurred.

54790 **RATIONALE**

54791 System V historically has *psignal()* and *psiginfo()* in `<siginfo.h>`. However, the `<siginfo.h>`  
54792 header is not specified in the Base Definitions volume of POSIX.1-2024, and the type `siginfo_t` is  
54793 defined in `<signal.h>`.

54794 **FUTURE DIRECTIONS**

54795 None.

54796 **SEE ALSO**

54797 *fputc()*, *perror()*, *strsignal()*

54798 XBD `<signal.h>`

54799 **CHANGE HISTORY**

54800 First released in Issue 7.

54801 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0447 [399,428], XSH/TC1-2008/0448  
54802 [399], and XSH/TC1-2008/0449 [399,401] are applied.

54803 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0260 [629] is applied.

54804 **NAME**

54805 pthread\_atfork — register fork handlers

54806 **SYNOPSIS**

```
54807 OB #include <pthread.h>
54808 int pthread_atfork(void (*prepare)(void), void (*parent)(void),
54809 void (*child)(void));
```

54810 **DESCRIPTION**

54811 The *pthread\_atfork()* function shall declare fork handlers to be called before and after *fork()*, in  
 54812 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*  
 54813 processing commences. The *parent* fork handler shall be called after *fork()* processing completes  
 54814 in the parent process. The *child* fork handler shall be called after *fork()* processing completes in  
 54815 the child process. If no handling is desired at one or more of these three points, the  
 54816 corresponding fork handler address(es) may be set to NULL.

54817 If a *fork()* call in a multi-threaded process leads to a *child* fork handler calling any function that is  
 54818 not async-signal-safe, the behavior is undefined.

54819 The order of calls to *pthread\_atfork()* is significant. The *parent* and *child* fork handlers shall be  
 54820 called in the order in which they were established by calls to *pthread\_atfork()*. The *prepare* fork  
 54821 handlers shall be called in the opposite order.

54822 **RETURN VALUE**

54823 Upon successful completion, *pthread\_atfork()* shall return a value of zero; otherwise, an error  
 54824 number shall be returned to indicate the error.

54825 **ERRORS**

54826 The *pthread\_atfork()* function shall fail if:

54827 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

54828 The *pthread\_atfork()* function shall not return an error code of [EINTR].

54829 **EXAMPLES**

54830 None.

54831 **APPLICATION USAGE**

54832 The original usage pattern envisaged for *pthread\_atfork()* was for the *prepare* fork handler to lock  
 54833 mutexes and other locks, and for the *parent* and *child* handlers to unlock them. However, since all  
 54834 of the relevant unlocking functions, except *sem\_post()*, are not async-signal-safe, this usage  
 54835 results in undefined behavior in the child process unless the only such unlocking function it calls  
 54836 is *sem\_post()*.

54837 **RATIONALE**

54838 There are at least two serious problems with the semantics of *fork()* in a multi-threaded  
 54839 program. One problem has to do with state (for example, memory) covered by mutexes.  
 54840 Consider the case where one thread has a mutex locked and the state covered by that mutex is  
 54841 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state  
 54842 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply  
 54843 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about  
 54844 how to correct or otherwise deal with the inconsistent state in the child.

54845 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the  
 54846 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe  
 54847 library routines are promised to be available.

54848 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application  
54849 programs may not be aware that a multi-threaded library is in use, and they feel free to call any  
54850 number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed,  
54851 they may be extant single-threaded programs and cannot, therefore, be expected to obey new  
54852 restrictions imposed by the threads library.

54853 On the other hand, the multi-threaded library needs a way to protect its internal state during  
54854 *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-  
54855 threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to  
54856 effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may  
54857 entail simply resetting the state in the child after the *fork()* processing completes.

54858 The *pthread\_atfork()* function was intended to provide multi-threaded libraries with a means to  
54859 protect themselves from innocent application programs that call *fork()*, and to provide multi-  
54860 threaded application programs with a standard mechanism for protecting themselves from  
54861 *fork()* calls in a library routine or the application itself.

54862 The expected usage was that the prepare handler would acquire all mutex locks and the other  
54863 two fork handlers would release them.

54864 For example, an application could have supplied a prepare routine that acquires the necessary  
54865 mutexes the library maintains and supplied child and parent routines that release those  
54866 mutexes, thus ensuring that the child would have got a consistent snapshot of the state of the  
54867 library (and that no mutexes would have been left stranded). This is good in theory, but in  
54868 reality not practical. Each and every mutex and lock in the process must be located and locked.  
54869 Every component of a program including third-party components must participate and they  
54870 must agree who is responsible for which mutex or lock. This is especially problematic for  
54871 mutexes and locks in dynamically allocated memory. All mutexes and locks internal to the  
54872 implementation must be locked, too. This possibly delays the thread calling *fork()* for a long  
54873 time or even indefinitely since uses of these synchronization objects may not be under control of  
54874 the application. A final problem to mention here is the problem of locking streams. At least the  
54875 streams under control of the system (like *stdin*, *stdout*, *stderr*) must be protected by locking the  
54876 stream with *flockfile()*. But the application itself could have done that, possibly in the same  
54877 thread calling *fork()*. In this case, the process will deadlock.

54878 Alternatively, some libraries might have been able to supply just a *child* routine that reinitializes  
54879 the mutexes in the library and all associated states to some known value (for example, what it  
54880 was when the image was originally executed). This approach is not possible, though, because  
54881 implementations are allowed to fail *\*\_init()* and *\*\_destroy()* calls for mutexes and locks if the  
54882 mutex or lock is still locked. In this case, the *child* routine is not able to reinitialize the mutexes  
54883 and locks.

54884 When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization  
54885 variables remain in the same state in the child as they were in the parent at the time *fork()* was  
54886 called. Thus, for example, mutex locks may be held by threads that no longer exist in the child  
54887 process, and any associated states may be inconsistent. The intention was that the parent process  
54888 could have avoided this by explicit code that acquires and releases locks critical to the child via  
54889 *pthread\_atfork()*. In addition, any critical threads would have needed to be recreated and  
54890 reinitialized to the proper state in the child (also via *pthread\_atfork()*).

54891 A higher-level package may acquire locks on its own data structures before invoking lower-level  
54892 packages. Under this scenario, the order specified for fork handler calls allows a simple rule of  
54893 initialization for avoiding package deadlock: a package initializes all packages on which it  
54894 depends before it calls the *pthread\_atfork()* function for itself.

54895 As explained, there is no suitable solution for functionality which requires non-atomic

54896 operations to be protected through mutexes and locks. This is why the POSIX.1 standard since  
54897 the 1996 release requires that the child process after *fork()* in a multi-threaded process only calls  
54898 *async-signal-safe* interfaces.

54899 An additional problem arises when *pthread\_atfork()* is called to register a function in a library  
54900 that was loaded using *dlopen()*. If the library is unloaded using *dlclose()*, and the  
54901 implementation of *dlclose()* does not unregister the function, then when *fork()* tries to call it the  
54902 result will be undefined behavior. Some implementations of *dlclose()* do unregister  
54903 *pthread\_atfork()* handlers, but this cannot be relied upon by portable applications. The standard  
54904 provides no portable method for unregistering a function installed as a handler via  
54905 *pthread\_atfork()*.

#### 54906 **FUTURE DIRECTIONS**

54907 The *pthread\_atfork()* function may be removed in a future version of this standard.

#### 54908 **SEE ALSO**

54909 *atexit()*, *exec*, *fork()*

54910 XBD [<pthread.h>](#), [<sys/types.h>](#)

#### 54911 **CHANGE HISTORY**

54912 First released in Issue 5. Derived from the POSIX Threads Extension.

54913 IEEE PASC Interpretation 1003.1c #4 is applied.

#### 54914 **Issue 6**

54915 The *pthread\_atfork()* function is marked as part of the Threads option.

54916 The [<pthread.h>](#) header is added to the SYNOPSIS.

#### 54917 **Issue 7**

54918 The *pthread\_atfork()* function is moved from the Threads option to the Base.

54919 SD5-XSH-ERN-145 is applied, updating the RATIONALE.

54920 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0261 [858] is applied.

#### 54921 **Issue 8**

54922 Austin Group Defect 851 is applied, marking *pthread\_atfork()* as obsolescent.

54923 **NAME**

54924 pthread\_attr\_destroy, pthread\_attr\_init — destroy and initialize the thread attributes object

54925 **SYNOPSIS**

54926 #include &lt;pthread.h&gt;

54927 int pthread\_attr\_destroy(pthread\_attr\_t \*attr);

54928 int pthread\_attr\_init(pthread\_attr\_t \*attr);

54929 **DESCRIPTION**

54930 The *pthread\_attr\_destroy()* function shall destroy a thread attributes object. An implementation  
54931 may cause *pthread\_attr\_destroy()* to set *attr* to an implementation-defined invalid value. A  
54932 destroyed *attr* attributes object can be reinitialized using *pthread\_attr\_init()*; the results of  
54933 otherwise referencing the object after it has been destroyed are undefined.

54934 The *pthread\_attr\_init()* function shall initialize a thread attributes object *attr* with the default  
54935 value for all of the individual attributes used by a given implementation.

54936 The resulting attributes object (possibly modified by setting individual attribute values) when  
54937 used by *pthread\_create()* defines the attributes of the thread created. A single attributes object can  
54938 be used in multiple simultaneous calls to *pthread\_create()*. Results are undefined if  
54939 *pthread\_attr\_init()* is called specifying an already initialized *attr* attributes object.

54940 The behavior is undefined if the value specified by the *attr* argument to *pthread\_attr\_destroy()*  
54941 does not refer to an initialized thread attributes object.

54942 **RETURN VALUE**

54943 Upon successful completion, *pthread\_attr\_destroy()* and *pthread\_attr\_init()* shall return a value of  
54944 0; otherwise, an error number shall be returned to indicate the error.

54945 **ERRORS**54946 The *pthread\_attr\_init()* function shall fail if:

54947 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

54948 These functions shall not return an error code of [EINTR].

54949 **EXAMPLES**

54950 None.

54951 **APPLICATION USAGE**

54952 None.

54953 **RATIONALE**

54954 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
54955 support probable future standardization in these areas without requiring that the function itself  
54956 be changed.

54957 Attributes objects provide clean isolation of the configurable aspects of threads. For example,  
54958 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When  
54959 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects  
54960 can help by allowing the changes to be isolated in a single place, rather than being spread across  
54961 every instance of thread creation.

54962 Attributes objects can be used to set up “classes” of threads with similar attributes; for example,  
54963 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
54964 can be defined in a single place and then referenced wherever threads need to be created.  
54965 Changes to “class” decisions become straightforward, and detailed analysis of each  
54966 *pthread\_create()* call is not required.

54967 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
54968 been specified as structures, adding new attributes would force recompilation of all multi-  
54969 threaded programs when the attributes objects are extended; this might not be possible if  
54970 different program components were supplied by different vendors.

54971 Additionally, opaque attributes objects present opportunities for improving performance.  
54972 Argument validity can be checked once when attributes are set, rather than each time a thread is  
54973 created. Implementations often need to cache kernel objects that are expensive to create.  
54974 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
54975 invalid due to attribute changes.

54976 Since assignment is not necessarily defined on a given opaque type, implementation-defined  
54977 default values cannot be defined in a portable way. The solution to this problem is to allow  
54978 attributes objects to be initialized dynamically by attributes object initialization functions, so that  
54979 default values can be supplied automatically by the implementation.

54980 The following proposal was provided as a suggested alternative to the supplied attributes:

- 54981 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
54982 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
54983 parameter containing the flags should be an opaque type for extensibility. If no flags are  
54984 set in the parameter, then the objects are created with default characteristics. An  
54985 implementation may specify implementation-defined flag values and associated  
54986 behavior.
- 54987 2. If further specialization of mutexes and condition variables is necessary, implementations  
54988 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
54989 **pthread\_cond\_t** objects (instead of on attributes objects).

54990 The difficulties with this solution are:

- 54991 1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using  
54992 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
54993 application programmers need to know the location of each bit. If bits are set or read by  
54994 encapsulation (that is, *get* and *set* functions), then the bitmask is merely an  
54995 implementation of attributes objects as currently defined and should not be exposed to  
54996 the programmer.
- 54997 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
54998 policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking  
54999 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,  
55000 the bitmask can only reasonably control whether particular attributes are set or not, and it  
55001 cannot serve as the repository of the value itself. The value needs to be specified as a  
55002 function parameter (which is non-extensible), or by setting a structure field (which is non-  
55003 opaque), or by *get* and *set* functions (making the bitmask a redundant addition to the  
55004 attributes objects).

55005 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
55006 machine-dependent. Some implementations may not be able to change the size of the stack, for  
55007 example, and others may not need to because stack pages may be discontinuous and can be  
55008 allocated and released on demand.

55009 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
55010 to the attribute mechanism or to any attributes object defined in this volume of POSIX.1-2024 has  
55011 to be done with care so as not to affect binary-compatibility.

55012 Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,

55013 may have their size fixed at compile time. This means, for example, a *pthread\_create()* in an  
55014 implementation with extensions to **pthread\_attr\_t** cannot look beyond the area that the binary  
55015 application assumes is valid. This suggests that implementations should maintain a size field in  
55016 the attributes object, as well as possibly version information, if extensions in different directions  
55017 (possibly by different vendors) are to be accommodated.

55018 If an implementation detects that the value specified by the *attr* argument to  
55019 *pthread\_attr\_destroy()* does not refer to an initialized thread attributes object, it is recommended  
55020 that the function should fail and report an [EINVAL] error.

55021 If an implementation detects that the value specified by the *attr* argument to *pthread\_attr\_init()*  
55022 refers to an already initialized thread attributes object, it is recommended that the function  
55023 should fail and report an [EBUSY] error.

#### 55024 FUTURE DIRECTIONS

55025 None.

#### 55026 SEE ALSO

55027 [\*pthread\\_attr\\_getstacksize\(\)\*](#), [\*pthread\\_attr\\_getdetachstate\(\)\*](#), [\*pthread\\_create\(\)\*](#)

55028 XBD <[pthread.h](#)>

#### 55029 CHANGE HISTORY

55030 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 55031 Issue 6

55032 The *pthread\_attr\_destroy()* and *pthread\_attr\_init()* functions are marked as part of the Threads  
55033 option.

55034 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already  
55035 initialized thread attributes object is undefined.

55036 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/71 is applied, updating the ERRORS  
55037 section to add the optional [EINVAL] error for the *pthread\_attr\_destroy()* function, and the  
55038 optional [EBUSY] error for the *pthread\_attr\_init()* function.

#### 55039 Issue 7

55040 The *pthread\_attr\_destroy()* and *pthread\_attr\_init()* functions are moved from the Threads option  
55041 to the Base.

55042 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55043 results in undefined behavior.

55044 The [EBUSY] error for an already initialized thread attributes object is removed; this condition  
55045 results in undefined behavior.



55046 **NAME**

55047 pthread\_attr\_getdetachstate, pthread\_attr\_setdetachstate — get and set the detachstate attribute

55048 **SYNOPSIS**

```
55049 #include <pthread.h>
55050 int pthread_attr_getdetachstate(const pthread_attr_t *attr,
55051 int *detachstate);
55052 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

55053 **DESCRIPTION**

55054 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread  
 55055 is created detached, then use of the ID of the newly created thread by the *pthread\_detach()* or  
 55056 *pthread\_join()* function is an error.

55057 The *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* functions, respectively, shall get  
 55058 and set the *detachstate* attribute in the *attr* object.

55059 For *pthread\_attr\_getdetachstate()*, *detachstate* shall be set to either  
 55060 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

55061 For *pthread\_attr\_setdetachstate()*, the application shall set *detachstate* to either  
 55062 PTHREAD\_CREATE\_DETACHED or PTHREAD\_CREATE\_JOINABLE.

55063 A value of PTHREAD\_CREATE\_DETACHED shall cause all threads created with *attr* to be in  
 55064 the detached state, whereas using a value of PTHREAD\_CREATE\_JOINABLE shall cause all  
 55065 threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute  
 55066 shall be PTHREAD\_CREATE\_JOINABLE.

55067 The behavior is undefined if the value specified by the *attr* argument to  
 55068 *pthread\_attr\_getdetachstate()* or *pthread\_attr\_setdetachstate()* does not refer to an initialized thread  
 55069 attributes object.

55070 **RETURN VALUE**

55071 Upon successful completion, *pthread\_attr\_getdetachstate()* and *pthread\_attr\_setdetachstate()* shall  
 55072 return a value of 0; otherwise, an error number shall be returned to indicate the error.

55073 The *pthread\_attr\_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*  
 55074 if successful.

55075 **ERRORS**

55076 The *pthread\_attr\_setdetachstate()* function shall fail if:

55077 [EINVAL] The value of *detachstate* was not valid

55078 These functions shall not return an error code of [EINTR].

55079 **EXAMPLES**55080 **Retrieving the detachstate Attribute**

55081 This example shows how to obtain the *detachstate* attribute of a thread attribute object.

```
55082 #include <pthread.h>
55083 pthread_attr_t thread_attr;
55084 int detachstate;
55085 int rc;
55086 /* code initializing thread_attr */
55087 ...
```

```
55088     rc = pthread_attr_getdetachstate (&thread_attr, &detachstate);
55089     if (rc!=0) {
55090         /* handle error */
55091         ...
55092     }
55093     else {
55094         /* legal values for detachstate are:
55095          * PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE
55096          */
55097         ...
55098     }
```

**55099 APPLICATION USAGE**

55100 None.

**55101 RATIONALE**

55102 If an implementation detects that the value specified by the *attr* argument to  
55103 *pthread\_attr\_getdetachstate()* or *pthread\_attr\_setdetachstate()* does not refer to an initialized thread  
55104 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

**55105 FUTURE DIRECTIONS**

55106 None.

**55107 SEE ALSO**

55108 [pthread\\_attr\\_destroy\(\)](#), [pthread\\_attr\\_getstacksize\(\)](#), [pthread\\_create\(\)](#)

55109 XBD <[pthread.h](#)>

**55110 CHANGE HISTORY**

55111 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**55112 Issue 6**

55113 The *pthread\_attr\_setdetachstate()* and *pthread\_attr\_getdetachstate()* functions are marked as part of  
55114 the Threads option.

55115 The normative text is updated to avoid use of the term “must” for application requirements.

55116 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/72 is applied, adding the example to the  
55117 EXAMPLES section.

55118 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/73 is applied, updating the ERRORS  
55119 section to include the optional [EINVAL] error.

**55120 Issue 7**

55121 The *pthread\_attr\_setdetachstate()* and *pthread\_attr\_getdetachstate()* functions are moved from the  
55122 Threads option to the Base.

55123 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55124 results in undefined behavior.

55125 **NAME**

55126 pthread\_attr\_getguardsize, pthread\_attr\_setguardsize — get and set the thread guardsize  
55127 attribute

55128 **SYNOPSIS**

```
55129 #include <pthread.h>
55130 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
55131                               size_t *restrict guardsize);
55132 int pthread_attr_setguardsize(pthread_attr_t *attr,
55133                               size_t guardsize);
```

55134 **DESCRIPTION**

55135 The *pthread\_attr\_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This  
55136 attribute shall be returned in the *guardsize* parameter.

55137 The *pthread\_attr\_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new  
55138 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard  
55139 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard  
55140 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

55141 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The  
55142 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is  
55143 created with guard protection, the implementation allocates extra memory at the overflow end  
55144 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows  
55145 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

55146 A conforming implementation may round up the value contained in *guardsize* to a multiple of  
55147 the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation  
55148 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread\_attr\_getguardsize()*  
55149 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous  
55150 *pthread\_attr\_setguardsize()* function call.

55151 The default value of the *guardsize* attribute is implementation-defined.

55152 If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread  
55153 stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the  
55154 implementation. It is the responsibility of the application to manage stack overflow along with  
55155 stack allocation and management in this case.

55156 The behavior is undefined if the value specified by the *attr* argument to  
55157 *pthread\_attr\_getguardsize()* or *pthread\_attr\_setguardsize()* does not refer to an initialized thread  
55158 attributes object.

55159 **RETURN VALUE**

55160 If successful, the *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions shall return  
55161 zero; otherwise, an error number shall be returned to indicate the error.

55162 **ERRORS**

55163 These functions shall fail if:

55164 [EINVAL] The parameter *guardsize* is invalid.

55165 These functions shall not return an error code of [EINTR].

55166 **EXAMPLES**55167 **Retrieving the guardsize Attribute**

55168 This example shows how to obtain the *guardsize* attribute of a thread attribute object.

```
55169 #include <pthread.h>
55170 pthread_attr_t thread_attr;
55171 size_t guardsize;
55172 int rc;
55173 /* code initializing thread_attr */
55174 ...
55175 rc = pthread_attr_getguardsize (&thread_attr, &guardsize);
55176 if (rc != 0) {
55177     /* handle error */
55178     ...
55179 }
55180 else {
55181     if (guardsize > 0) {
55182         /* a guard area of at least guardsize bytes is provided */
55183         ...
55184     }
55185     else {
55186         /* no guard area provided */
55187         ...
55188     }
55189 }
```

55190 **APPLICATION USAGE**

55191 None.

55192 **RATIONALE**

55193 The *guardsize* attribute is provided to the application for two reasons:

- 55194 1. Overflow protection can potentially result in wasted system resources. An application  
55195 that creates a large number of threads, and which knows its threads never overflow their  
55196 stack, can save system resources by turning off guard areas.
- 55197 2. When threads allocate large data structures on the stack, large guard areas may be needed  
55198 to detect stack overflow.

55199 The default size of the guard area is left implementation-defined since on systems supporting  
55200 very large page sizes, the overhead might be substantial if at least one guard page is required by  
55201 default.

55202 If an implementation detects that the value specified by the *attr* argument to  
55203 *pthread\_attr\_getguardsize()* or *pthread\_attr\_setguardsize()* does not refer to an initialized thread  
55204 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

55205 **FUTURE DIRECTIONS**

55206 None.

55207 **SEE ALSO**

55208 XBD &lt;pthread.h&gt;, &lt;sys/mman.h&gt;

55209 **CHANGE HISTORY**

55210 First released in Issue 5.

55211 **Issue 6**55212 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the  
55213 second error condition.55214 The **restrict** keyword is added to the *pthread\_attr\_getguardsize()* prototype for alignment with the  
55215 ISO/IEC 9899:1999 standard.55216 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/74 is applied, updating the ERRORS  
55217 section to remove the [EINVAL] error ("The attribute *attr* is invalid."), and replacing it with the  
55218 optional [EINVAL] error.55219 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/76 is applied, adding the example to the  
55220 EXAMPLES section.55221 **Issue 7**55222 SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.55223 SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the  
55224 guard area is implementation-defined.55225 The *pthread\_attr\_getguardsize()* and *pthread\_attr\_setguardsize()* functions are moved from the XSI  
55226 option to the Base.55227 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55228 results in undefined behavior.

55229 **NAME**

55230 pthread\_attr\_getinheritsched, pthread\_attr\_setinheritsched — get and set the inheritsched  
55231 attribute (**REALTIME THREADS**)

55232 **SYNOPSIS**

```
55233 TPS #include <pthread.h>
55234 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
55235 int *restrict inheritsched);
55236 int pthread_attr_setinheritsched(pthread_attr_t *attr,
55237 int inheritsched);
```

55238 **DESCRIPTION**

55239 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions, respectively, shall  
55240 get and set the *inheritsched* attribute in the *attr* argument.

55241 When the attributes objects are used by *pthread\_create()*, the *inheritsched* attribute determines  
55242 how the other scheduling attributes of the created thread shall be set.

55243 The supported values of *inheritsched* shall be:

55244 **PTHREAD\_INHERIT\_SCHED**

55245 Specifies that the thread scheduling attributes shall be inherited from the creating thread,  
55246 and the scheduling attributes in this *attr* argument shall be ignored.

55247 **PTHREAD\_EXPLICIT\_SCHED**

55248 Specifies that the thread scheduling attributes shall be set to the corresponding values from  
55249 this attributes object.

55250 The symbols **PTHREAD\_INHERIT\_SCHED** and **PTHREAD\_EXPLICIT\_SCHED** are defined in  
55251 the **<pthread.h>** header.

55252 The following thread scheduling attributes defined by POSIX.1-2024 are affected by the  
55253 *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and  
55254 scheduling contention scope (*contentionscope*).

55255 The behavior is undefined if the value specified by the *attr* argument to  
55256 *pthread\_attr\_getinheritsched()* or *pthread\_attr\_setinheritsched()* does not refer to an initialized  
55257 thread attributes object.

55258 **RETURN VALUE**

55259 If successful, the *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions shall  
55260 return zero; otherwise, an error number shall be returned to indicate the error.

55261 **ERRORS**

55262 The *pthread\_attr\_setinheritsched()* function shall fail if:

55263 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

55264 The *pthread\_attr\_setinheritsched()* function may fail if:

55265 [EINVAL] The value of *inheritsched* is not valid.

55266 These functions shall not return an error code of [EINTR].

55267 **EXAMPLES**

55268 None.

55269 **APPLICATION USAGE**55270 After these attributes have been set, a thread can be created with the specified attributes using  
55271 *pthread\_create()*. Using these routines does not affect the current running thread.55272 See [Section 2.9.4](#) (on page 540) for further details on thread scheduling attributes and their  
55273 default settings.55274 **RATIONALE**55275 If an implementation detects that the value specified by the *attr* argument to  
55276 *pthread\_attr\_getinheritsched()* or *pthread\_attr\_setinheritsched()* does not refer to an initialized  
55277 thread attributes object, it is recommended that the function should fail and report an [EINVAL]  
55278 error.55279 **FUTURE DIRECTIONS**

55280 None.

55281 **SEE ALSO**55282 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getschedpolicy()*,  
55283 *pthread\_attr\_getschedparam()*, *pthread\_create()*55284 XBD [<pthread.h>](#), [<sched.h>](#)55285 **CHANGE HISTORY**

55286 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55287 Marked as part of the Realtime Threads Feature Group.

55288 **Issue 6**55289 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions are marked as part  
55290 of the Threads and Thread Execution Scheduling options.55291 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
55292 implementation does not support the Thread Execution Scheduling option.55293 The **restrict** keyword is added to the *pthread\_attr\_getinheritsched()* prototype for alignment with  
55294 the ISO/IEC 9899:1999 standard.55295 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/75 is applied, clarifying the values of  
55296 *inheritsched* in the DESCRIPTION and adding two optional [EINVAL] errors to the ERRORS  
55297 section for checking when *attr* refers to an uninitialized thread attribute object.55298 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/77 is applied, adding a reference to  
55299 [Section 2.9.4](#) (on page 540) in the APPLICATION USAGE section.55300 **Issue 7**55301 The *pthread\_attr\_getinheritsched()* and *pthread\_attr\_setinheritsched()* functions are marked only as  
55302 part of the Thread Execution Scheduling option as the Threads option is now part of the Base.55303 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55304 results in undefined behavior.

55305 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0450 [314] is applied.

55306 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0262 [757] is applied.

55307 **NAME**

55308 pthread\_attr\_getschedparam, pthread\_attr\_setschedparam — get and set the schedparam  
55309 attribute

55310 **SYNOPSIS**

```
55311 #include <pthread.h>
55312 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
55313     struct sched_param *restrict param);
55314 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
55315     const struct sched_param *restrict param);
```

55316 **DESCRIPTION**

55317 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions, respectively, shall  
55318 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*  
55319 structure are defined in the **<sched.h>** header. For the SCHED\_FIFO and SCHED\_RR policies,  
55320 the only required member of *param* is *sched\_priority*.

55321 TSP For the SCHED\_SPORADIC policy, the required members of the *param* structure are  
55322 *sched\_priority*, *sched\_ss\_low\_priority*, *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, and  
55323 *sched\_ss\_max\_repl*. The specified *sched\_ss\_repl\_period* needs to be greater than or equal to the  
55324 specified *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.  
55325 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
55326 function to succeed; if not, the function shall fail. It is unspecified whether the  
55327 *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values are stored as provided by this function or are  
55328 rounded to align with the resolution of the clock being used.

55329 The behavior is undefined if the value specified by the *attr* argument to  
55330 *pthread\_attr\_getschedparam()* or *pthread\_attr\_setschedparam()* does not refer to an initialized thread  
55331 attributes object.

55332 **RETURN VALUE**

55333 If successful, the *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions shall  
55334 return zero; otherwise, an error number shall be returned to indicate the error.

55335 **ERRORS**

55336 The *pthread\_attr\_setschedparam()* function shall fail if:

55337 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

55338 The *pthread\_attr\_setschedparam()* function may fail if:

55339 [EINVAL] The value of *param* is not valid.

55340 These functions shall not return an error code of [EINTR].

55341 **EXAMPLES**

55342 None.

55343 **APPLICATION USAGE**

55344 After these attributes have been set, a thread can be created with the specified attributes using  
55345 *pthread\_create()*. Using these routines does not affect the current running thread.

55346 **RATIONALE**

55347 If an implementation detects that the value specified by the *attr* argument to  
55348 *pthread\_attr\_getschedparam()* or *pthread\_attr\_setschedparam()* does not refer to an initialized thread  
55349 attributes object, it is recommended that the function should fail and report an [EINVAL] error.



55350 **FUTURE DIRECTIONS**

55351 None.

55352 **SEE ALSO**55353 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
55354 *pthread\_attr\_getschedpolicy()*, *pthread\_create()*

55355 XBD &lt;pthread.h&gt;, &lt;sched.h&gt;

55356 **CHANGE HISTORY**

55357 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55358 **Issue 6**55359 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions are marked as part  
55360 of the Threads option.

55361 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

55362 The **restrict** keyword is added to the *pthread\_attr\_getschedparam()* and  
55363 *pthread\_attr\_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.55364 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/78 is applied, updating the ERRORS  
55365 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
55366 object.55367 **Issue 7**55368 The *pthread\_attr\_getschedparam()* and *pthread\_attr\_setschedparam()* functions are moved from the  
55369 Threads option to the Base.55370 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements  
55371 for the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values.55372 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55373 results in undefined behavior.

55374 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0451 [314] is applied.

55375 **NAME**

55376 pthread\_attr\_getschedpolicy, pthread\_attr\_setschedpolicy — get and set the schedpolicy  
55377 attribute (**REALTIME THREADS**)

55378 **SYNOPSIS**

```
55379 TPS #include <pthread.h>
55380 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
55381 int *restrict policy);
55382 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

55383 **DESCRIPTION**

55384 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions, respectively, shall  
55385 get and set the *schedpolicy* attribute in the *attr* argument.

55386 The supported values of *policy* shall include SCHED\_FIFO, SCHED\_RR, and SCHED\_OTHER,  
55387 which are defined in the **<sched.h>** header. When threads executing with the scheduling policy  
55388 TSP SCHED\_FIFO, SCHED\_RR, or SCHED\_SPORADIC are waiting on a mutex, they shall acquire  
55389 the mutex in priority order when the mutex is unlocked.

55390 The behavior is undefined if the value specified by the *attr* argument to  
55391 *pthread\_attr\_getschedpolicy()* or *pthread\_attr\_setschedpolicy()* does not refer to an initialized thread  
55392 attributes object.

55393 **RETURN VALUE**

55394 If successful, the *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions shall  
55395 return zero; otherwise, an error number shall be returned to indicate the error.

55396 **ERRORS**

55397 The *pthread\_attr\_setschedpolicy()* function shall fail if:

55398 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

55399 The *pthread\_attr\_setschedpolicy()* function may fail if:

55400 [EINVAL] The value of *policy* is not valid.

55401 These functions shall not return an error code of [EINTR].

55402 **EXAMPLES**

55403 None.

55404 **APPLICATION USAGE**

55405 After these attributes have been set, a thread can be created with the specified attributes using  
55406 *pthread\_create()*. Using these routines does not affect the current running thread.

55407 See [Section 2.9.4](#) (on page 540) for further details on thread scheduling attributes and their  
55408 default settings.

55409 **RATIONALE**

55410 If an implementation detects that the value specified by the *attr* argument to  
55411 *pthread\_attr\_getschedpolicy()* or *pthread\_attr\_setschedpolicy()* does not refer to an initialized thread  
55412 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

55413 **FUTURE DIRECTIONS**

55414 None.

55415 **SEE ALSO**

55416 *pthread\_attr\_destroy()*, *pthread\_attr\_getscope()*, *pthread\_attr\_getinheritsched()*,  
55417 *pthread\_attr\_getschedparam()*, *pthread\_create()*

55418 XBD <pthread.h>, <sched.h>

55419 **CHANGE HISTORY**

55420 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55421 Marked as part of the Realtime Threads Feature Group.

55422 **Issue 6**

55423 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions are marked as part of  
55424 the Threads and Thread Execution Scheduling options.

55425 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
55426 implementation does not support the Thread Execution Scheduling option.

55427 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

55428 The **restrict** keyword is added to the *pthread\_attr\_getschedpolicy()* prototype for alignment with  
55429 the ISO/IEC 9899:1999 standard.

55430 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/79 is applied, adding a reference to  
55431 [Section 2.9.4](#) (on page 540) in the APPLICATION USAGE section.

55432 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/80 is applied, updating the ERRORS  
55433 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
55434 object.

55435 **Issue 7**

55436 The *pthread\_attr\_getschedpolicy()* and *pthread\_attr\_setschedpolicy()* functions are marked only as  
55437 part of the Thread Execution Scheduling option as the Threads option is now part of the Base.

55438 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55439 results in undefined behavior.

55440 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0452 [314] is applied.

55441 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0263 [757] is applied.

55442 **NAME**

55443 pthread\_attr\_getscope, pthread\_attr\_setscope — get and set the contentionscope attribute  
55444 (**REALTIME THREADS**)

55445 **SYNOPSIS**

```
55446 TPS #include <pthread.h>
55447
55447 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
55448                          int *restrict contentionscope);
55449
55449 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

55450 **DESCRIPTION**

55451 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions, respectively, shall get and set  
55452 the *contentionscope* attribute in the *attr* object.

55453 The *contentionscope* attribute may have the values `PTHREAD_SCOPE_SYSTEM`, signifying  
55454 system scheduling contention scope, or `PTHREAD_SCOPE_PROCESS`, signifying process  
55455 scheduling contention scope. The symbols `PTHREAD_SCOPE_SYSTEM` and  
55456 `PTHREAD_SCOPE_PROCESS` are defined in the `<pthread.h>` header.

55457 The behavior is undefined if the value specified by the *attr* argument to *pthread\_attr\_getscope()* or  
55458 *pthread\_attr\_setscope()* does not refer to an initialized thread attributes object.

55459 **RETURN VALUE**

55460 If successful, the *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions shall return zero;  
55461 otherwise, an error number shall be returned to indicate the error.

55462 **ERRORS**

55463 The *pthread\_attr\_setscope()* function shall fail if:

55464 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

55465 The *pthread\_attr\_setscope()* function may fail if:

55466 [EINVAL] The value of *contentionscope* is not valid.

55467 These functions shall not return an error code of [EINTR].

55468 **EXAMPLES**

55469 None.

55470 **APPLICATION USAGE**

55471 After these attributes have been set, a thread can be created with the specified attributes using  
55472 *pthread\_create()*. Using these routines does not affect the current running thread.

55473 See [Section 2.9.4](#) (on page 540) for further details on thread scheduling attributes and their  
55474 default settings.

55475 **RATIONALE**

55476 If an implementation detects that the value specified by the *attr* argument to  
55477 *pthread\_attr\_getscope()* or *pthread\_attr\_setscope()* does not refer to an initialized thread attributes  
55478 object, it is recommended that the function should fail and report an [EINVAL] error.

55479 **FUTURE DIRECTIONS**

55480 None.

55481 **SEE ALSO**

55482 *pthread\_attr\_destroy()*, *pthread\_attr\_getinheritsched()*, *pthread\_attr\_getschedpolicy()*,  
55483 *pthread\_attr\_getschedparam()*, *pthread\_create()*

55484 XBD <pthread.h>, <sched.h>

55485 **CHANGE HISTORY**

55486 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55487 Marked as part of the Realtime Threads Feature Group.

55488 **Issue 6**

55489 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions are marked as part of the  
55490 Threads and Thread Execution Scheduling options.

55491 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
55492 implementation does not support the Thread Execution Scheduling option.

55493 The **restrict** keyword is added to the *pthread\_attr\_getscope()* prototype for alignment with the  
55494 ISO/IEC 9899:1999 standard.

55495 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/81 is applied, adding a reference to  
55496 [Section 2.9.4](#) (on page 540) in the APPLICATION USAGE section.

55497 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/82 is applied, updating the ERRORS  
55498 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
55499 object.

55500 **Issue 7**

55501 The *pthread\_attr\_getscope()* and *pthread\_attr\_setscope()* functions are marked only as part of the  
55502 Thread Execution Scheduling option as the Threads option is now part of the Base.

55503 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55504 results in undefined behavior.

55505 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0453 [314] is applied.

55506 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0264 [757] is applied.

55507 **NAME**

55508 pthread\_attr\_getstack, pthread\_attr\_setstack — get and set stack attributes

55509 **SYNOPSIS**

```
55510 TSA TSS #include <pthread.h>
55511 int pthread_attr_getstack(const pthread_attr_t *restrict attr,
55512 void **restrict stackaddr, size_t *restrict stacksize);
55513 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
55514 size_t stacksize);
```

55515 **DESCRIPTION**55516 The *pthread\_attr\_getstack()* and *pthread\_attr\_setstack()* functions, respectively, shall get and set the  
55517 thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.55518 The stack attributes specify the area of storage to be used for the created thread's stack. The base  
55519 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be  
55520 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD\_STACK\_MIN}. The  
55521 *pthread\_attr\_setstack()* function may fail with [EINVAL] if *stackaddr* does not meet  
55522 implementation-defined alignment requirements. All pages within the stack described by  
55523 *stackaddr* and *stacksize* shall be both readable and writable by the thread.55524 If the *pthread\_attr\_getstack()* function is called before the *stackaddr* attribute has been set, the  
55525 behavior is unspecified.55526 The behavior is undefined if the value specified by the *attr* argument to *pthread\_attr\_getstack()* or  
55527 *pthread\_attr\_setstack()* does not refer to an initialized thread attributes object.55528 **RETURN VALUE**55529 Upon successful completion, these functions shall return a value of 0; otherwise, an error  
55530 number shall be returned to indicate the error.55531 The *pthread\_attr\_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*  
55532 if successful.55533 **ERRORS**55534 The *pthread\_attr\_setstack()* function shall fail if:55535 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds an  
55536 implementation-defined limit.55537 The *pthread\_attr\_setstack()* function may fail if:55538 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or  
55539 ((**char** \*)*stackaddr* + *stacksize*) lacks proper alignment.55540 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable  
55541 and writable by the thread.

55542 These functions shall not return an error code of [EINTR].

55543 **EXAMPLES**

55544 None.

55545 **APPLICATION USAGE**55546 These functions are appropriate for use by applications in an environment where the stack for a  
55547 thread must be placed in some particular region of memory.55548 While it might seem that an application could detect stack overflow by providing a protected  
55549 page outside the specified stack region, this cannot be done portably. Implementations are free  
55550 to place the thread's initial stack pointer anywhere within the specified region to accommodate  
55551 the machine's stack pointer behavior and allocation requirements. Furthermore, on some  
55552 architectures, such as the IA-64, "overflow" might mean that two separate stack pointers  
55553 allocated within the region will overlap somewhere in the middle of the region.55554 After a successful call to *pthread\_attr\_setstack()*, the storage area specified by the *stackaddr*  
55555 parameter is under the control of the implementation, as described in [Section 2.9.8](#) (on page 548).55556 The specification of the *stackaddr* attribute presents several ambiguities that make portable use of  
55557 these functions impossible. For example, the standard allows implementations to impose  
55558 arbitrary alignment requirements on *stackaddr*. Applications cannot assume that a buffer  
55559 obtained from *malloc()* is suitably aligned. Note that although the *stacksize* value passed to  
55560 *pthread\_attr\_setstack()* must satisfy alignment requirements, the same is not true for  
55561 *pthread\_attr\_setstacksize()* where the implementation must increase the specified size if necessary  
55562 to achieve the proper alignment.55563 **RATIONALE**55564 If an implementation detects that the value specified by the *attr* argument to  
55565 *pthread\_attr\_getstack()* or *pthread\_attr\_setstack()* does not refer to an initialized thread attributes  
55566 object, it is recommended that the function should fail and report an [EINVAL] error.55567 **FUTURE DIRECTIONS**

55568 None.

55569 **SEE ALSO**55570 [\*pthread\\_attr\\_destroy\(\)\*](#), [\*pthread\\_attr\\_getdetachstate\(\)\*](#), [\*pthread\\_attr\\_getstacksize\(\)\*](#), [\*pthread\\_create\(\)\*](#)55571 XBD [`<limits.h>`](#), [`<pthread.h>`](#)55572 **CHANGE HISTORY**55573 First released in Issue 6. Developed as part of the XSI option and brought into the BASE by IEEE  
55574 PASC Interpretation 1003.1 #101.55575 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/83 is applied, updating the  
55576 APPLICATION USAGE section to refer to [Section 2.9.8](#) (on page 548).55577 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC/D6/84 is applied, updating the ERRORS  
55578 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
55579 object.55580 **Issue 7**55581 SD5-XSH-ERN-66 is applied, correcting the use of *attr* in the [EINVAL] error condition.55582 Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is  
55583 called before the *stackaddr* attribute is set.

55584 SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

55585 The description of the *stackaddr* attribute is updated in the DESCRIPTION and APPLICATION  
55586 USAGE sections.

55587  
55588

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.



55589 **NAME**

55590 pthread\_attr\_getstacksize, pthread\_attr\_setstacksize — get and set the stacksize attribute

55591 **SYNOPSIS**

```
55592 TSS #include <pthread.h>
55593 int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
55594 size_t *restrict stacksize);
55595 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

55596 **DESCRIPTION**55597 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions, respectively, shall get and  
55598 set the thread creation *stacksize* attribute in the *attr* object.55599 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created  
55600 threads stack.55601 The behavior is undefined if the value specified by the *attr* argument to  
55602 *pthread\_attr\_getstacksize()* or *pthread\_attr\_setstacksize()* does not refer to an initialized thread  
55603 attributes object.55604 **RETURN VALUE**55605 Upon successful completion, *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* shall  
55606 return a value of 0; otherwise, an error number shall be returned to indicate the error.55607 The *pthread\_attr\_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if  
55608 successful.55609 **ERRORS**55610 The *pthread\_attr\_setstacksize()* function shall fail if:55611 [EINVAL] The value of *stacksize* is less than {PTHREAD\_STACK\_MIN} or exceeds a  
55612 system-imposed limit.

55613 These functions shall not return an error code of [EINTR].

55614 **EXAMPLES**

55615 None.

55616 **APPLICATION USAGE**

55617 None.

55618 **RATIONALE**55619 If an implementation detects that the value specified by the *attr* argument to  
55620 *pthread\_attr\_getstacksize()* or *pthread\_attr\_setstacksize()* does not refer to an initialized thread  
55621 attributes object, it is recommended that the function should fail and report an [EINVAL] error.55622 **FUTURE DIRECTIONS**

55623 None.

55624 **SEE ALSO**55625 *pthread\_attr\_destroy()*, *pthread\_attr\_getdetachstate()*, *pthread\_create()*

55626 XBD &lt;limits.h&gt;, &lt;pthread.h&gt;

55627 **CHANGE HISTORY**

55628 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55629 **Issue 6**

55630 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions are marked as part of the  
55631 Threads and Thread Stack Size Attribute options.

55632 The **restrict** keyword is added to the *pthread\_attr\_getstacksize()* prototype for alignment with the  
55633 ISO/IEC 9899:1999 standard.

55634 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code  
55635 in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from “Thread Stack  
55636 Address Attribute” option to “Thread Stack Size Attribute” option.

55637 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/87 is applied, updating the ERRORS  
55638 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute  
55639 object.

55640 **Issue 7**

55641 The *pthread\_attr\_getstacksize()* and *pthread\_attr\_setstacksize()* functions are marked only as part of  
55642 the Thread Stack Size Attribute option as the Threads option is now part of the Base.

55643 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition  
55644 results in undefined behavior.

55645 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0265 [757] is applied.

55646 **NAME**

55647 pthread\_attr\_init — initialize the thread attributes object

55648 **SYNOPSIS**

55649 #include &lt;pthread.h&gt;

55650 int pthread\_attr\_init(pthread\_attr\_t \*attr);

55651 **DESCRIPTION**55652 Refer to *pthread\_attr\_destroy()*.

55653 **NAME**

55654 pthread\_attr\_setdetachstate — set the detachstate attribute

55655 **SYNOPSIS**

55656 #include <pthread.h>

55657 int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate);

55658 **DESCRIPTION**

55659 Refer to [pthread\\_attr\\_getdetachstate\(\)](#).

55660 **NAME**

55661 pthread\_attr\_setguardsize — set the thread guardsize attribute

55662 **SYNOPSIS**

55663 #include &lt;pthread.h&gt;

55664 int pthread\_attr\_setguardsize(pthread\_attr\_t \*attr,  
55665 size\_t guardsize);55666 **DESCRIPTION**55667 Refer to *pthread\_attr\_getguardsize()*.

55668 **NAME**

55669 pthread\_attr\_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)

55670 **SYNOPSIS**

```
55671 TPS #include <pthread.h>
55672 int pthread_attr_setinheritsched(pthread_attr_t *attr,
55673 int inheritsched);
```

55674 **DESCRIPTION**

55675 Refer to *pthread\_attr\_getinheritsched()*.

55676 **NAME**

55677 pthread\_attr\_setschedparam — set the schedparam attribute

55678 **SYNOPSIS**

55679 #include <pthread.h>

55680 int pthread\_attr\_setschedparam(pthread\_attr\_t \*restrict attr,  
55681 const struct sched\_param \*restrict param);

55682 **DESCRIPTION**

55683 Refer to [pthread\\_attr\\_getschedparam\(\)](#).

55684 **NAME**

55685 pthread\_attr\_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)

55686 **SYNOPSIS**

```
55687 TPS #include <pthread.h>  
55688 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

55689 **DESCRIPTION**

55690 Refer to [pthread\\_attr\\_getschedpolicy\(\)](#).



55691 **NAME**

55692 pthread\_attr\_setscope — set the contentionscope attribute (**REALTIME THREADS**)

55693 **SYNOPSIS**

```
55694 TPS #include <pthread.h>  
55695 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

55696 **DESCRIPTION**

55697 Refer to *pthread\_attr\_getscope()*.

55698 **NAME**

55699 pthread\_attr\_setstack — set the stack attribute

55700 **SYNOPSIS**

```
55701 TSA TSS #include <pthread.h>
55702 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
55703 size_t stacksize);
```

55704 **DESCRIPTION**

55705 Refer to [pthread\\_attr\\_getstack\(\)](#).

55706 **NAME**

55707 pthread\_attr\_setstacksize — set the stacksize attribute

55708 **SYNOPSIS**

```
55709 TSS #include <pthread.h>  
55710 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

55711 **DESCRIPTION**55712 Refer to *pthread\_attr\_getstacksize()*.

55713 **NAME**

55714 pthread\_barrier\_destroy, pthread\_barrier\_init — destroy and initialize a barrier object

55715 **SYNOPSIS**

55716 #include &lt;pthread.h&gt;

```
55717 int pthread_barrier_destroy(pthread_barrier_t *barrier);
55718 int pthread_barrier_init(pthread_barrier_t *restrict barrier,
55719     const pthread_barrierattr_t *restrict attr, unsigned count);
```

55720 **DESCRIPTION**

55721 The *pthread\_barrier\_destroy()* function shall destroy the barrier referenced by *barrier* and release  
55722 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until  
55723 the barrier is reinitialized by another call to *pthread\_barrier\_init()*. An implementation may use  
55724 this function to set *barrier* to an invalid value. The results are undefined if  
55725 *pthread\_barrier\_destroy()* is called when any thread is blocked on the barrier, or if this function is  
55726 called with an uninitialized barrier.

55727 The *pthread\_barrier\_init()* function shall allocate any resources required to use the barrier  
55728 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is  
55729 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of  
55730 a default barrier attributes object. The results are undefined if *pthread\_barrier\_init()* is called  
55731 when any thread is blocked on the barrier (that is, has not returned from the  
55732 *pthread\_barrier\_wait()* call). The results are undefined if a barrier is used without first being  
55733 initialized. The results are undefined if *pthread\_barrier\_init()* is called specifying an already  
55734 initialized barrier.

55735 The *count* argument specifies the number of threads that have to call *pthread\_barrier\_wait()*  
55736 before any of them successfully return from the call. The value specified by *count* needs to be  
55737 greater than zero.

55738 If the *pthread\_barrier\_init()* function fails, the barrier shall not be initialized and the contents of  
55739 *barrier* are undefined.

55740 See [Section 2.9.9](#) (on page 548) for further requirements.

55741 **RETURN VALUE**

55742 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
55743 be returned to indicate the error.

55744 **ERRORS**

55745 The *pthread\_barrier\_init()* function shall fail if:

55746 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

55747 [EINVAL] The value specified by *count* is equal to zero.

55748 [ENOMEM] Insufficient memory exists to initialize the barrier.

55749 These functions shall not return an error code of [EINTR].

55750 **EXAMPLES**

55751 None.

55752 **APPLICATION USAGE**

55753 None.

55754 **RATIONALE**

55755 If an implementation detects that the value specified by the *barrier* argument to  
55756 *pthread\_barrier\_destroy()* does not refer to an initialized barrier object, it is recommended that the  
55757 function should fail and report an [EINVAL] error.

55758 If an implementation detects that the value specified by the *attr* argument to  
55759 *pthread\_barrier\_init()* does not refer to an initialized barrier attributes object, it is recommended  
55760 that the function should fail and report an [EINVAL] error.

55761 If an implementation detects that the value specified by the *barrier* argument to  
55762 *pthread\_barrier\_destroy()* or *pthread\_barrier\_init()* refers to a barrier that is in use (for example, in  
55763 a *pthread\_barrier\_wait()* call) by another thread, or detects that the value specified by the *barrier*  
55764 argument to *pthread\_barrier\_init()* refers to an already initialized barrier object, it is  
55765 recommended that the function should fail and report an [EBUSY] error.

55766 **FUTURE DIRECTIONS**

55767 None.

55768 **SEE ALSO**55769 [\*pthread\\_barrier\\_wait\(\)\*](#)55770 XBD <[pthread.h](#)>55771 **CHANGE HISTORY**

55772 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55773 **Issue 7**

55774 The *pthread\_barrier\_destroy()* and *pthread\_barrier\_init()* functions are moved from the Barriers  
55775 option to the Base.

55776 The [EINVAL] error for an uninitialized barrier object and an uninitialized barrier attributes  
55777 object is removed; this condition results in undefined behavior.

55778 The [EBUSY] error for a barrier that is in use or an already initialized barrier object is removed;  
55779 this condition results in undefined behavior.

55780 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0266 [972] is applied.

55781 **NAME**

55782 pthread\_barrier\_wait — synchronize at a barrier

55783 **SYNOPSIS**

55784 #include &lt;pthread.h&gt;

55785 int pthread\_barrier\_wait(pthread\_barrier\_t \*barrier);

55786 **DESCRIPTION**

55787 The *pthread\_barrier\_wait()* function shall synchronize participating threads at the barrier  
55788 referenced by *barrier*. The calling thread shall block until the required number of threads have  
55789 called *pthread\_barrier\_wait()* specifying the barrier.

55790 When the required number of threads have called *pthread\_barrier\_wait()* specifying the barrier,  
55791 the constant PTHREAD\_BARRIER\_SERIAL\_THREAD shall be returned to one unspecified  
55792 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall  
55793 be reset to the state it had as a result of the most recent *pthread\_barrier\_init()* function that  
55794 referenced it.

55795 The constant PTHREAD\_BARRIER\_SERIAL\_THREAD is defined in <pthread.h> and its value  
55796 shall be distinct from any other value returned by *pthread\_barrier\_wait()*.

55797 The results are undefined if this function is called with an uninitialized barrier.

55798 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the  
55799 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the  
55800 required number of threads have not arrived at the barrier during the execution of the signal  
55801 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until  
55802 the thread in the signal handler returns from it, it is unspecified whether other threads may  
55803 proceed past the barrier once they have all reached it.

55804 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to  
55805 use the same processing resources from eventually making forward progress in its execution.  
55806 Eligibility for processing resources shall be determined by the scheduling policy.

55807 **RETURN VALUE**

55808 Upon successful completion, the *pthread\_barrier\_wait()* function shall return  
55809 PTHREAD\_BARRIER\_SERIAL\_THREAD for a single (arbitrary) thread synchronized at the  
55810 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to  
55811 indicate the error.

55812 **ERRORS**

55813 This function shall not return an error code of [EINTR].

55814 **EXAMPLES**

55815 None.

55816 **APPLICATION USAGE**

55817 Applications using this function may be subject to priority inversion, as discussed in XBD  
55818 [Section 3.275](#) (on page 72).

55819 **RATIONALE**

55820 If an implementation detects that the value specified by the *barrier* argument to  
55821 *pthread\_barrier\_wait()* does not refer to an initialized barrier object, it is recommended that the  
55822 function should fail and report an [EINVAL] error.

55823 **FUTURE DIRECTIONS**

55824 None.

55825 **SEE ALSO**55826 [pthread\\_barrier\\_destroy\(\)](#)55827 XBD [Section 3.275](#) (on page 72), [Section 4.15.2](#) (on page 104), [<pthread.h>](#)55828 **CHANGE HISTORY**

55829 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55830 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.55831 **Issue 7**55832 The [pthread\\_barrier\\_wait\(\)](#) function is moved from the Barriers option to the Base.55833 The [EINVAL] error for an uninitialized barrier object is removed; this condition results in  
55834 undefined behavior.

55835 **NAME**

55836 pthread\_barrierattr\_destroy, pthread\_barrierattr\_init — destroy and initialize the barrier  
55837 attributes object

55838 **SYNOPSIS**

```
55839 #include <pthread.h>
55840 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);
55841 int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

55842 **DESCRIPTION**

55843 The *pthread\_barrierattr\_destroy()* function shall destroy a barrier attributes object. A destroyed  
55844 *attr* attributes object can be reinitialized using *pthread\_barrierattr\_init()*; the results of otherwise  
55845 referencing the object after it has been destroyed are undefined. An implementation may cause  
55846 *pthread\_barrierattr\_destroy()* to set the object referenced by *attr* to an invalid value.

55847 The *pthread\_barrierattr\_init()* function shall initialize a barrier attributes object *attr* with the  
55848 default value for all of the attributes defined by the implementation.

55849 If *pthread\_barrierattr\_init()* is called specifying an already initialized *attr* attributes object, the  
55850 results are undefined.

55851 After a barrier attributes object has been used to initialize one or more barriers, any function  
55852 affecting the attributes object (including destruction) shall not affect any previously initialized  
55853 barrier.

55854 The behavior is undefined if the value specified by the *attr* argument to  
55855 *pthread\_barrierattr\_destroy()* does not refer to an initialized barrier attributes object.

55856 **RETURN VALUE**

55857 If successful, the *pthread\_barrierattr\_destroy()* and *pthread\_barrierattr\_init()* functions shall return  
55858 zero; otherwise, an error number shall be returned to indicate the error.

55859 **ERRORS**

55860 The *pthread\_barrierattr\_init()* function shall fail if:

55861 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

55862 These functions shall not return an error code of [EINTR].

55863 **EXAMPLES**

55864 None.

55865 **APPLICATION USAGE**

55866 None.

55867 **RATIONALE**

55868 If an implementation detects that the value specified by the *attr* argument to  
55869 *pthread\_barrierattr\_destroy()* does not refer to an initialized barrier attributes object, it is  
55870 recommended that the function should fail and report an [EINVAL] error.

55871 **FUTURE DIRECTIONS**

55872 None.

55873 **SEE ALSO**

55874 [pthread\\_barrierattr\\_getshared\(\)](#)

55875 XBD [<pthread.h>](#)



55876 **CHANGE HISTORY**

55877 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55878 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

55879 **Issue 7**

55880 The `pthread_barrierattr_destroy()` and `pthread_barrierattr_init()` functions are moved from the Barriers option to the Base.

55882 The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition results in undefined behavior.

55883

55884 **NAME**

55885 pthread\_barrierattr\_getpshared, pthread\_barrierattr\_setpshared — get and set the process-  
55886 shared attribute of the barrier attributes object

55887 **SYNOPSIS**

```
55888 TSH #include <pthread.h>
55889
55889 int pthread_barrierattr_getpshared(const pthread_barrierattr_t
55890     *restrict attr, int *restrict pshared);
55891 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
55892     int pshared);
```

55893 **DESCRIPTION**

55894 The *pthread\_barrierattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
55895 from the attributes object referenced by *attr*. The *pthread\_barrierattr\_setpshared()* function shall  
55896 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

55897 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a barrier to be  
55898 operated upon by any thread that has access to the memory where the barrier is allocated. See  
55899 [Section 2.9.9](#) (on page 548) for further requirements. The default value of the attribute shall be  
55900 `PTHREAD_PROCESS_PRIVATE`. Both constants `PTHREAD_PROCESS_SHARED` and  
55901 `PTHREAD_PROCESS_PRIVATE` are defined in **<pthread.h>**.

55902 Additional attributes, their default values, and the names of the associated functions to get and  
55903 set those attribute values are implementation-defined.

55904 The behavior is undefined if the value specified by the *attr* argument to  
55905 *pthread\_barrierattr\_getpshared()* or *pthread\_barrierattr\_setpshared()* does not refer to an initialized  
55906 barrier attributes object.

55907 **RETURN VALUE**

55908 If successful, the *pthread\_barrierattr\_getpshared()* function shall return zero and store the value of  
55909 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
55910 an error number shall be returned to indicate the error.

55911 If successful, the *pthread\_barrierattr\_setpshared()* function shall return zero; otherwise, an error  
55912 number shall be returned to indicate the error.

55913 **ERRORS**

55914 The *pthread\_barrierattr\_setpshared()* function may fail if:

55915 [EINVAL] The new value specified for the *process-shared* attribute is not one of the legal  
55916 values `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

55917 These functions shall not return an error code of [EINTR].

55918 **EXAMPLES**

55919 None.

55920 **APPLICATION USAGE**

55921 The *pthread\_barrierattr\_getpshared()* and *pthread\_barrierattr\_setpshared()* functions are part of the  
55922 Thread Process-Shared Synchronization option and need not be provided on all  
55923 implementations.

55924 **RATIONALE**

55925 If an implementation detects that the value specified by the *attr* argument to  
55926 *pthread\_barrierattr\_getpshared()* or *pthread\_barrierattr\_setpshared()* does not refer to an initialized  
55927 barrier attributes object, it is recommended that the function should fail and report an [EINVAL]

55928 error.

55929 **FUTURE DIRECTIONS**

55930 None.

55931 **SEE ALSO**

55932 *pthread\_barrier\_destroy()*, *pthread\_barrierattr\_destroy()*

55933 XBD <pthread.h>

55934 **CHANGE HISTORY**

55935 First released in Issue 6. Derived from IEEE Std 1003.1j-2000

55936 **Issue 7**

55937 The *pthread\_barrierattr\_getpshared()* and *pthread\_barrierattr\_setpshared()* functions are marked  
55938 only as part of the Thread Process-Shared Synchronization option as the Threads option is now  
55939 part of the Base.

55940 The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition  
55941 results in undefined behavior.

55942 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0266 [972] and XSH/TC2-2008/0267  
55943 [757] are applied.

55944 **NAME**

55945 pthread\_barrierattr\_init — initialize the barrier attributes object

55946 **SYNOPSIS**

55947 #include <pthread.h>

55948 int pthread\_barrierattr\_init(pthread\_barrierattr\_t \*attr);

55949 **DESCRIPTION**

55950 Refer to *pthread\_barrierattr\_destroy()*.

55951 **NAME**

55952 pthread\_barrierattr\_setpshared — set the process-shared attribute of the barrier attributes object

55953 **SYNOPSIS**

```
55954 TSH #include <pthread.h>
55955      int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
55956      int pshared);
```

55957 **DESCRIPTION**

55958 Refer to [pthread\\_barrierattr\\_getpshared\(\)](#).

55959 **NAME**

55960 pthread\_cancel — cancel execution of a thread

55961 **SYNOPSIS**

55962 #include &lt;pthread.h&gt;

55963 int pthread\_cancel(pthread\_t thread);

55964 **DESCRIPTION**

55965 The *pthread\_cancel()* function shall request that *thread* be canceled. The target thread's  
55966 cancelability state and type determines when the cancellation takes effect. When the cancellation  
55967 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last  
55968 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called  
55969 for *thread*. When the last destructor function returns, *thread* shall be terminated. It shall not be an  
55970 error to request cancellation of a zombie thread.

55971 The cancellation processing in the target thread shall run asynchronously with respect to the  
55972 calling thread returning from *pthread\_cancel()*.

55973 If *thread* refers to a thread that was created using *thr\_create()*, the behavior is undefined.

55974 **RETURN VALUE**

55975 If successful, the *pthread\_cancel()* function shall return zero; otherwise, an error number shall be  
55976 returned to indicate the error.

55977 **ERRORS**

55978 The *pthread\_cancel()* function shall not return an error code of [EINTR].

55979 **EXAMPLES**

55980 None.

55981 **APPLICATION USAGE**

55982 None.

55983 **RATIONALE**

55984 Two alternative functions were considered for sending the cancellation notification to a thread.  
55985 One would be to define a new SIGCANCEL signal that had the cancellation semantics when  
55986 delivered; the other was to define the new *pthread\_cancel()* function, which would trigger the  
55987 cancellation semantics.

55988 The advantage of a new signal was that so much of the delivery criteria were identical to that  
55989 used when trying to deliver a signal that making cancellation notification a signal was seen as  
55990 consistent. Indeed, many implementations implement cancellation using a special signal. On the  
55991 other hand, there would be no signal functions that could be used with this signal except  
55992 *pthread\_kill()*, and the behavior of the delivered cancellation signal would be unlike any  
55993 previously existing defined signal.

55994 The benefits of a special function include the recognition that this signal would be defined  
55995 because of the similar delivery criteria and that this is the only common behavior between a  
55996 cancellation request and a signal. In addition, the cancellation delivery mechanism does not  
55997 have to be implemented as a signal. There are also strong, if not stronger, parallels with  
55998 language exception mechanisms than with signals that are potentially obscured if the delivery  
55999 mechanism is visibly closer to signals.

56000 In the end, it was considered that as there were so many exceptions to the use of the new signal  
56001 with existing signals functions it would be misleading. A special function has resolved this  
56002 problem. This function was carefully defined so that an implementation wishing to provide the  
56003 cancellation functions on top of signals could do so. The special function also means that  
56004 implementations are not obliged to implement cancellation with signals.

56005 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
56006 that the function should fail and report an [ESRCH] error.

56007 Historical implementations varied on the result of a *pthread\_cancel()* with a thread ID indicating  
56008 a zombie thread. Some indicated success with nothing further to do because the thread had  
56009 already terminated, while others gave an error of [ESRCH]. Since the definition of thread  
56010 lifetime in this standard covers zombie threads, the [ESRCH] error as described is inappropriate  
56011 in this case and implementations that give this error do not conform.

56012 Use of *pthread\_cancel()* to cancel a thread that was created using *thrd\_create()* is undefined  
56013 because *thrd\_join()* has no way to indicate a thread was cancelled. The standard developers  
56014 considered adding a `thrd_canceled` enumeration constant that *thrd\_join()* would return in  
56015 this case. However, this return would be unexpected in code that is written to conform to the  
56016 ISO C standard, and it would also not solve the problem that threads which use only ISO C  
56017 `<threads.h>` interfaces (such as ones created by third party libraries written to conform to the  
56018 ISO C standard) have no way to handle being cancelled, as the ISO C standard does not provide  
56019 cancellation cleanup handlers.

#### 56020 FUTURE DIRECTIONS

56021 None.

#### 56022 SEE ALSO

56023 [\*pthread\\_exit\(\)\*](#), [\*pthread\\_cond\\_clockwait\(\)\*](#), [\*pthread\\_join\(\)\*](#), [\*pthread\\_setcancelstate\(\)\*](#)

56024 XBD `<pthread.h>`

#### 56025 CHANGE HISTORY

56026 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 56027 Issue 6

56028 The *pthread\_cancel()* function is marked as part of the Threads option.

#### 56029 Issue 7

56030 The *pthread\_cancel()* function is moved from the Threads option to the Base.

56031 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

#### 56032 Issue 8

56033 Austin Group Defect 792 is applied, adding a requirement that passing the thread ID of a zombie  
56034 thread to *pthread\_cancel()* is not treated as an error.

56035 Austin Group Defect 1302 is applied, updating the page to account for the addition of  
56036 `<threads.h>` interfaces.

56037 **NAME**

56038 pthread\_cleanup\_pop, pthread\_cleanup\_push — establish cancellation handlers

56039 **SYNOPSIS**

56040 #include &lt;pthread.h&gt;

56041 void pthread\_cleanup\_pop(int *execute*);56042 void pthread\_cleanup\_push(void (\**routine*)(void\*), void \**arg*);56043 **DESCRIPTION**56044 The *pthread\_cleanup\_pop()* function shall remove the routine at the top of the calling thread's  
56045 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).56046 The *pthread\_cleanup\_push()* function shall push the specified cancellation cleanup handler *routine*  
56047 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be  
56048 popped from the cancellation cleanup stack and invoked with the argument *arg* when:

- 56049
- The thread exits (that is, calls *pthread\_exit()*).
  - The thread acts upon a cancellation request.
  - The thread calls *pthread\_cleanup\_pop()* with a non-zero *execute* argument.

56052 It is unspecified whether *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* are macros or  
56053 functions. If a macro definition is suppressed in order to access an actual function, or a program  
56054 defines an external identifier with any of these names, the behavior is undefined. The  
56055 application shall ensure that they appear as statements, and in pairs within the same lexical  
56056 scope (that is, the *pthread\_cleanup\_push()* macro may be thought to expand to a token list whose  
56057 first token is '{' with *pthread\_cleanup\_pop()* expanding to a token list whose last token is the  
56058 corresponding '}').56059 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to  
56060 *pthread\_cleanup\_push()* or *pthread\_cleanup\_pop()* made without the matching call since the jump  
56061 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancellation  
56062 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation  
56063 cleanup handler.56064 Invoking a cancellation cleanup handler may terminate the execution of any code block being  
56065 executed by the thread whose execution began after the corresponding invocation of  
56066 *pthread\_cleanup\_push()*.56067 The effect of the use of **return**, **break**, **continue**, and **goto** to prematurely leave a code block  
56068 described by a pair of *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions calls is  
56069 undefined.56070 **RETURN VALUE**56071 The *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()* functions shall not return a value.56072 **ERRORS**

56073 No errors are defined.

56074 These functions shall not return an error code of [EINTR].



56075 **EXAMPLES**

56076 The following is an example using thread primitives to implement a cancelable, writers-priority  
56077 read-write lock:

```

56078 typedef struct {
56079     pthread_mutex_t lock;
56080     pthread_cond_t rcond,
56081     wcond;
56082     int lock_count; /* < 0 .. Held by writer. */
56083                   /* > 0 .. Held by lock_count readers. */
56084                   /* = 0 .. Held by nobody. */
56085     int waiting_writers; /* Count of waiting writers. */
56086 } rwlock;

56087 void
56088 waiting_reader_cleanup(void *arg)
56089 {
56090     rwlock *l;
56091
56092     l = (rwlock *) arg;
56093     pthread_mutex_unlock(&l->lock);
56094 }

56094 void
56095 lock_for_read(rwlock *l)
56096 {
56097     pthread_mutex_lock(&l->lock);
56098     pthread_cleanup_push(waiting_reader_cleanup, l);
56099     while ((l->lock_count < 0) || (l->waiting_writers != 0))
56100         pthread_cond_wait(&l->rcond, &l->lock);
56101     l->lock_count++;
56102     /*
56103      * Note the pthread_cleanup_pop executes
56104      * waiting_reader_cleanup.
56105      */
56106     pthread_cleanup_pop(1);
56107 }

56108 void
56109 release_read_lock(rwlock *l)
56110 {
56111     pthread_mutex_lock(&l->lock);
56112     if (--l->lock_count == 0)
56113         pthread_cond_signal(&l->wcond);
56114     pthread_mutex_unlock(&l->lock);
56115 }

56116 void
56117 waiting_writer_cleanup(void *arg)
56118 {
56119     rwlock *l;
56120
56121     l = (rwlock *) arg;
56122     if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
56123         /*

```

```
56123         * This only happens if we have been canceled. If the
56124         * lock is not held by a writer, there may be readers who
56125         * were blocked because waiting_writers was positive; they
56126         * can now be unblocked.
56127         */
56128         pthread_cond_broadcast (&l->rcond);
56129     }
56130     pthread_mutex_unlock (&l->lock);
56131 }

56132 void
56133 lock_for_write (rwlock *l)
56134 {
56135     pthread_mutex_lock (&l->lock);
56136     l->waiting_writers++;
56137     pthread_cleanup_push (waiting_writer_cleanup, l);
56138     while (l->lock_count != 0)
56139         pthread_cond_wait (&l->wcond, &l->lock);
56140     l->lock_count = -1;
56141     /*
56142     * Note the pthread_cleanup_pop executes
56143     * waiting_writer_cleanup.
56144     */
56145     pthread_cleanup_pop (1);
56146 }

56147 void
56148 release_write_lock (rwlock *l)
56149 {
56150     pthread_mutex_lock (&l->lock);
56151     l->lock_count = 0;
56152     if (l->waiting_writers == 0)
56153         pthread_cond_broadcast (&l->rcond);
56154     else
56155         pthread_cond_signal (&l->wcond);
56156     pthread_mutex_unlock (&l->lock);
56157 }

56158 /*
56159 * This function is called to initialize the read/write lock.
56160 */
56161 void
56162 initialize_rwlock (rwlock *l)
56163 {
56164     pthread_mutex_init (&l->lock, pthread_mutexattr_default);
56165     pthread_cond_init (&l->wcond, pthread_condattr_default);
56166     pthread_cond_init (&l->rcond, pthread_condattr_default);
56167     l->lock_count = 0;
56168     l->waiting_writers = 0;
56169 }

56170 reader_thread ()
56171 {
56172     lock_for_read (&lock);
```

```

56173     pthread_cleanup_push(release_read_lock, &lock);
56174     /*
56175     * Thread has read lock.
56176     */
56177     pthread_cleanup_pop(1);
56178 }

56179 writer_thread()
56180 {
56181     lock_for_write(&lock);
56182     pthread_cleanup_push(release_write_lock, &lock);
56183     /*
56184     * Thread has write lock.
56185     */
56186     pthread_cleanup_pop(1);
56187 }

```

### 56188 APPLICATION USAGE

56189 The two routines that push and pop cancellation cleanup handlers, *pthread\_cleanup\_push()* and  
56190 *pthread\_cleanup\_pop()*, can be thought of as left and right-parentheses. They always need to be  
56191 matched.

### 56192 RATIONALE

56193 The restriction that the two routines that push and pop cancellation cleanup handlers,  
56194 *pthread\_cleanup\_push()* and *pthread\_cleanup\_pop()*, have to appear in the same lexical scope  
56195 allows for efficient macro or compiler implementations and efficient storage management. A  
56196 sample implementation of these routines as macros might look like this:

```

56197 #define pthread_cleanup_push(rtn, arg) { \
56198     struct _pthread_handler_rec __cleanup_handler, **__head; \
56199     __cleanup_handler.rtn = rtn; \
56200     __cleanup_handler.arg = arg; \
56201     (void) pthread_getspecific(_pthread_handler_key, &__head); \
56202     __cleanup_handler.next = *__head; \
56203     *__head = &__cleanup_handler;
56204
56205 #define pthread_cleanup_pop(ex) \
56206     *__head = __cleanup_handler.next; \
56207     if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
56208 }

```

56208 A more ambitious implementation of these routines might do even better by allowing the  
56209 compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

56210 This volume of POSIX.1-2024 currently leaves unspecified the effect of calling *longjmp()* from a  
56211 signal handler executing in a POSIX System Interfaces function. If an implementation wants to  
56212 allow this and give the programmer reasonable behavior, the *longjmp()* function has to call all  
56213 cancellation cleanup handlers that have been pushed but not popped since the time *setjmp()* was  
56214 called.

56215 Consider a multi-threaded function called by a thread that uses signals. If a signal were  
56216 delivered to a signal handler during the operation of *qsort()* and that handler were to call  
56217 *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads  
56218 created by the *qsort()* function would not be canceled. Instead, they would continue to execute  
56219 and write into the argument array even though the array might have been popped off the stack.

56220 Note that the specified cleanup handling mechanism is especially tied to the C language and,  
56221 while the requirement for a uniform mechanism for expressing cleanup is language-  
56222 independent, the mechanism used in other languages may be quite different. In addition, this  
56223 mechanism is really only necessary due to the lack of a real exception mechanism in the C  
56224 language, which would be the ideal solution.

56225 There is no notion of a cancellation cleanup-safe function. If an application has no cancellation  
56226 points in its signal handlers, blocks any signal whose handler may have cancellation points  
56227 while calling async-unsafe functions, or disables cancellation while calling async-unsafe  
56228 functions, all functions may be safely called from cancellation cleanup routines.

#### 56229 FUTURE DIRECTIONS

56230 None.

#### 56231 SEE ALSO

56232 [\*pthread\\_cancel\(\)\*](#), [\*pthread\\_setcancelstate\(\)\*](#)

56233 XBD <[\*pthread.h\*](#)>

#### 56234 CHANGE HISTORY

56235 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 56236 Issue 6

56237 The [\*pthread\\_cleanup\\_pop\(\)\*](#) and [\*pthread\\_cleanup\\_push\(\)\*](#) functions are marked as part of the  
56238 Threads option.

56239 The APPLICATION USAGE section is added.

56240 The normative text is updated to avoid use of the term “must” for application requirements.

56241 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/88 is applied, updating the  
56242 DESCRIPTION to describe the consequences of prematurely leaving a code block defined by the  
56243 [\*pthread\\_cleanup\\_push\(\)\*](#) and [\*pthread\\_cleanup\\_pop\(\)\*](#) functions.

#### 56244 Issue 7

56245 The [\*pthread\\_cleanup\\_pop\(\)\*](#) and [\*pthread\\_cleanup\\_push\(\)\*](#) functions are moved from the Threads  
56246 option to the Base.

56247 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0454 [229] is applied.

56248 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0268 [624] is applied.

#### 56249 Issue 8

56250 Austin Group Defect 613 is applied, clarifying the relationship of automatic object lifetimes to  
56251 cancellation cleanup functions.

56252 **NAME**

56253 pthread\_cond\_broadcast, pthread\_cond\_signal — broadcast or signal a condition

56254 **SYNOPSIS**

56255 #include &lt;pthread.h&gt;

56256 int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

56257 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

56258 **DESCRIPTION**

56259 These functions shall unblock threads blocked on a condition variable.

56260 The *pthread\_cond\_broadcast()* function shall, as a single atomic operation, determine which threads, if any, are blocked on the specified condition variable *cond* and unblock all of these threads.

56263 The *pthread\_cond\_signal()* function shall, as a single atomic operation, determine which threads, if any, are blocked on the specified condition variable *cond* and unblock at least one of these threads.

56266 If more than one thread is blocked on a condition variable, the scheduling policy shall determine the order in which threads are unblocked. When each thread unblocked as a result of a *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* returns from its call to *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()*, the thread shall own the mutex with which it called *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()*. The thread(s) that are unblocked shall contend for the mutex according to the scheduling policy (if applicable), and as if each had called *pthread\_mutex\_lock()*.

56273 The *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* functions may be called by a thread whether or not it currently owns the mutex that threads calling *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* have associated with the condition variable during their waits; however, if predictable scheduling behavior is required, then that mutex shall be locked by the thread calling *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()*.

56278 The *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall have no effect if they determine that there are no threads blocked on *cond*.

56280 The behavior is undefined if the value specified by the *cond* argument to *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* does not refer to an initialized condition variable.

56282 **RETURN VALUE**

56283 If successful, the *pthread\_cond\_broadcast()* and *pthread\_cond\_signal()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

56285 **ERRORS**

56286 These functions shall not return an error code of [EINTR].

56287 **EXAMPLES**

56288 None.

56289 **APPLICATION USAGE**

56290 The *pthread\_cond\_broadcast()* function is used whenever the shared-variable state has been changed in a way that more than one thread can proceed with its task. Consider a single producer/multiple consumer problem, where the producer can insert multiple items on a list that is accessed one item at a time by the consumers. By calling the *pthread\_cond\_broadcast()* function, the producer would notify all consumers that might be waiting, and thereby the application would receive more throughput on a multi-processor. In addition, *pthread\_cond\_broadcast()* makes it easier to implement a read-write lock. The *pthread\_cond\_broadcast()* function is needed in order to wake up all waiting readers when a

56298 writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function  
56299 to notify all clients of an impending transaction commit.

56300 It is not safe to use the *pthread\_cond\_signal()* function in a signal handler that is invoked  
56301 asynchronously. Even if it were safe, there would still be a race between the test of the Boolean  
56302 *pthread\_cond\_wait()* that could not be efficiently eliminated.

56303 Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling  
56304 from code running in a signal handler.

#### 56305 RATIONALE

56306 If an implementation detects that the value specified by the *cond* argument to  
56307 *pthread\_cond\_broadcast()* or *pthread\_cond\_signal()* does not refer to an initialized condition  
56308 variable, it is recommended that the function should fail and report an [EINVAL] error.

#### 56309 Multiple Awakenings by Condition Signal

56310 On a multi-processor, it may be impossible for an implementation of *pthread\_cond\_signal()* to  
56311 avoid the unblocking of more than one thread blocked on a condition variable. For example,  
56312 consider the following partial implementation of *pthread\_cond\_wait()* and *pthread\_cond\_signal()*,  
56313 executed by two threads in the order given. One thread is trying to wait on the condition  
56314 variable, another is concurrently executing *pthread\_cond\_signal()*, while a third thread is already  
56315 waiting.

```
56316 pthread_cond_wait(mutex, cond):
56317     value = cond->value; /* 1 */
56318     pthread_mutex_unlock(mutex); /* 2 */
56319     pthread_mutex_lock(cond->mutex); /* 10 */
56320     if (value == cond->value) { /* 11 */
56321         me->next_cond = cond->waiter;
56322         cond->waiter = me;
56323         pthread_mutex_unlock(cond->mutex);
56324         unable_to_run(me);
56325     } else
56326         pthread_mutex_unlock(cond->mutex); /* 12 */
56327     pthread_mutex_lock(mutex); /* 13 */

56328 pthread_cond_signal(cond):
56329     pthread_mutex_lock(cond->mutex); /* 3 */
56330     cond->value++; /* 4 */
56331     if (cond->waiter) { /* 5 */
56332         sleeper = cond->waiter; /* 6 */
56333         cond->waiter = sleeper->next_cond; /* 7 */
56334         able_to_run(sleeper); /* 8 */
56335     }
56336     pthread_mutex_unlock(cond->mutex); /* 9 */
```

56337 The effect is that more than one thread can return from its call to *pthread\_cond\_clockwait()*,  
56338 *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* as a result of one call to *pthread\_cond\_signal()*.  
56339 This effect is called “spurious wakeup”. Note that the situation is self-correcting in that the  
56340 number of threads that are so awakened is finite; for example, the next thread to call  
56341 *pthread\_cond\_wait()* after the sequence of events above blocks.

56342 While this problem could be resolved, the loss of efficiency for a fringe condition that occurs  
56343 only rarely is unacceptable, especially given that one has to check the predicate associated with a  
56344 condition variable anyway. Correcting this problem would unnecessarily reduce the degree of

- 56345 concurrency in this basic building block for all higher-level synchronization operations.
- 56346 An added benefit of allowing spurious wakeups is that applications are forced to code a  
56347 predicate-testing-loop around the condition wait. This also makes the application tolerate  
56348 superfluous condition broadcasts or signals on the same condition variable that may be coded in  
56349 some other part of the application. The resulting applications are thus more robust. Therefore,  
56350 POSIX.1-2024 explicitly documents that spurious wakeups may occur.
- 56351 **FUTURE DIRECTIONS**
- 56352 None.
- 56353 **SEE ALSO**
- 56354 [\*pthread\\_cond\\_clockwait\(\)\*](#), [\*pthread\\_cond\\_destroy\(\)\*](#)
- 56355 XBD Section 4.15.2 (on page 104), [\*\*<pthread.h>\*\*](#)
- 56356 **CHANGE HISTORY**
- 56357 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 56358 **Issue 6**
- 56359 The [\*pthread\\_cond\\_broadcast\(\)\*](#) and [\*pthread\\_cond\\_signal\(\)\*](#) functions are marked as part of the  
56360 Threads option.
- 56361 The APPLICATION USAGE section is added.
- 56362 **Issue 7**
- 56363 The [\*pthread\\_cond\\_broadcast\(\)\*](#) and [\*pthread\\_cond\\_signal\(\)\*](#) functions are moved from the Threads  
56364 option to the Base.
- 56365 The [EINVAL] error for an uninitialized condition variable is removed; this condition results in  
56366 undefined behavior.
- 56367 **Issue 8**
- 56368 Austin Group Defect 609 is applied, adding atomicity requirements.
- 56369 Austin Group Defect 1216 is applied, adding [\*pthread\\_cond\\_clockwait\(\)\*](#).

56370 **NAME**

56371 pthread\_cond\_clockwait, pthread\_cond\_timedwait, pthread\_cond\_wait — wait on a condition

56372 **SYNOPSIS**

```
56373 #include <pthread.h>
56374 int pthread_cond_clockwait(pthread_cond_t *restrict cond,
56375 pthread_mutex_t *restrict mutex, clockid_t clock_id,
56376 const struct timespec *restrict abstime);
56377 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
56378 pthread_mutex_t *restrict mutex,
56379 const struct timespec *restrict abstime);
56380 int pthread_cond_wait(pthread_cond_t *restrict cond,
56381 pthread_mutex_t *restrict mutex);
```

56382 **DESCRIPTION**

56383 The *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, and *pthread\_cond\_wait()* functions shall  
 56384 block on a condition variable. The application shall ensure that these functions are called with  
 56385 *mutex* locked by the calling thread; otherwise, an error (for PTHREAD\_MUTEX\_ERRORCHECK  
 56386 and robust mutexes) or undefined behavior (for other mutexes) results.

56387 These functions atomically release *mutex* and cause the calling thread to block on the condition  
 56388 variable *cond*; atomically here means “atomically with respect to access by another thread to the  
 56389 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex  
 56390 after the about-to-block thread has released it, then a subsequent call to *pthread\_cond\_broadcast()*  
 56391 or *pthread\_cond\_signal()* in that thread shall behave as if it were issued after the about-to-block  
 56392 thread has blocked.

56393 Upon successful return, the mutex shall have been locked and shall be owned by the calling  
 56394 thread.

56395 If *mutex* is a robust mutex where an owner terminated while holding the lock and the state is  
 56396 recoverable, the mutex shall be acquired even though the function returns [EOWNERDEAD].

56397 When using condition variables there is always a Boolean predicate involving shared variables  
 56398 associated with each condition wait that is true if the thread should proceed. Spurious wakeups  
 56399 from the *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* functions  
 56400 may occur. Since the return from *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or  
 56401 *pthread\_cond\_wait()* does not imply anything about the value of this predicate, the predicate  
 56402 should be re-evaluated upon such return.

56403 When a thread waits on a condition variable, having specified a particular mutex to the  
 56404 *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* operation, a dynamic  
 56405 binding is formed between that mutex and condition variable that remains in effect as long as at  
 56406 least one thread is blocked on the condition variable. During this time, the effect of an attempt  
 56407 by any thread to wait on that condition variable using a different mutex is undefined. Once all  
 56408 waiting threads have been unblocked (as by the *pthread\_cond\_broadcast()* operation), the next  
 56409 wait operation on that condition variable shall form a new dynamic binding with the mutex  
 56410 specified by that wait operation. Even though the dynamic binding between condition variable  
 56411 and mutex may be removed or replaced between the time a thread is unblocked from a wait on  
 56412 the condition variable and the time that it returns to the caller or begins cancellation cleanup,  
 56413 the unblocked thread shall always re-acquire the mutex specified in the condition wait operation  
 56414 call from which it is returning.

56415 A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a  
 56416 thread is set to PTHREAD\_CANCEL\_DEFERRED, a side-effect of acting upon a cancellation  
 56417 request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first



56418 cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up  
 56419 to the point of returning from the call to *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or  
 56420 *pthread\_cond\_wait()*, but at that point notices the cancellation request and, instead of returning to  
 56421 the caller, starts the thread cancellation activities, which includes calling cancellation cleanup  
 56422 handlers.

56423 A thread that has been unblocked because it has been canceled while blocked in a call to  
 56424 *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* shall not consume any  
 56425 condition signal that may be directed concurrently at the condition variable if there are other  
 56426 threads blocked on the condition variable.

56427 The *pthread\_cond\_clockwait()* function shall be equivalent to *pthread\_cond\_wait()*, except that an  
 56428 error is returned if the absolute time specified by *abstime* as measured against the clock indicated  
 56429 by *clock\_id* passes (that is, the current time measured by that clock equals or exceeds *abstime*)  
 56430 before the condition *cond* is signaled or broadcasted, or if the absolute time specified by *abstime*  
 56431 has already been passed at the time of the call. Implementations shall support passing  
 56432 CLOCK\_REALTIME and CLOCK\_MONOTONIC to *pthread\_cond\_clockwait()* as the *clock\_id*  
 56433 argument. When such timeouts occur, *pthread\_cond\_clockwait()* shall nonetheless release and re-  
 56434 acquire the mutex referenced by *mutex*, and may consume a condition signal directed  
 56435 concurrently at the condition variable.

56436 The *pthread\_cond\_timedwait()* function shall be equivalent to *pthread\_cond\_clockwait()*, except that  
 56437 it lacks the *clock\_id* argument. The clock to measure *abstime* against shall instead come from the  
 56438 condition variable's clock attribute which can be set by *pthread\_condattr\_setclock()* prior to the  
 56439 condition variable's creation. If no clock attribute has been set, the default shall be  
 56440 CLOCK\_REALTIME.

56441 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal  
 56442 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it  
 56443 shall return zero due to spurious wakeup.

56444 The behavior is undefined if the value specified by the *cond* or *mutex* argument to these  
 56445 functions does not refer to an initialized condition variable or an initialized mutex object,  
 56446 respectively.

#### 56447 RETURN VALUE

56448 Except for [ETIMEDOUT], [ENOTRECOVERABLE], and [EOWNERDEAD], all these error  
 56449 checks shall act as if they were performed immediately at the beginning of processing for the  
 56450 function and shall cause an error return, in effect, prior to modifying the state of the mutex  
 56451 specified by *mutex* or the condition variable specified by *cond*.

56452 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall  
 56453 be returned to indicate the error.

#### 56454 ERRORS

56455 These functions shall fail if:

56456 [EAGAIN] The mutex is a robust mutex and the system resources available for robust  
 56457 mutexes owned would be exceeded.

56458 [ENOTRECOVERABLE]  
 56459 The state protected by the mutex is not recoverable.

56460 [EOWNERDEAD]  
 56461 The mutex is a robust mutex and the process containing the previous owning  
 56462 thread terminated while holding the mutex lock. The mutex lock shall be  
 56463 acquired by the calling thread and it is up to the new owner to make the state

56464 consistent.

56465 [EPERM] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK or the mutex is a  
56466 robust mutex, and the current thread does not own the mutex.

56467 The `pthread_cond_clockwait()` and `pthread_cond_timedwait()` functions shall fail if:

56468 [ETIMEDOUT] The time specified by *abstime* has passed.

56469 [EINVAL] The *abstime* argument specified a nanosecond value less than zero or greater  
56470 than or equal to 1000 million, or the *clock\_id* argument passed to  
56471 `pthread_cond_clockwait()` is invalid or not supported.

56472 These functions may fail if:

56473 [EOWNERDEAD]  
56474 The mutex is a robust mutex and the previous owning thread terminated  
56475 while holding the mutex lock. The mutex lock shall be acquired by the calling  
56476 thread and it is up to the new owner to make the state consistent.

56477 These functions shall not return an error code of [EINTR].

**EXAMPLES**

56478 None.  
56479

**APPLICATION USAGE**

56481 Applications that have assumed that non-zero return values are errors will need updating for  
56482 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting  
56483 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error  
56484 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If  
56485 an application is supposed to work with normal and robust mutexes, it should check all return  
56486 values for error conditions and if necessary take appropriate action.

**RATIONALE**

56487 If an implementation detects that the value specified by the *cond* argument to  
56488 `pthread_cond_clockwait()`, `pthread_cond_timedwait()`, or `pthread_cond_wait()` does not refer to an  
56489 initialized condition variable, or detects that the value specified by the *mutex* argument does not  
56490 refer to an initialized mutex object, it is recommended that the function should fail and report an  
56491 [EINVAL] error.  
56492

**Condition Wait Semantics**

56493 It is important to note that when `pthread_cond_clockwait()`, `pthread_cond_timedwait()`, and  
56494 `pthread_cond_wait()` return without error, the associated predicate may still be false. Similarly,  
56495 when `pthread_cond_clockwait()` or `pthread_cond_timedwait()` returns with the timeout error, the  
56496 associated predicate may be true due to an unavoidable race between the expiration of the  
56497 timeout and the predicate state change.  
56498

56499 The application needs to recheck the predicate on any return because it cannot be sure there is  
56500 another thread waiting on the thread to handle the signal, and if there is not then the signal is  
56501 lost. The burden is on the application to check the predicate.

56502 Some implementations, particularly on a multi-processor, may sometimes cause multiple  
56503 threads to wake up when the condition variable is signaled simultaneously on different  
56504 processors.

56505 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate  
56506 associated with the condition wait to determine whether it can safely proceed, should wait  
56507 again, or should declare a timeout. A return from the wait does not imply that the associated

56508 predicate is either true or false.

56509 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”  
56510 that checks the predicate.

### 56511 **Timed Wait Semantics**

56512 An absolute time measure was chosen for specifying the timeout parameter for two reasons.  
56513 First, a relative time measure can be easily implemented on top of a function that specifies  
56514 absolute time, but there is a race condition associated with specifying an absolute timeout on top  
56515 of a function that specifies relative timeouts. For example, assume that `clock_gettime()` returns  
56516 the current time and `cond_relative_timed_wait()` uses relative timeouts:

```
56517 clock_gettime(CLOCK_REALTIME, &now)
56518 reltime = sleep_til_this_absolute_time -now;
56519 cond_relative_timed_wait(c, m, &reltime);
```

56520 If the thread is preempted between the first statement and the last statement, the thread blocks  
56521 for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout  
56522 also need not be recomputed if it is used multiple times in a loop, such as that enclosing a  
56523 condition wait.

56524 For cases when the system clock is advanced discontinuously by an operator, it is expected that  
56525 implementations process any timed wait expiring at an intervening time as if that time had  
56526 actually occurred.

### 56527 **Choice of Clock**

56528 Care should be taken to decide which clock is most appropriate when waiting with a timeout.  
56529 The system clock `CLOCK_REALTIME`, as used by default with `pthread_cond_timedwait()`, may be  
56530 subject to jumps forwards and backwards in order to correct it against actual time.  
56531 `CLOCK_MONOTONIC` is guaranteed not to jump backwards and must also advance in real  
56532 time, so using it via `pthread_cond_clockwait()` or `pthread_condattr_setclock()` may be more  
56533 appropriate.

### 56534 **Cancellation and Condition Wait**

56535 A condition wait, whether timed or not, is a cancellation point. That is, the functions  
56536 `pthread_cond_clockwait()`, `pthread_cond_timedwait()`, and `pthread_cond_wait()` are points where a  
56537 pending (or concurrent) cancellation request is noticed. The reason for this is that an indefinite  
56538 wait is possible at these points—whatever event is being waited for, even if the program is  
56539 totally correct, might never occur; for example, some input data being awaited might never be  
56540 sent. By making condition wait a cancellation point, the thread can be canceled and perform its  
56541 cancellation cleanup handler even though it may be stuck in some indefinite wait.

56542 A side-effect of acting on a cancellation request while a thread is blocked on a condition variable  
56543 is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in  
56544 order to ensure that the cancellation cleanup handler is executed in the same state as the critical  
56545 code that lies both before and after the call to the condition wait function. This rule is also  
56546 required when interfacing to POSIX threads from languages, such as Ada or C++, which may  
56547 choose to map cancellation onto a language exception; this rule ensures that each exception  
56548 handler guarding a critical section can always safely depend upon the fact that the associated  
56549 mutex has already been locked regardless of exactly where within the critical section the  
56550 exception was raised. Without this rule, there would not be a uniform rule that exception  
56551 handlers could follow regarding the lock, and so coding would become very cumbersome.

56552 Therefore, since *some* statement has to be made regarding the state of the lock when a  
56553 cancellation is delivered during a wait, a definition has been chosen that makes application  
56554 coding most convenient and error free.

56555 When acting on a cancellation request while a thread is blocked on a condition variable, the  
56556 implementation is required to ensure that the thread does not consume any condition signals  
56557 directed at that condition variable if there are any other threads waiting on that condition  
56558 variable. This rule is specified in order to avoid deadlock conditions that could occur if these  
56559 two independent requests (one acting on a thread and the other acting on the condition variable)  
56560 were not processed independently.

#### 56561 **Performance of Mutexes and Condition Variables**

56562 Mutexes are expected to be locked only for a few instructions. This practice is almost  
56563 automatically enforced by the desire of programmers to avoid long serial regions of execution  
56564 (which would reduce total effective parallelism).

56565 When using mutexes and condition variables, one tries to ensure that the usual case is to lock the  
56566 mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a  
56567 relatively rare situation. For example, when implementing a read-write lock, code that acquires a  
56568 read-lock typically needs only to increment the count of readers (under mutual-exclusion) and  
56569 return. The calling thread would actually wait on the condition variable only when there is  
56570 already an active writer. So the efficiency of a synchronization operation is bounded by the cost  
56571 of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context  
56572 switch.

56573 This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be  
56574 at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable  
56575 is important. The cost of waiting on a condition variable should be little more than the minimal  
56576 cost for a context switch plus the time to unlock and lock the mutex.

#### 56577 **Features of Mutexes and Condition Variables**

56578 It had been suggested that the mutex acquisition and release be decoupled from condition wait.  
56579 This was rejected because it is the combined nature of the operation that, in fact, facilitates  
56580 realtime implementations. Those implementations can atomically move a high-priority thread  
56581 between the condition variable and the mutex in a manner that is transparent to the caller. This  
56582 can prevent extra context switches and provide more deterministic acquisition of a mutex when  
56583 the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the  
56584 scheduling discipline. Furthermore, the current condition wait operation matches existing  
56585 practice.

#### 56586 **Scheduling Behavior of Mutexes and Condition Variables**

56587 Synchronization primitives that attempt to interfere with scheduling policy by specifying an  
56588 ordering rule are considered undesirable. Threads waiting on mutexes and condition variables  
56589 are selected to proceed in an order dependent upon the scheduling policy rather than in some  
56590 fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which  
56591 thread(s) are awakened and allowed to proceed.

56592 **Timed Condition Wait**

56593 The `pthread_cond_clockwait()` and `pthread_cond_timedwait()` functions allow an application to give  
56594 up waiting for a particular condition after a given amount of time. An example follows:

```
56595 (void) pthread_mutex_lock(&t.mn);
56596     t.waiters++;
56597     clock_gettime(CLOCK_MONOTONIC, &ts);
56598     ts.tv_sec += 5;
56599     rc = 0;
56600     while (! mypredicate(&t) && rc == 0)
56601         rc = pthread_cond_clockwait(&t.cond, &t.mn,
56602             CLOCK_MONOTONIC, &ts);
56603     t.waiters--;
56604     if (rc == 0 || mypredicate(&t))
56605         setmystate(&t);
56606 (void) pthread_mutex_unlock(&t.mn);
```

56607 By making the timeout parameter absolute, it does not need to be recomputed each time the  
56608 program checks its blocking predicate. If the timeout was relative, it would have to be  
56609 recomputed before each call. This would be especially difficult since such code would need to  
56610 take into account the possibility of extra wakeups that result from extra broadcasts or signals on  
56611 the condition variable that occur before either the predicate is true or the timeout is due. Using  
56612 `CLOCK_MONOTONIC` rather than `CLOCK_REALTIME` means that the timeout is not  
56613 influenced by the system clock being changed.

56614 **FUTURE DIRECTIONS**

56615 None.

56616 **SEE ALSO**

56617 [pthread\\_cond\\_broadcast\(\)](#)

56618 XBD Section 4.15.2 (on page 104), [<pthread.h>](#)

56619 **CHANGE HISTORY**

56620 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

56621 **Issue 6**

56622 The `pthread_cond_timedwait()` and `pthread_cond_wait()` functions are marked as part of the  
56623 Threads option.

56624 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the  
56625 `pthread_cond_wait()` function.

56626 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics  
56627 for the Clock Selection option.

56628 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC  
56629 Interpretation 1003.1c #28.

56630 The **restrict** keyword is added to the `pthread_cond_timedwait()` and `pthread_cond_wait()`  
56631 prototypes for alignment with the ISO/IEC 9899:1999 standard.

56632 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/89 is applied, updating the  
56633 DESCRIPTION for consistency with the `pthread_cond_destroy()` function that states it is safe to  
56634 destroy an initialized condition variable upon which no threads are currently blocked.

56635 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/90 is applied, updating words in the  
56636 DESCRIPTION from “the cancelability enable state” to “the cancelability type”.

56637 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/91 is applied, updating the ERRORS  
56638 section to remove the error case related to *abstime* from the *pthread\_cond\_wait()* function, and to  
56639 make the error case related to *abstime* mandatory for *pthread\_cond\_timedwait()* for consistency  
56640 with other functions.

56641 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/92 is applied, adding a new paragraph to  
56642 the RATIONALE section stating that an application should check the predicate on any return  
56643 from this function.

#### 56644 Issue 7

56645 SD5-XSH-ERN-44 is applied, changing the definition of the “shall fail” case of the [EINVAL]  
56646 error.

56647 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

56648 The *pthread\_cond\_timedwait()* and *pthread\_cond\_wait()* functions are moved from the Threads  
56649 option to the Base.

56650 The [EINVAL] error for an uninitialized condition variable or uninitialized mutex object is  
56651 removed; this condition results in undefined behavior”

56652 The [EPERM] error is revised and moved to the “shall fail” list of error conditions for the  
56653 *pthread\_cond\_timedwait()* function.

56654 The DESCRIPTION is updated to clarify the behavior when *mutex* is a robust mutex.

56655 The ERRORS section is updated to include “shall fail” cases for  
56656 PTHREAD\_MUTEX\_ERRORCHECK mutexes.

56657 The DESCRIPTION is rewritten to clarify that undefined behavior occurs only for mutexes  
56658 where the [EPERM] error is not mandated.

56659 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0456 [91,286,437] and  
56660 XSH/TC1-2008/0457 [239] are applied.

56661 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0271 [749] is applied.

#### 56662 Issue 8

56663 Austin Group Defect 354 is applied, adding the [EAGAIN] error.

56664 Austin Group Defect 1162 is applied, changing “an error code” to “[EOWNERDEAD]”.

56665 Austin Group Defects 1216 and 1485 are applied, adding *pthread\_cond\_clockwait()*.

56666 **NAME**

56667 pthread\_cond\_destroy, pthread\_cond\_init — destroy and initialize condition variables

56668 **SYNOPSIS**

```
56669 #include <pthread.h>
56670 int pthread_cond_destroy(pthread_cond_t *cond);
56671 int pthread_cond_init(pthread_cond_t *restrict cond,
56672     const pthread_condattr_t *restrict attr);
56673 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

56674 **DESCRIPTION**

56675 The *pthread\_cond\_destroy()* function shall destroy the given condition variable specified by *cond*;  
 56676 the object becomes, in effect, uninitialized. An implementation may cause *pthread\_cond\_destroy()*  
 56677 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can  
 56678 be reinitialized using *pthread\_cond\_init()*; the results of otherwise referencing the object after it  
 56679 has been destroyed are undefined.

56680 It shall be safe to destroy an initialized condition variable upon which no threads are currently  
 56681 blocked. Attempting to destroy a condition variable upon which other threads are currently  
 56682 blocked results in undefined behavior.

56683 The *pthread\_cond\_init()* function shall initialize the condition variable referenced by *cond* with  
 56684 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be  
 56685 used; the effect is the same as passing the address of a default condition variable attributes  
 56686 object. Upon successful initialization, the state of the condition variable shall become initialized.

56687 See [Section 2.9.9](#) (on page 548) for further requirements.

56688 Attempting to initialize an already initialized condition variable results in undefined behavior.

56689 In cases where default condition variable attributes are appropriate, the macro  
 56690 PTHREAD\_COND\_INITIALIZER can be used to initialize condition variables. The effect shall  
 56691 be equivalent to dynamic initialization by a call to *pthread\_cond\_init()* with parameter *attr*  
 56692 specified as NULL, except that no error checks are performed.

56693 The behavior is undefined if the value specified by the *cond* argument to *pthread\_cond\_destroy()*  
 56694 does not refer to an initialized condition variable.

56695 The behavior is undefined if the value specified by the *attr* argument to *pthread\_cond\_init()* does  
 56696 not refer to an initialized condition variable attributes object.

56697 **RETURN VALUE**

56698 If successful, the *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions shall return zero;  
 56699 otherwise, an error number shall be returned to indicate the error.

56700 **ERRORS**

56701 The *pthread\_cond\_init()* function shall fail if:

56702 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 56703 another condition variable.

56704 [ENOMEM] Insufficient memory exists to initialize the condition variable.

56705 These functions shall not return an error code of [EINTR].

56706 **EXAMPLES**

56707 A condition variable can be destroyed immediately after all the threads that are blocked on it are  
56708 awakened. For example, consider the following code:

```
56709 struct list {
56710     pthread_mutex_t lm;
56711     ...
56712 }

56713 struct elt {
56714     key k;
56715     int busy;
56716     pthread_cond_t notbusy;
56717     ...
56718 }

56719 /* Find a list element and reserve it. */
56720 struct elt *
56721 list_find(struct list *lp, key k)
56722 {
56723     struct elt *ep;

56724     pthread_mutex_lock(&lp->lm);
56725     while ((ep = find_elt(l, k) != NULL) && ep->busy)
56726         pthread_cond_wait(&ep->notbusy, &lp->lm);
56727     if (ep != NULL)
56728         ep->busy = 1;
56729     pthread_mutex_unlock(&lp->lm);
56730     return(ep);
56731 }
56732 delete_elt(struct list *lp, struct elt *ep)
56733 {
56734     pthread_mutex_lock(&lp->lm);
56735     assert(ep->busy);
56736     ... remove ep from list ...
56737     ep->busy = 0; /* Paranoid. */
56738 (A) pthread_cond_broadcast(&ep->notbusy);
56739     pthread_mutex_unlock(&lp->lm);
56740 (B) pthread_cond_destroy(&ep->notbusy);
56741     free(ep);
56742 }
```

56743 In this example, the condition variable and its list element may be freed (line B) immediately  
56744 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that  
56745 no other thread can touch the element to be deleted.

56746 **APPLICATION USAGE**

56747 None.

56748 **RATIONALE**

56749 If an implementation detects that the value specified by the *cond* argument to  
56750 *pthread\_cond\_destroy()* does not refer to an initialized condition variable, it is recommended that  
56751 the function should fail and report an [EINVAL] error.

56752 If an implementation detects that the value specified by the *cond* argument to  
56753 *pthread\_cond\_destroy()* or *pthread\_cond\_init()* refers to a condition variable that is in use (for



56754 example, in a *pthread\_cond\_wait()* call) by another thread, or detects that the value specified by  
56755 the *cond* argument to *pthread\_cond\_init()* refers to an already initialized condition variable, it is  
56756 recommended that the function should fail and report an [EBUSY] error.

56757 If an implementation detects that the value specified by the *attr* argument to *pthread\_cond\_init()*  
56758 does not refer to an initialized condition variable attributes object, it is recommended that the  
56759 function should fail and report an [EINVAL] error.

56760 See also *pthread\_mutex\_destroy()*.

#### 56761 FUTURE DIRECTIONS

56762 None.

#### 56763 SEE ALSO

56764 *pthread\_cond\_broadcast()*, *pthread\_cond\_clockwait()*, *pthread\_mutex\_destroy()*

56765 XBD <pthread.h>

#### 56766 CHANGE HISTORY

56767 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 56768 Issue 6

56769 The *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions are marked as part of the Threads  
56770 option.

56771 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

56772 The **restrict** keyword is added to the *pthread\_cond\_init()* prototype for alignment with the  
56773 ISO/IEC 9899:1999 standard.

#### 56774 Issue 7

56775 The *pthread\_cond\_destroy()* and *pthread\_cond\_init()* functions are moved from the Threads option  
56776 to the Base.

56777 The [EINVAL] error for an uninitialized condition variable and an uninitialized condition  
56778 variable attributes object is removed; this condition results in undefined behavior.

56779 The [EBUSY] error for a condition variable already in use or an already initialized condition  
56780 variable is removed; this condition results in undefined behavior.

56781 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0455 [70] is applied.

56782 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0269 [972] and XSH/TC2-2008/0270  
56783 [910] are applied.

56784 **NAME**

56785 pthread\_cond\_signal — signal a condition

56786 **SYNOPSIS**

56787 #include <pthread.h>

56788 int pthread\_cond\_signal(pthread\_cond\_t \*cond);

56789 **DESCRIPTION**

56790 Refer to [pthread\\_cond\\_broadcast\(\)](#).

56791 **NAME**

56792 pthread\_cond\_timedwait, pthread\_cond\_wait — wait on a condition

56793 **SYNOPSIS**

56794 #include &lt;pthread.h&gt;

56795 int pthread\_cond\_timedwait(pthread\_cond\_t \*restrict cond,

56796 pthread\_mutex\_t \*restrict mutex,

56797 const struct timespec \*restrict abstime);

56798 int pthread\_cond\_wait(pthread\_cond\_t \*restrict cond,

56799 pthread\_mutex\_t \*restrict mutex);

56800 **DESCRIPTION**56801 Refer to [pthread\\_cond\\_clockwait\(\)](#).

56802 **NAME**

56803 pthread\_condattr\_destroy, pthread\_condattr\_init — destroy and initialize the condition variable  
56804 attributes object

56805 **SYNOPSIS**

```
56806 #include <pthread.h>  
  
56807 int pthread_condattr_destroy(pthread_condattr_t *attr);  
56808 int pthread_condattr_init(pthread_condattr_t *attr);
```

56809 **DESCRIPTION**

56810 The *pthread\_condattr\_destroy()* function shall destroy a condition variable attributes object; the  
56811 object becomes, in effect, uninitialized. An implementation may cause *pthread\_condattr\_destroy()*  
56812 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be  
56813 reinitialized using *pthread\_condattr\_init()*; the results of otherwise referencing the object after it  
56814 has been destroyed are undefined.

56815 The *pthread\_condattr\_init()* function shall initialize a condition variable attributes object *attr* with  
56816 the default value for all of the attributes defined by the implementation.

56817 Results are undefined if *pthread\_condattr\_init()* is called specifying an already initialized *attr*  
56818 attributes object.

56819 After a condition variable attributes object has been used to initialize one or more condition  
56820 variables, any function affecting the attributes object (including destruction) shall not affect any  
56821 previously initialized condition variables.

56822 This volume of POSIX.1-2024 requires two attributes, the *clock* attribute and the *process-shared*  
56823 attribute.

56824 Additional attributes, their default values, and the names of the associated functions to get and  
56825 set those attribute values are implementation-defined.

56826 The behavior is undefined if the value specified by the *attr* argument to  
56827 *pthread\_condattr\_destroy()* does not refer to an initialized condition variable attributes object.

56828 **RETURN VALUE**

56829 If successful, the *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions shall return  
56830 zero; otherwise, an error number shall be returned to indicate the error.

56831 **ERRORS**

56832 The *pthread\_condattr\_init()* function shall fail if:

56833 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

56834 These functions shall not return an error code of [EINTR].

56835 **EXAMPLES**

56836 None.

56837 **APPLICATION USAGE**

56838 None.

56839 **RATIONALE**

56840 A *process-shared* attribute has been defined for condition variables for the same reason it has been  
56841 defined for mutexes.

56842 If an implementation detects that the value specified by the *attr* argument to  
56843 *pthread\_condattr\_destroy()* does not refer to an initialized condition variable attributes object, it is  
56844 recommended that the function should fail and report an [EINVAL] error.

56845 See also *pthread\_attr\_destroy()* and *pthread\_mutex\_destroy()*.

56846 **FUTURE DIRECTIONS**

56847 None.

56848 **SEE ALSO**

56849 *pthread\_attr\_destroy()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_getpshared()*, *pthread\_create()*,  
56850 *pthread\_mutex\_destroy()*

56851 XBD <pthread.h>

56852 **CHANGE HISTORY**

56853 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

56854 **Issue 6**

56855 The *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions are marked as part of the  
56856 Threads option.

56857 **Issue 7**

56858 The *pthread\_condattr\_destroy()* and *pthread\_condattr\_init()* functions are moved from the Threads  
56859 option to the Base.

56860 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this  
56861 condition results in undefined behavior.

56862 **NAME**

56863 pthread\_condattr\_getclock, pthread\_condattr\_setclock — get and set the clock selection  
56864 condition variable attribute

56865 **SYNOPSIS**

```
56866 #include <pthread.h>
56867 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
56868 clockid_t *restrict clock_id);
56869 int pthread_condattr_setclock(pthread_condattr_t *attr,
56870 clockid_t clock_id);
```

56871 **DESCRIPTION**

56872 The *pthread\_condattr\_getclock()* function shall obtain the value of the *clock* attribute from the  
56873 attributes object referenced by *attr*.

56874 The *pthread\_condattr\_setclock()* function shall set the *clock* attribute in an initialized attributes  
56875 object referenced by *attr*. If *pthread\_condattr\_setclock()* is called with a *clock\_id* argument that  
56876 refers to a CPU-time clock, the call shall fail.

56877 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of  
56878 *pthread\_cond\_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.  
56879 The *clock* attribute shall have no effect on the *pthread\_cond\_clockwait()* function.

56880 The behavior is undefined if the value specified by the *attr* argument to  
56881 *pthread\_condattr\_getclock()* or *pthread\_condattr\_setclock()* does not refer to an initialized condition  
56882 variable attributes object.

56883 **RETURN VALUE**

56884 If successful, the *pthread\_condattr\_getclock()* function shall return zero and store the value of the  
56885 clock attribute of *attr* into the object referenced by the *clock\_id* argument. Otherwise, an error  
56886 number shall be returned to indicate the error.

56887 If successful, the *pthread\_condattr\_setclock()* function shall return zero; otherwise, an error  
56888 number shall be returned to indicate the error.

56889 **ERRORS**

56890 The *pthread\_condattr\_setclock()* function may fail if:

56891 [EINVAL] The value specified by *clock\_id* does not refer to a known clock, or is a CPU-  
56892 time clock.

56893 These functions shall not return an error code of [EINTR].

56894 **EXAMPLES**

56895 None.

56896 **APPLICATION USAGE**

56897 None.

56898 **RATIONALE**

56899 If an implementation detects that the value specified by the *attr* argument to  
56900 *pthread\_condattr\_getclock()* or *pthread\_condattr\_setclock()* does not refer to an initialized condition  
56901 variable attributes object, it is recommended that the function should fail and report an  
56902 [EINVAL] error.

56903 **FUTURE DIRECTIONS**

56904 None.

56905 **SEE ALSO**56906 *pthread\_cond\_clockwait()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_destroy()*,  
56907 *pthread\_condattr\_getpshared()*, *pthread\_create()*, *pthread\_mutex\_destroy()*

56908 XBD &lt;pthread.h&gt;

56909 **CHANGE HISTORY**

56910 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

56911 **Issue 7**56912 The *pthread\_condattr\_getclock()* and *pthread\_condattr\_setclock()* functions are moved from the  
56913 Clock Selection option to the Base.56914 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this  
56915 condition results in undefined behavior.56916 **Issue 8**56917 Austin Group Defect 1216 is applied, adding *pthread\_cond\_clockwait()*.

56918 **NAME**

56919 pthread\_condattr\_getpshared, pthread\_condattr\_setpshared — get and set the process-shared  
56920 condition variable attributes

56921 **SYNOPSIS**

```
56922 TSH #include <pthread.h>
56923
56923 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
56924 int *restrict pshared);
56925 int pthread_condattr_setpshared(pthread_condattr_t *attr,
56926 int pshared);
```

56927 **DESCRIPTION**

56928 The `pthread_condattr_getpshared()` function shall obtain the value of the *process-shared* attribute  
56929 from the attributes object referenced by *attr*.

56930 The `pthread_condattr_setpshared()` function shall set the *process-shared* attribute in an initialized  
56931 attributes object referenced by *attr*.

56932 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a condition  
56933 variable to be operated upon by any thread that has access to the memory where the condition  
56934 variable is allocated, even if the condition variable is allocated in memory that is shared by  
56935 multiple processes. See [Section 2.9.9](#) (on page 548) for further requirements. The default value of  
56936 the attribute is `PTHREAD_PROCESS_PRIVATE`.

56937 The behavior is undefined if the value specified by the *attr* argument to  
56938 `pthread_condattr_getpshared()` or `pthread_condattr_setpshared()` does not refer to an initialized  
56939 condition variable attributes object.

56940 **RETURN VALUE**

56941 If successful, the `pthread_condattr_setpshared()` function shall return zero; otherwise, an error  
56942 number shall be returned to indicate the error.

56943 If successful, the `pthread_condattr_getpshared()` function shall return zero and store the value of  
56944 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,  
56945 an error number shall be returned to indicate the error.

56946 **ERRORS**

56947 The `pthread_condattr_setpshared()` function may fail if:

56948 [EINVAL] The new value specified for the attribute is outside the range of legal values  
56949 for that attribute.

56950 These functions shall not return an error code of [EINTR].

56951 **EXAMPLES**

56952 None.

56953 **APPLICATION USAGE**

56954 None.

56955 **RATIONALE**

56956 If an implementation detects that the value specified by the *attr* argument to  
56957 `pthread_condattr_getpshared()` or `pthread_condattr_setpshared()` does not refer to an initialized  
56958 condition variable attributes object, it is recommended that the function should fail and report  
56959 an [EINVAL] error.



56960 **FUTURE DIRECTIONS**

56961 None.

56962 **SEE ALSO**56963 *pthread\_create()*, *pthread\_cond\_destroy()*, *pthread\_condattr\_destroy()*, *pthread\_mutex\_destroy()*

56964 XBD &lt;pthread.h&gt;

56965 **CHANGE HISTORY**

56966 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

56967 **Issue 6**56968 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions are marked as part  
56969 of the Threads and Thread Process-Shared Synchronization options.56970 The **restrict** keyword is added to the *pthread\_condattr\_getpshared()* prototype for alignment with  
56971 the ISO/IEC 9899:1999 standard.56972 **Issue 7**56973 The *pthread\_condattr\_getpshared()* and *pthread\_condattr\_setpshared()* functions are marked only as  
56974 part of the Thread Process-Shared Synchronization option as the Threads option is now part of  
56975 the Base.56976 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this  
56977 condition results in undefined behavior.56978 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0272 [972] and XSH/TC2-2008/0273  
56979 [757] are applied.

56980 **NAME**

56981 pthread\_condattr\_init — initialize the condition variable attributes object

56982 **SYNOPSIS**

56983 #include <pthread.h>

56984 int pthread\_condattr\_init(pthread\_condattr\_t \*attr);

56985 **DESCRIPTION**

56986 Refer to *pthread\_condattr\_destroy()*.

56987 **NAME**

56988 pthread\_condattr\_setclock — set the clock selection condition variable attribute

56989 **SYNOPSIS**

56990 #include &lt;pthread.h&gt;

56991 int pthread\_condattr\_setclock(pthread\_condattr\_t \*attr,  
56992 clockid\_t clock\_id);56993 **DESCRIPTION**56994 Refer to [pthread\\_condattr\\_getclock\(\)](#).

56995 **NAME**

56996 pthread\_condattr\_setpshared — set the process-shared condition variable attribute

56997 **SYNOPSIS**

```
56998 TSH #include <pthread.h>
56999 int pthread_condattr_setpshared(pthread_condattr_t *attr,
57000 int pshared);
```

57001 **DESCRIPTION**

57002 Refer to [pthread\\_condattr\\_getpshared\(\)](#).

57003 **NAME**

57004 pthread\_create — thread creation

57005 **SYNOPSIS**

57006 #include &lt;pthread.h&gt;

```
57007 int pthread_create(pthread_t *restrict thread,
57008                  const pthread_attr_t *restrict attr,
57009                  void *(*start_routine)(void*), void *restrict arg);
```

57010 **DESCRIPTION**

57011 The *pthread\_create()* function shall create a new thread, with attributes specified by *attr*, within a  
 57012 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are  
 57013 modified later, the thread's attributes shall not be affected. Upon successful completion,  
 57014 *pthread\_create()* shall store the ID of the created thread in the location referenced by *thread*.

57015 The thread is created executing *start\_routine* with *arg* as its sole argument. If the *start\_routine*  
 57016 returns, the effect shall be as if there was an implicit call to *pthread\_exit()* using the return value  
 57017 of *start\_routine* as the exit status. Note that the thread in which *main()* was originally invoked  
 57018 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call to  
 57019 *exit()* using the return value of *main()* as the exit status.

57020 The signal state of the new thread shall be initialized as follows:

- 57021 • The signal mask shall be inherited from the creating thread.
- 57022 • The set of signals pending for the new thread shall be empty.

57023 XSI The thread-local current locale and the alternate stack shall not be inherited.

57024 The floating-point environment shall be inherited from the creating thread.

57025 If *pthread\_create()* fails, no new thread is created and the contents of the location referenced by  
 57026 *thread* are undefined.

57027 TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock  
 57028 accessible, and the initial value of this clock shall be set to zero.

57029 The behavior is undefined if the value specified by the *attr* argument to *pthread\_create()* does not  
 57030 refer to an initialized thread attributes object.

57031 **RETURN VALUE**

57032 If successful, the *pthread\_create()* function shall return zero; otherwise, an error number shall be  
 57033 returned to indicate the error.

57034 **ERRORS**

57035 The *pthread\_create()* function shall fail if:

57036 [EAGAIN] The system lacked the necessary resources to create another thread, or the  
 57037 system-imposed limit on the total number of threads in a process  
 57038 {PTHREAD\_THREADS\_MAX} would be exceeded.

57039 [EPERM] The caller does not have appropriate privileges to set the required scheduling  
 57040 parameters or scheduling policy.

57041 The *pthread\_create()* function shall not return an error code of [EINTR].

57042 **EXAMPLES**

57043 None.

57044 **APPLICATION USAGE**

57045 There is no requirement on the implementation that the ID of the created thread be available  
 57046 before the newly created thread starts executing. The calling thread can obtain the ID of the  
 57047 created thread through the *thread* argument of the *pthread\_create()* function, and the newly  
 57048 created thread can obtain its ID by a call to *pthread\_self()*.

57049 **RATIONALE**

57050 A suggested alternative to *pthread\_create()* would be to define two separate operations: create  
 57051 and start. Some applications would find such behavior more natural. Ada, in particular,  
 57052 separates the “creation” of a task from its “activation”.

57053 Splitting the operation was rejected by the standard developers for many reasons:

- 57054 • The number of calls required to start a thread would increase from one to two and thus  
 57055 place an additional burden on applications that do not require the additional  
 57056 synchronization. The second call, however, could be avoided by the additional  
 57057 complication of a start-up state attribute.
- 57058 • An extra state would be introduced: “created but not started”. This would require the  
 57059 standard to specify the behavior of the thread operations when the target has not yet  
 57060 started executing.
- 57061 • For those applications that require such behavior, it is possible to simulate the two separate  
 57062 steps with the facilities that are currently provided. The *start\_routine()* can synchronize by  
 57063 waiting on a condition variable that is signaled by the start operation.

57064 An Ada implementor can choose to create the thread at either of two points in the Ada program:  
 57065 when the task object is created, or when the task is activated (generally at a “begin”). If the first  
 57066 approach is adopted, the *start\_routine()* needs to wait on a condition variable to receive the order  
 57067 to begin “activation”. The second approach requires no such condition variable or extra  
 57068 synchronization. In either approach, a separate Ada task control block would need to be created  
 57069 when the task object is created to hold rendezvous queues, and so on.

57070 An extension of the preceding model would be to allow the state of the thread to be modified  
 57071 between the create and start. This would allow the thread attributes object to be eliminated. This  
 57072 has been rejected because:

- 57073 • All state in the thread attributes object has to be able to be set for the thread. This would  
 57074 require the definition of functions to modify thread attributes. There would be no  
 57075 reduction in the number of function calls required to set up the thread. In fact, for an  
 57076 application that creates all threads using identical attributes, the number of function calls  
 57077 required to set up the threads would be dramatically increased. Use of a thread attributes  
 57078 object permits the application to make one set of attribute setting function calls.  
 57079 Otherwise, the set of attribute setting function calls needs to be made for each thread  
 57080 creation.
- 57081 • Depending on the implementation architecture, functions to set thread state would require  
 57082 kernel calls, or for other implementation reasons would not be able to be implemented as  
 57083 macros, thereby increasing the cost of thread creation.
- 57084 • The ability for applications to segregate threads by class would be lost.

57085 Another suggested alternative uses a model similar to that for process creation, such as “thread  
 57086 fork”. The fork semantics would provide more flexibility and the “create” function can be  
 57087 implemented simply by doing a thread fork followed immediately by a call to the desired “start

- 57088 routine” for the thread. This alternative has these problems:
- 57089       • For many implementations, the entire stack of the calling thread would need to be
- 57090       duplicated, since in many architectures there is no way to determine the size of the calling
- 57091       frame.
- 57092       • Efficiency is reduced since at least some part of the stack has to be copied, even though in
- 57093       most cases the thread never needs the copied context, since it merely calls the desired start
- 57094       routine.

57095 If an implementation detects that the value specified by the *attr* argument to *pthread\_create()*

57096 does not refer to an initialized thread attributes object, it is recommended that the function

57097 should fail and report an [EINVAL] error.

#### 57098 FUTURE DIRECTIONS

57099 None.

#### 57100 SEE ALSO

57101 *fork()*, *pthread\_exit()*, *pthread\_join()*

57102 XBD Section 4.15.2 (on page 104), [<pthread.h>](#)

#### 57103 CHANGE HISTORY

57104 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 57105 Issue 6

57106 The *pthread\_create()* function is marked as part of the Threads option.

57107 The following new requirements on POSIX implementations derive from alignment with the

57108 Single UNIX Specification:

- 57109       • The [EPERM] mandatory error condition is added.

57110 The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

57111 The **restrict** keyword is added to the *pthread\_create()* prototype for alignment with the

57112 ISO/IEC 9899:1999 standard.

57113 The DESCRIPTION is updated to make it explicit that the floating-point environment is

57114 inherited from the creating thread.

57115 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the

57116 alternate stack is not inherited.

57117 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/93 is applied, updating the ERRORS

57118 section to remove the mandatory [EINVAL] error (“The value specified by *attr* is invalid”), and

57119 adding the optional [EINVAL] error (“The attributes specified by *attr* are invalid”).

57120 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/94 is applied, adding the APPLICATION

57121 USAGE section.

#### 57122 Issue 7

57123 The *pthread\_create()* function is moved from the Threads option to the Base.

57124 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition

57125 results in undefined behavior.

57126 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0458 [302] is applied.

57127 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0274 [849] is applied.

57128 **NAME**

57129 pthread\_detach — detach a thread

57130 **SYNOPSIS**

57131 #include &lt;pthread.h&gt;

57132 int pthread\_detach(pthread\_t thread);

57133 **DESCRIPTION**

57134 The *pthread\_detach()* function shall change the thread *thread* from joinable to detached, indicating  
57135 to the implementation that storage for the thread can be reclaimed when the thread terminates.  
57136 If *thread* has not terminated, *pthread\_detach()* shall not cause it to terminate, but shall prevent the  
57137 thread from becoming a zombie thread when it does terminate.

57138 The behavior is undefined if the value specified by the *thread* argument to *pthread\_detach()* does  
57139 not refer to a joinable thread.

57140 **RETURN VALUE**

57141 If the call succeeds, *pthread\_detach()* shall return 0; otherwise, an error number shall be returned  
57142 to indicate the error.

57143 **ERRORS**57144 The *pthread\_detach()* function shall not return an error code of [EINTR].57145 **EXAMPLES**

57146 None.

57147 **APPLICATION USAGE**

57148 None.

57149 **RATIONALE**

57150 The *pthread\_join()* or *pthread\_detach()* functions should eventually be called for every thread that  
57151 is created so that storage associated with the thread may be reclaimed.

57152 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation  
57153 attribute is sufficient, since a thread need never be dynamically detached. However, need arises  
57154 in at least two cases:

57155 1. In a cancellation handler for a *pthread\_join()* it is nearly essential to have a  
57156 *pthread\_detach()* function in order to detach the thread on which *pthread\_join()* was  
57157 waiting. Without it, it would be necessary to have the handler do another *pthread\_join()* to  
57158 attempt to detach the thread, which would both delay the cancellation processing for an  
57159 unbounded period and introduce a new call to *pthread\_join()*, which might itself need a  
57160 cancellation handler. A dynamic detach is nearly essential in this case.

57161 2. In order to detach the “initial thread” (as may be desirable in processes that set up server  
57162 threads).

57163 If an implementation detects that the value specified by the *thread* argument to *pthread\_detach()*  
57164 does not refer to a joinable thread, it is recommended that the function should fail and report an  
57165 [EINVAL] error.

57166 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
57167 that the function should fail and report an [ESRCH] error.

57168 **FUTURE DIRECTIONS**

57169 None.



57170 **SEE ALSO**57171 [pthread\\_join\(\)](#)

57172 XBD &lt;pthread.h&gt;

57173 **CHANGE HISTORY**

57174 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

57175 **Issue 6**57176 The *pthread\_detach()* function is marked as part of the Threads option.57177 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/95 is applied, updating the ERRORS  
57178 section so that the [EINVAL] and [ESRCH] error cases become optional.57179 **Issue 7**57180 The *pthread\_detach()* function is moved from the Threads option to the Base.

57181 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

57182 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined  
57183 behavior.57184 **Issue 8**57185 Austin Group Defect 792 is applied, clarifying that detaching a live thread prevents it becoming  
57186 a zombie thread when it terminates.57187 Austin Group Defect 1167 is applied, clarifying that a thread is no longer joinable after  
57188 *pthread\_detach()* has been called for it.

57189 **NAME**

57190 pthread\_equal — compare thread IDs

57191 **SYNOPSIS**

57192 #include &lt;pthread.h&gt;

57193 int pthread\_equal(pthread\_t t1, pthread\_t t2);

57194 **DESCRIPTION**57195 This function shall compare the thread IDs *t1* and *t2*.57196 **RETURN VALUE**57197 The *pthread\_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero  
57198 shall be returned.57199 If either *t1* or *t2* is not a valid thread ID and is not equal to PTHREAD\_NULL, the behavior is  
57200 undefined.57201 **ERRORS**

57202 No errors are defined.

57203 The *pthread\_equal()* function shall not return an error code of [EINTR].57204 **EXAMPLES**

57205 None.

57206 **APPLICATION USAGE**

57207 None.

57208 **RATIONALE**57209 Implementations may choose to define a thread ID as a structure. This allows additional  
57210 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence  
57211 number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).  
57212 Since the C language does not support comparison on structure types, the *pthread\_equal()*  
57213 function is provided to compare thread IDs.57214 **FUTURE DIRECTIONS**

57215 None.

57216 **SEE ALSO**57217 *pthread\_create()*, *pthread\_self()*

57218 XBD &lt;pthread.h&gt;

57219 **CHANGE HISTORY**

57220 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

57221 **Issue 6**57222 The *pthread\_equal()* function is marked as part of the Threads option.57223 **Issue 7**57224 The *pthread\_equal()* function is moved from the Threads option to the Base.57225 **Issue 8**57226 Austin Group Defect 599 is applied, changing the RETURN VALUE section to mention  
57227 PTHREAD\_NULL.

57228 **NAME**

57229 pthread\_exit — thread termination

57230 **SYNOPSIS**

57231 #include &lt;pthread.h&gt;

57232 \_Noreturn void pthread\_exit(void \*value\_ptr);

57233 **DESCRIPTION**

57234 The *pthread\_exit()* function shall terminate the calling thread and make the value *value\_ptr*  
57235 available to any successful join with the terminating thread. Any cancellation cleanup handlers  
57236 that have been pushed and not yet popped shall be popped in the reverse order that they were  
57237 pushed and then executed. After all cancellation cleanup handlers have been executed, if the  
57238 thread has any thread-specific data (whether associated with key type *tss\_t* or *pthread\_key\_t*),  
57239 appropriate destructor functions shall be called in an unspecified order. Thread termination  
57240 does not release any application visible process resources, including, but not limited to, mutexes  
57241 and file descriptors, nor does it perform any process-level cleanup actions, including, but not  
57242 limited to, calling any *atexit()* routines that may exist.

57243 An implicit call to *pthread\_exit()* is made when a thread that was not created using *thrd\_create()*,  
57244 and is not the thread in which *main()* was first invoked, returns from the start routine that was  
57245 used to create it. The function's return value shall serve as the thread's exit status.

57246 The behavior of *pthread\_exit()* is undefined if called from a cancellation cleanup handler or  
57247 destructor function that was invoked as a result of either an implicit or explicit call to  
57248 *pthread\_exit()*.

57249 After a thread has terminated, the result of access to local (auto) variables of the thread is  
57250 undefined. Thus, references to local variables of the exiting thread should not be used for the  
57251 *pthread\_exit()* *value\_ptr* parameter value.

57252 The process shall exit with an exit status of 0 after the last thread has been terminated. The  
57253 behavior shall be as if the implementation called *exit()* with a zero argument at thread  
57254 termination time.

57255 **RETURN VALUE**57256 The *pthread\_exit()* function cannot return to its caller.57257 **ERRORS**

57258 No errors are defined.

57259 **EXAMPLES**

57260 None.

57261 **APPLICATION USAGE**

57262 Calls to *pthread\_exit()* should not be made from threads created using *thrd\_create()*, as their exit  
57263 status has a different type (**int** instead of **void \***). If *pthread\_exit()* is called from the initial thread  
57264 and it is not the last thread to terminate, other threads should not try to obtain its exit status  
57265 using *thrd\_join()*.

57266 **RATIONALE**

57267 The normal mechanism by which a thread that was started using *pthread\_create()* terminates is to  
57268 return from the routine that was specified in the *pthread\_create()* call that started it. The  
57269 *pthread\_exit()* function provides the capability for a thread to terminate without requiring a  
57270 return from the start routine of that thread, thereby providing a function analogous to *exit()*.

57271 Regardless of the method of thread termination, any cancellation cleanup handlers that have  
57272 been pushed and not yet popped are executed, and the destructors for any existing thread-  
57273 specific data are executed. This volume of POSIX.1-2024 requires that cancellation cleanup

57274 handlers be popped and called in order. After all cancellation cleanup handlers have been  
57275 executed, thread-specific data destructors are called, in an unspecified order, for each item of  
57276 thread-specific data that exists in the thread. This ordering is necessary because cancellation  
57277 cleanup handlers may rely on thread-specific data.

57278 As the meaning of the status is determined by the application (except when the thread has been  
57279 canceled, in which case it is PTHREAD\_CANCELED), the implementation has no idea what an  
57280 illegal status value is, which is why no address error checking is done.

#### 57281 **FUTURE DIRECTIONS**

57282 None.

#### 57283 **SEE ALSO**

57284 *exit()*, *pthread\_create()*, *pthread\_join()*, *pthread\_key\_create()*, *thrd\_create()*, *thrd\_exit()*, *tss\_create()*

57285 XBD <pthread.h>

#### 57286 **CHANGE HISTORY**

57287 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 57288 **Issue 6**

57289 The *pthread\_exit()* function is marked as part of the Threads option.

#### 57290 **Issue 7**

57291 The *pthread\_exit()* function is moved from the Threads option to the Base.

#### 57292 **Issue 8**

57293 Austin Group Defect 1302 is applied, adding `_Noreturn` to the SYNOPSIS, and updating the  
57294 page to account for the addition of <threads.h> interfaces.

57295 **NAME**

57296 pthread\_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME**  
57297 **THREADS**)

57298 **SYNOPSIS**

```
57299 TCT #include <pthread.h>
57300 #include <time.h>
57301 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

57302 **DESCRIPTION**

57303 The *pthread\_getcpuclockid()* function shall return in *clock\_id* the clock ID of the CPU-time clock of  
57304 the thread specified by *thread\_id*, if the thread specified by *thread\_id* exists.

57305 **RETURN VALUE**

57306 Upon successful completion, *pthread\_getcpuclockid()* shall return zero; otherwise, an error  
57307 number shall be returned to indicate the error.

57308 **ERRORS**

57309 No errors are defined.

57310 **EXAMPLES**

57311 None.

57312 **APPLICATION USAGE**

57313 The *pthread\_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not  
57314 be provided on all implementations.

57315 **RATIONALE**

57316 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
57317 that the function should fail and report an [ESRCH] error.

57318 **FUTURE DIRECTIONS**

57319 None.

57320 **SEE ALSO**

57321 *clock\_getcpuclockid()*, *clock\_getres()*, *timer\_create()*

57322 XBD [<pthread.h>](#), [<time.h>](#)

57323 **CHANGE HISTORY**

57324 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

57325 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

57326 **Issue 7**

57327 The *pthread\_getcpuclockid()* function is marked only as part of the Thread CPU-Time Clocks  
57328 option as the Threads option is now part of the Base.

57329 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

57330 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0275 [757] is applied.

## 57331 NAME

57332 pthread\_getschedparam, pthread\_setschedparam — dynamic thread scheduling parameters  
57333 access (REALTIME THREADS)

## 57334 SYNOPSIS

```
57335 TPS #include <pthread.h>
57336
57336 int pthread_getschedparam(pthread_t thread, int *restrict policy,
57337 struct sched_param *restrict param);
57338 int pthread_setschedparam(pthread_t thread, int policy,
57339 const struct sched_param *param);
```

## 57340 DESCRIPTION

57341 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall, respectively, get and set  
57342 the scheduling policy and parameters of individual threads within a multi-threaded process to  
57343 be retrieved and set. For SCHED\_FIFO and SCHED\_RR, the only required member of the  
57344 **sched\_param** structure is the priority *sched\_priority*. For SCHED\_OTHER, the affected  
57345 scheduling parameters are implementation-defined.

57346 The *pthread\_getschedparam()* function shall retrieve the scheduling policy and scheduling  
57347 parameters for the thread whose thread ID is given by *thread* and shall store those values in  
57348 *policy* and *param*, respectively. The priority value returned from *pthread\_getschedparam()* shall be  
57349 the value specified by the most recent *pthread\_setschedparam()*, *pthread\_setschedprio()*, or  
57350 *pthread\_create()* call affecting the target thread. It shall not reflect any temporary adjustments to  
57351 its priority as a result of any priority inheritance or ceiling functions. The *pthread\_setschedparam()*  
57352 function shall set the scheduling policy and associated scheduling parameters for the thread  
57353 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy*  
57354 and *param*, respectively.

57355 The *policy* parameter may have the value SCHED\_OTHER, SCHED\_FIFO, or SCHED\_RR. The  
57356 scheduling parameters for the SCHED\_OTHER policy are implementation-defined. The  
57357 SCHED\_FIFO and SCHED\_RR policies shall have a single scheduling parameter, *priority*.

57358 TSP If \_POSIX\_THREAD\_SPORADIC\_SERVER is defined, then the *policy* argument may have the  
57359 value SCHED\_SPORADIC, with the exception for the *pthread\_setschedparam()* function that if the  
57360 scheduling policy was not SCHED\_SPORADIC at the time of the call, it is implementation-  
57361 defined whether the function is supported; in other words, the implementation need not allow  
57362 the application to dynamically change the scheduling policy to SCHED\_SPORADIC. The  
57363 sporadic server scheduling policy has the associated parameters *sched\_ss\_low\_priority*,  
57364 *sched\_ss\_repl\_period*, *sched\_ss\_init\_budget*, *sched\_priority*, and *sched\_ss\_max\_repl*. The specified  
57365 *sched\_ss\_repl\_period* shall be greater than or equal to the specified *sched\_ss\_init\_budget* for the  
57366 function to succeed; if it is not, then the function shall fail. The value of *sched\_ss\_max\_repl* shall  
57367 be within the inclusive range [1,{SS\_REPL\_MAX}] for the function to succeed; if not, the function  
57368 shall fail. It is unspecified whether the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values are  
57369 stored as provided by this function or are rounded to align with the resolution of the clock being  
57370 used.

57371 If the *pthread\_setschedparam()* function fails, the scheduling parameters shall not be changed for  
57372 the target thread.

## 57373 RETURN VALUE

57374 If successful, the *pthread\_getschedparam()* and *pthread\_setschedparam()* functions shall return zero;  
57375 otherwise, an error number shall be returned to indicate the error.

57376 **ERRORS**57377 The *pthread\_setschedparam()* function shall fail if:57378 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an  
57379 unsupported value.57380 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to  
57381 SCHED\_SPORADIC, and the implementation does not support this change.57382 The *pthread\_setschedparam()* function may fail if:57383 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated  
57384 with the scheduling policy *policy* is invalid.57385 [EPERM] The caller does not have appropriate privileges to set either the scheduling  
57386 parameters or the scheduling policy of the specified thread.57387 [EPERM] The implementation does not allow the application to modify one of the  
57388 parameters to the value specified.

57389 These functions shall not return an error code of [EINTR].

57390 **EXAMPLES**

57391 None.

57392 **APPLICATION USAGE**

57393 None.

57394 **RATIONALE**57395 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
57396 that the function should fail and report an [ESRCH] error.57397 **FUTURE DIRECTIONS**

57398 None.

57399 **SEE ALSO**57400 *pthread\_setschedprio()*, *sched\_getparam()*, *sched\_getscheduler()*

57401 XBD &lt;pthread.h&gt;, &lt;sched.h&gt;

57402 **CHANGE HISTORY**

57403 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

57404 **Issue 6**57405 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions are marked as part of the  
57406 Threads and Thread Execution Scheduling options.57407 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
57408 implementation does not support the Thread Execution Scheduling option.57409 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the  
57410 *pthread\_setschedparam()* function so that its second argument is of type **int**.

57411 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

57412 The **restrict** keyword is added to the *pthread\_getschedparam()* prototype for alignment with the  
57413 ISO/IEC 9899:1999 standard.

57414 The Open Group Corrigendum U047/1 is applied.

57415 IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a  
57416 call to the *pthread\_setschedprio()* function.

57417 **Issue 7**

57418 The *pthread\_getschedparam()* and *pthread\_setschedparam()* functions are marked only as part of the  
57419 Thread Execution Scheduling option as the Threads option is now part of the Base.

57420 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements  
57421 for the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values.

57422 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

57423 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0459 [314] is applied.

57424 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0276 [757] is applied.



57425 **NAME**

57426 pthread\_getspecific, pthread\_setspecific — thread-specific data management

57427 **SYNOPSIS**

57428 #include &lt;pthread.h&gt;

57429 void \*pthread\_getspecific(pthread\_key\_t key);

57430 int pthread\_setspecific(pthread\_key\_t key, const void \*value);

57431 **DESCRIPTION**57432 The *pthread\_getspecific()* function shall return the value currently bound to the specified *key* on  
57433 behalf of the calling thread.57434 The *pthread\_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a  
57435 previous call to *pthread\_key\_create()*. Different threads may bind different values to the same  
57436 key. These values are typically pointers to blocks of dynamically allocated memory that have  
57437 been reserved for use by the calling thread.57438 The effect of calling *pthread\_getspecific()* or *pthread\_setspecific()* with a *key* value not obtained  
57439 from *pthread\_key\_create()* or after *key* has been deleted with *pthread\_key\_delete()* is undefined.57440 Both *pthread\_getspecific()* and *pthread\_setspecific()* may be called from a thread-specific data  
57441 destructor function. A call to *pthread\_getspecific()* for the thread-specific data key being  
57442 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)  
57443 by a call to *pthread\_setspecific()*. Calling *pthread\_setspecific()* from a thread-specific data  
57444 destructor routine may result either in lost storage (after at least  
57445 PTHREAD\_DESTRUCTOR\_ITERATIONS attempts at destruction) or in an infinite loop.

57446 Both functions may be implemented as macros.

57447 **RETURN VALUE**57448 The *pthread\_getspecific()* function shall return the thread-specific data value associated with the  
57449 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be  
57450 returned.57451 If successful, the *pthread\_setspecific()* function shall return zero; otherwise, an error number shall  
57452 be returned to indicate the error.57453 **ERRORS**57454 No errors are returned from *pthread\_getspecific()*.57455 The *pthread\_setspecific()* function shall fail if:

57456 [ENOMEM] Insufficient memory exists to associate the non-NULL value with the key.

57457 The *pthread\_setspecific()* function shall not return an error code of [EINTR].57458 **EXAMPLES**

57459 None.

57460 **APPLICATION USAGE**

57461 None.

57462 **RATIONALE**57463 Performance and ease-of-use of *pthread\_getspecific()* are critical for functions that rely on  
57464 maintaining state in thread-specific data. Since no errors are required to be detected by it, and  
57465 since the only error that could be detected is the use of an invalid key, the function to  
57466 *pthread\_getspecific()* has been designed to favor speed and simplicity over error reporting.57467 If an implementation detects that the value specified by the *key* argument to *pthread\_setspecific()*  
57468 does not refer to a key value obtained from *pthread\_key\_create()* or refers to a key that has been

57469 deleted with *pthread\_key\_delete()*, it is recommended that the function should fail and report an  
57470 [EINVAL] error.

#### 57471 FUTURE DIRECTIONS

57472 None.

#### 57473 SEE ALSO

57474 *pthread\_key\_create()*

57475 XBD <pthread.h>

#### 57476 CHANGE HISTORY

57477 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 57478 Issue 6

57479 The *pthread\_getspecific()* and *pthread\_setspecific()* functions are marked as part of the Threads  
57480 option.

57481 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

57482 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/96 is applied, updating the ERRORS  
57483 section so that the [ENOMEM] error case is changed from ``to associate the value with the key''  
57484 to ``to associate the non-NULL value with the key''.

#### 57485 Issue 7

57486 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the ERRORS section.

57487 The *pthread\_getspecific()* and *pthread\_setspecific()* functions are moved from the Threads option to  
57488 the Base.

57489 The [EINVAL] error for a key value not obtained from *pthread\_key\_create()* or a key deleted with  
57490 *pthread\_key\_delete()* is removed; this condition results in undefined behavior.

57491 **NAME**

57492 pthread\_join — wait for thread termination

57493 **SYNOPSIS**

57494 #include &lt;pthread.h&gt;

57495 int pthread\_join(pthread\_t thread, void \*\*value\_ptr);

57496 **DESCRIPTION**

57497 The *pthread\_join()* function shall suspend execution of the calling thread until the target *thread*  
 57498 terminates, unless the target *thread* has already terminated. On return from a successful  
 57499 *pthread\_join()* call with a non-NULL *value\_ptr* argument, the value passed to *pthread\_exit()* by  
 57500 the terminating thread shall be made available in the location referenced by *value\_ptr*. When a  
 57501 *pthread\_join()* returns successfully, the target thread has been terminated. The results of multiple  
 57502 simultaneous calls to *pthread\_join()* specifying the same target thread are undefined. If the  
 57503 thread calling *pthread\_join()* is canceled, then the target thread shall not be detached.

57504 It is unspecified whether a zombie thread counts against {PTHREAD\_THREADS\_MAX}.

57505 The behavior is undefined if the value specified by the *thread* argument to *pthread\_join()* does not  
 57506 refer to a joinable thread.

57507 The behavior is undefined if the value specified by the *thread* argument to *pthread\_join()* refers to  
 57508 the calling thread.

57509 If *thread* refers to a thread that was created using *thrd\_create()* and the thread terminates, or has  
 57510 already terminated, by returning from its start routine, the behavior of *pthread\_join()* is  
 57511 undefined. If *thread* refers to a thread that terminates, or has already terminated, by calling  
 57512 *thrd\_exit()*, the behavior of *pthread\_join()* is undefined.

57513 **RETURN VALUE**

57514 If successful, the *pthread\_join()* function shall return zero; otherwise, an error number shall be  
 57515 returned to indicate the error.

57516 **ERRORS**

57517 The *pthread\_join()* function may fail if:

57518 [EDEADLK] A deadlock was detected.

57519 The *pthread\_join()* function shall not return an error code of [EINTR].

57520 **EXAMPLES**

57521 An example of thread creation and deletion follows:

```
57522 typedef struct {
57523     int *ar;
57524     long n;
57525 } subarray;

57526 void *
57527 incer(void *arg)
57528 {
57529     long i;

57530     for (i = 0; i < ((subarray *)arg)->n; i++)
57531         ((subarray *)arg)->ar[i]++;
57532 }

57533 int main(void)
57534 {
```

```

57535         int            ar[1000000];
57536         pthread_t    th1, th2;
57537         subarray     sb1, sb2;

57538         sb1.ar = &ar[0];
57539         sb1.n  = 500000;
57540         (void) pthread_create(&th1, NULL, incer, &sb1);

57541         sb2.ar = &ar[500000];
57542         sb2.n  = 500000;
57543         (void) pthread_create(&th2, NULL, incer, &sb2);

57544         (void) pthread_join(th1, NULL);
57545         (void) pthread_join(th2, NULL);
57546         return 0;
57547     }

```

#### APPLICATION USAGE

None.

#### RATIONALE

The *pthread\_join()* function is a convenience that has proven useful in multi-threaded applications. It is true that a programmer could simulate this function if it were not provided by passing extra state as part of the argument to the *start\_routine()*. The terminating thread would set a flag to indicate termination and broadcast a condition that is part of that state; a joining thread would wait on that condition variable. While such a technique would allow a thread to wait on more complex conditions (for example, waiting for multiple threads to terminate), waiting on individual thread termination is considered widely useful. Also, including the *pthread\_join()* function in no way precludes a programmer from coding such complex waits. Thus, while not a primitive, including *pthread\_join()* in this volume of POSIX.1-2024 was considered valuable.

The *pthread\_join()* function provides a simple mechanism allowing an application to wait for a thread to terminate. After the thread terminates, the application may then choose to clean up resources that were used by the thread. For instance, after *pthread\_join()* returns, any application-provided stack storage could be reclaimed.

The *pthread\_join()* or *pthread\_detach()* function should eventually be called for every thread that is created with the *detachstate* attribute set to *PTHREAD\_CREATE\_JOINABLE* so that storage associated with the thread may be reclaimed.

The interaction between *pthread\_join()* and cancellation is well-defined for the following reasons:

- The *pthread\_join()* function, like all other non-async-cancel-safe functions, can only be called with deferred cancelability type.
- Cancellation cannot occur in the disabled cancelability state.

Thus, only the default cancelability state need be considered. As specified, either the *pthread\_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the application, since either a cancellation handler is run or *pthread\_join()* returns. There are no race conditions since *pthread\_join()* was called in the deferred cancelability state.

If an implementation detects that the value specified by the *thread* argument to *pthread\_join()* does not refer to a joinable thread, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the *thread* argument to *pthread\_join()*

57580 refers to the calling thread, it is recommended that the function should fail and report an  
57581 [EDEADLK] error.

57582 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
57583 that the function should fail and report an [ESRCH] error.

57584 The *pthread\_join()* function cannot be used to obtain the exit status of a thread that was created  
57585 using *thrd\_create()* and which terminates by returning from its start routine, or of a thread that  
57586 terminates by calling *thrd\_exit()*, because such threads have an **int** exit status, instead of the  
57587 **void \*** that *pthread\_join()* returns via its *value\_ptr* argument.

#### 57588 FUTURE DIRECTIONS

57589 None.

#### 57590 SEE ALSO

57591 *pthread\_create()*, *thrd\_create()*, *thrd\_exit()*, *wait()*

57592 XBD Section 4.15.2 (on page 104), [<pthread.h>](#)

#### 57593 CHANGE HISTORY

57594 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 57595 Issue 6

57596 The *pthread\_join()* function is marked as part of the Threads option.

57597 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/97 is applied, updating the ERRORS  
57598 section so that the [EINVAL] error is made optional and the words “the implementation has  
57599 detected” are removed from it.

#### 57600 Issue 7

57601 The *pthread\_join()* function is moved from the Threads option to the Base.

57602 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

57603 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined  
57604 behavior.

57605 The [EDEADLK] error for the calling thread is removed; this condition results in undefined  
57606 behavior.

#### 57607 Issue 8

57608 Austin Group Defect 792 is applied, changing “a thread that has exited but remains unjoined” to  
57609 “a zombie thread”.

57610 Austin Group Defect 1302 is applied, updating the page to account for the addition of  
57611 [<threads.h>](#) interfaces.

57612 **NAME**

57613 pthread\_key\_create — thread-specific data key creation

57614 **SYNOPSIS**

57615 #include &lt;pthread.h&gt;

57616 int pthread\_key\_create(pthread\_key\_t \*key, void (\*destructor)(void\*));

57617 **DESCRIPTION**

57618 The *pthread\_key\_create()* function shall create a thread-specific data key visible to all threads in  
57619 the process. Key values provided by *pthread\_key\_create()* are opaque objects used to locate  
57620 thread-specific data. Although the same key value may be used by different threads, the values  
57621 bound to the key by *pthread\_setspecific()* are maintained on a per-thread basis and persist for the  
57622 life of the calling thread.

57623 Upon key creation, the value NULL shall be associated with the new key in all active threads.  
57624 Upon thread creation, the value NULL shall be associated with all defined keys in the new  
57625 thread.

57626 An optional destructor function may be associated with each key value. At thread exit, if a key  
57627 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with  
57628 that key, the value of the key is set to NULL, and then the function pointed to is called with the  
57629 previously associated value as its sole argument. The order of destructor calls is unspecified if  
57630 more than one destructor exists for a thread when it exits.

57631 If, after all the destructors have been called for all non-NULL values with associated destructors,  
57632 there are still some non-NULL values with associated destructors, then the process is repeated.  
57633 If, after at least {PTHREAD\_DESTRUCTOR\_ITERATIONS} iterations of destructor calls for  
57634 outstanding non-NULL values, there are still some non-NULL values with associated  
57635 destructors, implementations may stop calling destructors, or they may continue calling  
57636 destructors until no non-NULL values with associated destructors exist, even though this might  
57637 result in an infinite loop.

57638 **RETURN VALUE**

57639 If successful, the *pthread\_key\_create()* function shall store the newly created key value at *\*key* and  
57640 shall return zero. Otherwise, an error number shall be returned to indicate the error.

57641 **ERRORS**

57642 The *pthread\_key\_create()* function shall fail if:

57643 [EAGAIN] The system lacked the necessary resources to create another thread-specific  
57644 data key, or the system-imposed limit on the total number of keys per process  
57645 {PTHREAD\_KEYS\_MAX} has been exceeded.

57646 [ENOMEM] Insufficient memory exists to create the key.

57647 The *pthread\_key\_create()* function shall not return an error code of [EINTR].

57648 **EXAMPLES**

57649 The following example demonstrates a function that initializes a thread-specific data key when it  
57650 is first called, and associates a thread-specific object with each calling thread, initializing this  
57651 object when necessary.

```
57652 static pthread_key_t key;  
57653 static pthread_once_t key_once = PTHREAD_ONCE_INIT;  
  
57654 static void  
57655 make_key()  
57656 {
```

```

57657         (void) pthread_key_create (&key, NULL);
57658     }
57659     func ()
57660     {
57661         void *ptr;
57662
57663         (void) pthread_once (&key_once, make_key);
57664         if ((ptr = pthread_getspecific(key)) == NULL) {
57665             ptr = malloc(OBJECT_SIZE);
57666             ...
57667             (void) pthread_setspecific(key, ptr);
57668         }
57669     }

```

57670 Note that the key has to be initialized before *pthread\_getspecific()* or *pthread\_setspecific()* can be  
57671 used. The *pthread\_key\_create()* call could either be explicitly made in a module initialization  
57672 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt  
57673 to use the key before it is initialized is a programming error, making the code below incorrect.

```

57674     static pthread_key_t key;
57675
57676     func ()
57677     {
57678         void *ptr;
57679
57680         /* KEY NOT INITIALIZED!!! THIS WILL NOT WORK!!! */
57681         if ((ptr = pthread_getspecific(key)) == NULL &&
57682             pthread_setspecific(key, NULL) != 0) {
57683             pthread_key_create (&key, NULL);
57684             ...
57685         }
57686     }

```

#### 57685 APPLICATION USAGE

57686 None.

#### 57687 RATIONALE

##### 57688 Destructor Functions

57689 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage  
57690 allocated dynamically on behalf of the calling thread. The destructor functions specified with  
57691 *pthread\_key\_create()* are intended to be used to free this storage when the thread exits. Thread  
57692 cancellation cleanup handlers cannot be used for this purpose because thread-specific data may  
57693 persist outside the lexical scope in which the cancellation cleanup handlers operate.

57694 If the value associated with a key needs to be updated during the lifetime of the thread, it may  
57695 be necessary to release the storage associated with the old value before the new value is bound.  
57696 Although the *pthread\_setspecific()* function could do this automatically, this feature is not needed  
57697 often enough to justify the added complexity. Instead, the programmer is responsible for freeing  
57698 the stale storage:

```

57699     old = pthread_getspecific(key);
57700     new = allocate();
57701     destructor(old);

```

57702 pthread\_setspecific(key, new);

57703 **Note:** The above example could leak storage if run with asynchronous cancellation enabled. No such  
57704 problems occur in the default cancellation state if no cancellation points occur between the get  
57705 and set.

57706 There is no notion of a destructor-safe function. If an application does not call *pthread\_exit()*  
57707 from a signal handler, or if it blocks any signal whose handler may call *pthread\_exit()* while  
57708 calling async-unsafe functions, all functions may be safely called from destructors.

#### 57709 Non-Idempotent Data Key Creation

57710 There were requests to make *pthread\_key\_create()* idempotent with respect to a given *key* address  
57711 parameter. This would allow applications to call *pthread\_key\_create()* multiple times for a given  
57712 *key* address and be guaranteed that only one key would be created. Doing so would require the  
57713 key value to be previously initialized (possibly at compile time) to a known null value and  
57714 would require that implicit mutual-exclusion be performed based on the address and contents of  
57715 the *key* parameter in order to guarantee that exactly one key would be created.

57716 Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread\_key\_create()*.  
57717 On many implementations, implicit mutual-exclusion would also have to be performed by  
57718 *pthread\_getspecific()* and *pthread\_setspecific()* in order to guard against using incompletely stored  
57719 or not-yet-visible key values. This could significantly increase the cost of important operations,  
57720 particularly *pthread\_getspecific()*.

57721 Thus, this proposal was rejected. The *pthread\_key\_create()* function performs no implicit  
57722 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once  
57723 per key before use of the key. Several straightforward mechanisms can already be used to  
57724 accomplish this, including calling explicit module initialization functions, using mutexes, and  
57725 using *pthread\_once()*. This places no significant burden on the programmer, introduces no  
57726 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows  
57727 commonly used thread-specific data operations to be more efficient.

#### 57728 FUTURE DIRECTIONS

57729 None.

#### 57730 SEE ALSO

57731 [pthread\\_getspecific\(\)](#), [pthread\\_key\\_delete\(\)](#)

57732 XBD <[pthread.h](#)>

#### 57733 CHANGE HISTORY

57734 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 57735 Issue 6

57736 The *pthread\_key\_create()* function is marked as part of the Threads option.

57737 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

#### 57738 Issue 7

57739 The *pthread\_key\_create()* function is moved from the Threads option to the Base.

#### 57740 Issue 8

57741 Austin Group Defect 1059 is applied, changing the example code in the RATIONALE section.



57742 **NAME**

57743 pthread\_key\_delete — thread-specific data key deletion

57744 **SYNOPSIS**

57745 #include &lt;pthread.h&gt;

57746 int pthread\_key\_delete(pthread\_key\_t key);

57747 **DESCRIPTION**

57748 The *pthread\_key\_delete()* function shall delete a thread-specific data key previously returned by  
57749 *pthread\_key\_create()*. The thread-specific data values associated with *key* need not be NULL at  
57750 the time *pthread\_key\_delete()* is called. It is the responsibility of the application to free any  
57751 application storage or perform any cleanup actions for data structures related to the deleted key  
57752 or associated thread-specific data in any threads; this cleanup can be done either before or after  
57753 *pthread\_key\_delete()* is called. Any attempt to use *key* following the call to *pthread\_key\_delete()*  
57754 results in undefined behavior.

57755 The *pthread\_key\_delete()* function shall be callable from within destructor functions. No  
57756 destructor functions shall be invoked by *pthread\_key\_delete()*. Any destructor function that may  
57757 have been associated with *key* shall no longer be called upon thread exit.

57758 **RETURN VALUE**

57759 If successful, the *pthread\_key\_delete()* function shall return zero; otherwise, an error number shall  
57760 be returned to indicate the error.

57761 **ERRORS**

57762 The *pthread\_key\_delete()* function shall not return an error code of [EINTR].

57763 **EXAMPLES**

57764 None.

57765 **APPLICATION USAGE**

57766 None.

57767 **RATIONALE**

57768 A thread-specific data key deletion function has been included in order to allow the resources  
57769 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys  
57770 can arise, among other scenarios, when a dynamically loaded module that allocated a key is  
57771 unloaded.

57772 Conforming applications are responsible for performing any cleanup actions needed for data  
57773 structures associated with the key to be deleted, including data referenced by thread-specific  
57774 data values. No such cleanup is done by *pthread\_key\_delete()*. In particular, destructor functions  
57775 are not called. There are several reasons for this division of responsibility:

- 57776 1. The associated destructor functions used to free thread-specific data at thread exit time  
57777 are only guaranteed to work correctly when called in the thread that allocated the thread-  
57778 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot  
57779 be used to free thread-specific data in other threads at key deletion time. Attempting to  
57780 have them called by other threads at key deletion time would require other threads to be  
57781 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,  
57782 including holding locks necessary for the destructor to run, this approach would fail. In  
57783 general, there is no safe mechanism whereby an implementation could free thread-  
57784 specific data at key deletion time.
- 57785 2. Even if there were a means of safely freeing thread-specific data associated with keys to  
57786 be deleted, doing so would require that implementations be able to enumerate the  
57787 threads with non-NULL data and potentially keep them from creating more thread-

57788 specific data while the key deletion is occurring. This special case could cause extra  
57789 synchronization in the normal case, which would otherwise be unnecessary.

57790 For an application to know that it is safe to delete a key, it has to know that all the threads that  
57791 might potentially ever use the key do not attempt to use it again. For example, it could know  
57792 this if all the client threads have called a cleanup procedure declaring that they are through with  
57793 the module that is being shut down, perhaps by setting a reference count to zero.

57794 If an implementation detects that the value specified by the *key* argument to *pthread\_key\_delete()*  
57795 does not refer to a key value obtained from *pthread\_key\_create()* or refers to a key that has been  
57796 deleted with *pthread\_key\_delete()*, it is recommended that the function should fail and report an  
57797 [EINVAL] error.

#### 57798 FUTURE DIRECTIONS

57799 None.

#### 57800 SEE ALSO

57801 [\*pthread\\_key\\_create\(\)\*](#)

57802 XBD <[pthread.h](#)>

#### 57803 CHANGE HISTORY

57804 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 57805 Issue 6

57806 The *pthread\_key\_delete()* function is marked as part of the Threads option.

#### 57807 Issue 7

57808 The *pthread\_key\_delete()* function is moved from the Threads option to the Base.

57809 The [EINVAL] error for a key value not obtained from *pthread\_key\_create()* or a key deleted with  
57810 *pthread\_key\_delete()* is removed; this condition results in undefined behavior.

57811 **NAME**

57812 pthread\_kill — send a signal to a thread

57813 **SYNOPSIS**

```
57814 CX #include <signal.h>
57815 int pthread_kill(pthread_t thread, int sig);
```

57816 **DESCRIPTION**

57817 The *pthread\_kill()* function shall request that a signal be delivered to the specified thread. It shall  
57818 not be an error if *thread* is a zombie thread.

57819 **RETURN VALUE**

57820 Upon successful completion, the function shall return a value of zero. Otherwise, the function  
57821 shall return an error number. If the *pthread\_kill()* function fails, no signal shall be sent.

57822 **ERRORS**

57823 The *pthread\_kill()* function may fail if:

57824 [EINVAL] The value of the *sig* argument is zero.

57825 The *pthread\_kill()* function shall fail if:

57826 [EINVAL] The value of the *sig* argument is non-zero and is an invalid or unsupported  
57827 signal number.

57828 The *pthread\_kill()* function shall not return an error code of [EINTR].

57829 **EXAMPLES**

57830 None.

57831 **APPLICATION USAGE**

57832 The *pthread\_kill()* function provides a mechanism for asynchronously directing a signal at a  
57833 thread in the calling process. This could be used, for example, by one thread to affect broadcast  
57834 delivery of a signal to a set of threads.

57835 Note that *pthread\_kill()* only causes the signal to be handled in the context of the given thread;  
57836 the signal action (termination or stopping) affects the process as a whole.

57837 **RATIONALE**

57838 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
57839 that the function should fail and report an [ESRCH] error.

57840 Historical implementations varied on the result of a *pthread\_kill()* with a thread ID indicating a  
57841 zombie thread. Some indicated success on such a call, while others gave an error of [ESRCH].  
57842 Since the definition of thread lifetime in this volume of POSIX.1-2024 covers zombie threads, the  
57843 [ESRCH] error as described is inappropriate in this case and implementations that give this error  
57844 do not conform. In particular, this means that an application cannot have one thread check for  
57845 termination of another by calling *pthread\_kill()* with a *sig* argument of zero, and implementations  
57846 may indicate that it is not possible by returning [EINVAL] when *sig* is zero.

57847 **FUTURE DIRECTIONS**

57848 None.

57849 **SEE ALSO**

57850 *kill()*, *pthread\_self()*, *raise()*

57851 XBD <signal.h>

## 57852 CHANGE HISTORY

57853 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

### 57854 Issue 6

57855 The *pthread\_kill()* function is marked as part of the Threads option.

57856 The APPLICATION USAGE section is added.

### 57857 Issue 7

57858 The *pthread\_kill()* function is moved from the Threads option to the Base.

57859 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

57860 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0277 [765] is applied.

### 57861 Issue 8

57862 Austin Group Defect 792 is applied, adding a requirement that passing the thread ID of a zombie thread to *pthread\_kill()* is not treated as an error.

57864 Austin Group Defect 1214 is applied, allowing *pthread\_kill()* to fail with [EINVAL] when the *sig* argument is zero.

57866 **NAME**

57867 pthread\_mutex\_clocklock, pthread\_mutex\_timedlock — lock a mutex

57868 **SYNOPSIS**

```
57869 #include <pthread.h>
57870 int pthread_mutex_clocklock(pthread_mutex_t *restrict mutex,
57871     clockid_t clock_id, const struct timespec *restrict abstime);
57872 int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
57873     const struct timespec *restrict abstime);
```

57874 **DESCRIPTION**

57875 The *pthread\_mutex\_clocklock()* and *pthread\_mutex\_timedlock()* functions shall lock the mutex  
 57876 object referenced by *mutex*. If the mutex is already locked, the calling thread shall block until the  
 57877 mutex becomes available as in the *pthread\_mutex\_lock()* function. If the mutex cannot be locked  
 57878 without waiting for another thread to unlock the mutex, this wait shall be terminated when the  
 57879 specified timeout expires.

57880 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the  
 57881 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 57882 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the  
 57883 call.

57884 For *pthread\_mutex\_timedlock()*, the timeout shall be based on the CLOCK\_REALTIME clock. For  
 57885 *pthread\_mutex\_clocklock()*, the timeout shall be based on the clock specified by the *clock\_id*  
 57886 argument. The resolution of the timeout shall be the resolution of the clock on which it is based.  
 57887 Implementations shall support passing CLOCK\_REALTIME and CLOCK\_MONOTONIC to  
 57888 *pthread\_mutex\_clocklock()* as the *clock\_id* argument.

57889 Under no circumstance shall the function fail with a timeout if the mutex can be locked  
 57890 immediately. The validity of the *abstime* parameter need not be checked if the mutex can be  
 57891 locked immediately.

57892 RPI|TPI As a consequence of the priority inheritance rules (for mutexes initialized with the  
 57893 PRIO\_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the  
 57894 priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this  
 57895 thread is no longer among the threads waiting for the mutex.

57896 If *mutex* is a robust mutex and the process containing the owning thread terminated while  
 57897 holding the mutex lock, a call to *pthread\_mutex\_clocklock()* or *pthread\_mutex\_timedlock()* shall  
 57898 return the error value [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread  
 57899 terminated while holding the mutex lock, a call to *pthread\_mutex\_clocklock()* or  
 57900 *pthread\_mutex\_timedlock()* may return the error value [EOWNERDEAD] even if the process in  
 57901 which the owning thread resides has not terminated. In these cases, the mutex is locked by the  
 57902 thread but the state it protects is marked as inconsistent. The application should ensure that the  
 57903 state is made consistent for reuse and when that is complete call *pthread\_mutex\_consistent()*. If  
 57904 the application is unable to recover the state, it should unlock the mutex without a prior call to  
 57905 *pthread\_mutex\_consistent()*, after which the mutex is marked permanently unusable.

57906 If *mutex* does not refer to an initialized mutex object, the behavior is undefined.

57907 **RETURN VALUE**

57908 If successful, the *pthread\_mutex\_clocklock()* and *pthread\_mutex\_timedlock()* functions shall return  
 57909 zero; otherwise, an error number shall be returned to indicate the error.

57910 **ERRORS**

- 57911 The *pthread\_mutex\_clocklock()* and *pthread\_mutex\_timedlock()* functions shall fail if:
- 57912 [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
57913 locks for *mutex* has been exceeded.
- 57914 [EAGAIN] The mutex is a robust mutex and the system resources available for robust  
57915 mutexes owned would be exceeded.
- 57916 [EDEADLK] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK and the current  
57917 thread already owns the mutex.
- 57918 [EINVAL] The mutex was created with the protocol attribute having the value  
57919 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
57920 the mutex' current priority ceiling.
- 57921 [EINVAL] The process or thread would have blocked, and either the *abstime* parameter  
57922 specified a nanoseconds field value less than zero or greater than or equal to  
57923 1 000 million, or the *pthread\_mutex\_clocklock()* function was passed an invalid  
57924 or unsupported *clock\_id* value.
- 57925 [ENOTRECOVERABLE]  
57926 The state protected by the mutex is not recoverable.
- 57927 [EOWNERDEAD]  
57928 The mutex is a robust mutex and the process containing the previous owning  
57929 thread terminated while holding the mutex lock. The mutex lock shall be  
57930 acquired by the calling thread and it is up to the new owner to make the state  
57931 consistent.
- 57932 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.  
57933 The *pthread\_mutex\_clocklock()* and *pthread\_mutex\_timedlock()* functions may fail if:
- 57934 [EDEADLK] A deadlock condition was detected.
- 57935 [EOWNERDEAD]  
57936 The mutex is a robust mutex and the previous owning thread terminated  
57937 while holding the mutex lock. The mutex lock shall be acquired by the calling  
57938 thread and it is up to the new owner to make the state consistent.
- 57939 These functions shall not return an error code of [EINTR].

57940 **EXAMPLES**

57941 None.

57942 **APPLICATION USAGE**

57943 Applications that have assumed that non-zero return values are errors will need updating for  
57944 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting  
57945 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error  
57946 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If  
57947 an application is supposed to work with normal and robust mutexes, it should check all return  
57948 values for error conditions and if necessary take appropriate action.

57949 **RATIONALE**57950 Refer to *pthread\_mutex\_lock()*.

57951 **FUTURE DIRECTIONS**

57952 None.

57953 **SEE ALSO**57954 *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*, *time()*57955 XBD Section 4.15.2 (on page 104), **<pthread.h>**, **<time.h>**57956 **CHANGE HISTORY**

57957 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

57958 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/99 is applied, marking the last paragraph  
57959 in the DESCRIPTION as part of the Thread Priority Inheritance option.57960 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/100 is applied, updating the ERRORS  
57961 section so that the [EDEADLK] error includes detection of a deadlock condition.57962 **Issue 7**

57963 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

57964 The *pthread\_mutex\_timedlock()* function is moved from the Timeouts option to the Base.

57965 Functionality relating to the Timers option is moved to the Base.

57966 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized  
57967 mutex.

57968 The ERRORS section is updated to account properly for all of the various mutex types.

57969 **Issue 8**57970 Austin Group Defect 354 is applied, adding the [EAGAIN] error for exceeding system resources  
57971 available for robust mutexes owned.57972 Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and  
57973 DESCRIPTION sections.57974 Austin Group Defects 1216 and 1472 are applied, adding *pthread\_mutex\_clocklock()*.

57975 **NAME**

57976 pthread\_mutex\_consistent — mark state protected by robust mutex as consistent

57977 **SYNOPSIS**

57978 #include &lt;pthread.h&gt;

57979 int pthread\_mutex\_consistent(pthread\_mutex\_t \*mutex);

57980 **DESCRIPTION**57981 If *mutex* is a robust mutex in an inconsistent state, the *pthread\_mutex\_consistent()* function can be  
57982 used to mark the state protected by the mutex referenced by *mutex* as consistent again.57983 If an owner of a robust mutex terminates while holding the mutex, the mutex becomes  
57984 inconsistent and the next thread that acquires the mutex lock shall be notified of the state by the  
57985 return value [EOWNERDEAD]. In this case, the mutex does not become normally usable again  
57986 until the state is marked consistent.57987 If the thread which acquired the mutex lock with the return value [EOWNERDEAD] terminates  
57988 before calling either *pthread\_mutex\_consistent()* or *pthread\_mutex\_unlock()*, the next thread that  
57989 acquires the mutex lock shall be notified about the state of the mutex by the return value  
57990 [EOWNERDEAD].57991 The behavior is undefined if the value specified by the *mutex* argument to  
57992 *pthread\_mutex\_consistent()* does not refer to an initialized mutex.57993 **RETURN VALUE**57994 Upon successful completion, the *pthread\_mutex\_consistent()* function shall return zero.  
57995 Otherwise, an error value shall be returned to indicate the error.57996 **ERRORS**57997 The *pthread\_mutex\_consistent()* function shall fail if:57998 [EINVAL] The mutex object referenced by *mutex* is not robust or does not protect an  
57999 inconsistent state.

58000 These functions shall not return an error code of [EINTR].

58001 **EXAMPLES**

58002 None.

58003 **APPLICATION USAGE**58004 The *pthread\_mutex\_consistent()* function is only responsible for notifying the implementation that  
58005 the state protected by the mutex has been recovered and that normal operations with the mutex  
58006 can be resumed. It is the responsibility of the application to recover the state so it can be reused.  
58007 If the application is not able to perform the recovery, it can notify the implementation that the  
58008 situation is unrecoverable by a call to *pthread\_mutex\_unlock()* without a prior call to  
58009 *pthread\_mutex\_consistent()*, in which case subsequent threads that attempt to lock the mutex will  
58010 fail to acquire the lock and be returned [ENOTRECOVERABLE].58011 **RATIONALE**58012 If an implementation detects that the value specified by the *mutex* argument to  
58013 *pthread\_mutex\_consistent()* does not refer to an initialized mutex, it is recommended that the  
58014 function should fail and report an [EINVAL] error.58015 **FUTURE DIRECTIONS**

58016 None.



58017 **SEE ALSO**

58018 [pthread\\_mutex\\_lock\(\)](#), [pthread\\_mutexattr\\_getrobust\(\)](#)

58019 XBD [<pthread.h>](#)

58020 **CHANGE HISTORY**

58021 First released in Issue 7.

58022 **NAME**

58023 pthread\_mutex\_destroy, pthread\_mutex\_init — destroy and initialize a mutex

58024 **SYNOPSIS**

```
58025 #include <pthread.h>
58026 int pthread_mutex_destroy(pthread_mutex_t *mutex);
58027 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
58028     const pthread_mutexattr_t *restrict attr);
58029 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

58030 **DESCRIPTION**

58031 The *pthread\_mutex\_destroy()* function shall destroy the mutex object referenced by *mutex*; the  
58032 mutex object becomes, in effect, uninitialized. An implementation may cause  
58033 *pthread\_mutex\_destroy()* to set the object referenced by *mutex* to an invalid value.

58034 A destroyed mutex object can be reinitialized using *pthread\_mutex\_init()*; the results of otherwise  
58035 referencing the object after it has been destroyed are undefined.

58036 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked  
58037 mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a  
58038 *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* call by another thread,  
58039 results in undefined behavior.

58040 The *pthread\_mutex\_init()* function shall initialize the mutex referenced by *mutex* with attributes  
58041 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the  
58042 same as passing the address of a default mutex attributes object. Upon successful initialization,  
58043 the state of the mutex becomes initialized and unlocked.

58044 See [Section 2.9.9](#) (on page 548) for further requirements.

58045 Attempting to initialize an already initialized mutex results in undefined behavior.

58046 In cases where default mutex attributes are appropriate, the macro  
58047 PTHREAD\_MUTEX\_INITIALIZER can be used to initialize mutexes. The effect shall be  
58048 equivalent to dynamic initialization by a call to *pthread\_mutex\_init()* with parameter *attr*  
58049 specified as NULL, except that no error checks are performed.

58050 The behavior is undefined if the value specified by the *mutex* argument to  
58051 *pthread\_mutex\_destroy()* does not refer to an initialized mutex.

58052 The behavior is undefined if the value specified by the *attr* argument to *pthread\_mutex\_init()*  
58053 does not refer to an initialized mutex attributes object.

58054 **RETURN VALUE**

58055 If successful, the *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions shall return zero;  
58056 otherwise, an error number shall be returned to indicate the error.

58057 **ERRORS**

58058 The *pthread\_mutex\_init()* function shall fail if:

58059 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
58060 another mutex.

58061 [ENOMEM] Insufficient memory exists to initialize the mutex.

58062 [EPERM] The caller does not have the privilege to perform the operation.

58063 The *pthread\_mutex\_init()* function may fail if:  
 58064 [EINVAL] The attributes object referenced by *attr* has the robust mutex attribute set  
 58065 without the process-shared attribute being set.  
 58066 These functions shall not return an error code of [EINTR].

**EXAMPLES**

58067 None.  
 58068

**APPLICATION USAGE**

58069 None.  
 58070

**RATIONALE**

58071 If an implementation detects that the value specified by the *mutex* argument to  
 58072 *pthread\_mutex\_destroy()* does not refer to an initialized mutex, it is recommended that the  
 58073 function should fail and report an [EINVAL] error.  
 58074

58075 If an implementation detects that the value specified by the *mutex* argument to  
 58076 *pthread\_mutex\_destroy()* or *pthread\_mutex\_init()* refers to a locked mutex or a mutex that is  
 58077 referenced (for example, while being used in a *pthread\_cond\_clockwait()*,  
 58078 *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* call) by another thread, or detects that the value  
 58079 specified by the *mutex* argument to *pthread\_mutex\_init()* refers to an already initialized mutex, it  
 58080 is recommended that the function should fail and report an [EBUSY] error.

58081 If an implementation detects that the value specified by the *attr* argument to  
 58082 *pthread\_mutex\_init()* does not refer to an initialized mutex attributes object, it is recommended  
 58083 that the function should fail and report an [EINVAL] error.

**Alternate Implementations Possible**

58084  
 58085 This volume of POSIX.1-2024 supports several alternative implementations of mutexes. An  
 58086 implementation may store the lock directly in the object of type **pthread\_mutex\_t**. Alternatively,  
 58087 an implementation may store the lock in the heap and merely store a pointer, handle, or unique  
 58088 ID in the mutex object. Either implementation has advantages or may be required on certain  
 58089 hardware configurations. So that portable code can be written that is invariant to this choice, this  
 58090 volume of POSIX.1-2024 does not define assignment or equality for this type, and it uses the  
 58091 term “initialize” to reinforce the (more restrictive) notion that the lock may actually reside in the  
 58092 mutex object itself.

58093 Note that this precludes an over-specification of the type of the mutex or condition variable and  
 58094 motivates the opaqueness of the type.

58095 An implementation is permitted, but not required, to have *pthread\_mutex\_destroy()* store an  
 58096 illegal value into the mutex. This may help detect erroneous programs that try to lock (or  
 58097 otherwise reference) a mutex that has already been destroyed.

**Tradeoff Between Error Checks and Performance Supported**

58098  
 58099 Many error conditions that can occur are not required to be detected by the implementation in  
 58100 order to let implementations trade off performance *versus* degree of error checking according to  
 58101 the needs of their specific applications and execution environment. As a general rule, conditions  
 58102 caused by the system (such as insufficient memory) are required to be detected, but conditions  
 58103 caused by an erroneously coded application (such as failing to provide adequate  
 58104 synchronization to prevent a mutex from being deleted while in use) are specified to result in  
 58105 undefined behavior.

58106 A wide range of implementations is thus made possible. For example, an implementation

58107 intended for application debugging may implement all of the error checks, but an  
58108 implementation running a single, provably correct application under very tight performance  
58109 constraints in an embedded computer might implement minimal checks. An implementation  
58110 might even be provided in two versions, similar to the options that compilers provide: a full-  
58111 checking, but slower version; and a limited-checking, but faster version. To forbid this  
58112 optionality would be a disservice to users.

58113 By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly  
58114 coded) application might do, and by defining that resource-not-available errors are mandatory,  
58115 this volume of POSIX.1-2024 ensures that a fully-conforming application is portable across the  
58116 full range of implementations, while not forcing all implementations to add overhead to check  
58117 for numerous things that a correct program never does. When the behavior is undefined, no  
58118 error number is specified to be returned on implementations that do detect the condition. This is  
58119 because undefined behavior means *anything* can happen, which includes returning with any  
58120 value (which might happen to be a valid, but different, error number). However, since the error  
58121 number might be useful to application developers when diagnosing problems during  
58122 application development, a recommendation is made in rationale that implementors should  
58123 return a particular error number if their implementation does detect the condition.

#### 58124 **Why No Limits are Defined**

58125 Defining symbols for the maximum number of mutexes and condition variables was considered  
58126 but rejected because the number of these objects may change dynamically. Furthermore, many  
58127 implementations place these objects into application memory; thus, there is no explicit  
58128 maximum.

#### 58129 **Static Initializers for Mutexes and Condition Variables**

58130 Providing for static initialization of statically allocated synchronization objects allows modules  
58131 with private static synchronization variables to avoid runtime initialization tests and overhead.  
58132 Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in  
58133 C libraries, where for various reasons the design calls for self-initialization instead of requiring  
58134 an explicit module initialization function to be called. An example use of static initialization  
58135 follows.

58136 Without static initialization, a self-initializing routine *foo()* might look as follows:

```
58137 static pthread_once_t foo_once = PTHREAD_ONCE_INIT;  
58138 static pthread_mutex_t foo_mutex;  
  
58139 void foo_init()  
58140 {  
58141     pthread_mutex_init(&foo_mutex, NULL);  
58142 }  
  
58143 void foo()  
58144 {  
58145     pthread_once(&foo_once, foo_init);  
58146     pthread_mutex_lock(&foo_mutex);  
58147     /* Do work. */  
58148     pthread_mutex_unlock(&foo_mutex);  
58149 }
```

58150 With static initialization, the same routine could be coded as follows:

```
58151 static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```

58152     void foo()
58153     {
58154         pthread_mutex_lock(&foo_mutex);
58155         /* Do work. */
58156         pthread_mutex_unlock(&foo_mutex);
58157     }

```

58158 Note that the static initialization both eliminates the need for the initialization test inside  
58159 *pthread\_once()* and the fetch of *&foo\_mutex* to learn the address to be passed to  
58160 *pthread\_mutex\_lock()* or *pthread\_mutex\_unlock()*.

58161 Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a  
58162 large class of systems; those where the (entire) synchronization object can be stored in  
58163 application memory.

58164 Yet the locking performance question is likely to be raised for machines that require mutexes to  
58165 be allocated out of special memory. Such machines actually have to have mutexes and possibly  
58166 condition variables contain pointers to the actual hardware locks. For static initialization to work  
58167 on such machines, *pthread\_mutex\_lock()* also has to test whether or not the pointer to the actual  
58168 lock has been allocated. If it has not, *pthread\_mutex\_lock()* has to initialize it before use. The  
58169 reservation of such resources can be made when the program is loaded, and hence return codes  
58170 have not been added to mutex locking and condition variable waiting to indicate failure to  
58171 complete initialization.

58172 This runtime test in *pthread\_mutex\_lock()* would at first seem to be extra work; an extra test is  
58173 required to see whether the pointer has been initialized. On most machines this would actually  
58174 be implemented as a fetch of the pointer, testing the pointer against zero, and then using the  
58175 pointer if it has already been initialized. While the test might seem to add extra work, the extra  
58176 effort of testing a register is usually negligible since no extra memory references are actually  
58177 done. As more and more machines provide caches, the real expenses are memory references, not  
58178 instructions executed.

58179 Alternatively, depending on the machine architecture, there are often ways to eliminate *all*  
58180 overhead in the most important case: on the lock operations that occur *after* the lock has been  
58181 initialized. This can be done by shifting more overhead to the less frequent operation:  
58182 initialization. Since out-of-line mutex allocation also means that an address has to be  
58183 dereferenced to find the actual lock, one technique that is widely applicable is to have static  
58184 initialization store a bogus value for that address; in particular, an address that causes a machine  
58185 fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity  
58186 checks can be done, and then the correct address for the actual lock can be filled in. Subsequent  
58187 lock operations incur no extra overhead since they do not “fault”. This is merely one technique  
58188 that can be used to support static initialization, while not adversely affecting the performance of  
58189 lock acquisition. No doubt there are other techniques that are highly machine-dependent.

58190 The locking overhead for machines doing out-of-line mutex allocation is thus similar for  
58191 modules being implicitly initialized, where it is improved for those doing mutex allocation  
58192 entirely inline. The inline case is thus made much faster, and the out-of-line case is not  
58193 significantly worse.

58194 Besides the issue of locking performance for such machines, a concern is raised that it is possible  
58195 that threads would serialize contending for initialization locks when attempting to finish  
58196 initializing statically allocated mutexes. (Such finishing would typically involve taking an  
58197 internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing  
58198 the internal lock.) First, many implementations would reduce such serialization by hashing on  
58199 the mutex address. Second, such serialization can only occur a bounded number of times. In

58200 particular, it can happen at most as many times as there are statically allocated synchronization  
 58201 objects. Dynamically allocated objects would still be initialized via *pthread\_mutex\_init()* or  
 58202 *pthread\_cond\_init()*.

58203 Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient  
 58204 performance for an application on some implementation, the application can avoid static  
 58205 initialization altogether by explicitly initializing all synchronization objects with the  
 58206 corresponding *pthread\_\*\_init()* functions, which are supported by all implementations. An  
 58207 implementation can also document the tradeoffs and advise which initialization technique is  
 58208 more efficient for that particular implementation.

### 58209 Destroying Mutexes

58210 A mutex can be destroyed immediately after it is unlocked. However, since attempting to  
 58211 destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is  
 58212 being used in a *pthread\_cond\_clockwait()*, *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* call by  
 58213 another thread, results in undefined behavior, care must be taken to ensure that no other thread  
 58214 may be referencing the mutex.

### 58215 Robust Mutexes

58216 Implementations are required to provide robust mutexes for mutexes with the process-shared  
 58217 attribute set to PTHREAD\_PROCESS\_SHARED. Implementations are allowed, but not required,  
 58218 to provide robust mutexes when the process-shared attribute is set to  
 58219 PTHREAD\_PROCESS\_PRIVATE.

### 58220 FUTURE DIRECTIONS

58221 None.

### 58222 SEE ALSO

58223 *pthread\_mutex\_getprioceiling()*, *pthread\_mutexattr\_getrobust()*, *pthread\_mutex\_clocklock()*,  
 58224 *pthread\_mutex\_lock()*, *pthread\_mutexattr\_getpshared()*

58225 XBD <pthread.h>

### 58226 CHANGE HISTORY

58227 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

### 58228 Issue 6

58229 The *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions are marked as part of the  
 58230 Threads option.

58231 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
 58232 IEEE Std 1003.1d-1999.

58233 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

58234 The **restrict** keyword is added to the *pthread\_mutex\_init()* prototype for alignment with the  
 58235 ISO/IEC 9899:1999 standard.

### 58236 Issue 7

58237 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

58238 The *pthread\_mutex\_destroy()* and *pthread\_mutex\_init()* functions are moved from the Threads  
 58239 option to the Base.

58240 The [EINVAL] error for an uninitialized mutex or an uninitialized mutex attributes object is  
 58241 removed; this condition results in undefined behavior.

- 58242 The [EBUSY] error for a locked mutex, a mutex that is referenced, or an already initialized mutex  
58243 is removed; this condition results in undefined behavior.
- 58244 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0460 [70,428] is applied.
- 58245 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0278 [811], XSH/TC2-2008/0279 [972],  
58246 and XSH/TC2-2008/0280 [811] are applied.
- 58247 **Issue 8**  
58248 Austin Group Defect 1216 is applied, adding *pthread\_cond\_clockwait()*.

58249 **NAME**

58250 pthread\_mutex\_getprioceiling, pthread\_mutex\_setprioceiling — get and set the priority ceiling  
58251 of a mutex (**REALTIME THREADS**)

58252 **SYNOPSIS**

```
58253 RPP|TPP #include <pthread.h>
58254 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
58255 int *restrict prioceiling);
58256 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
58257 int prioceiling, int *restrict old_ceiling);
```

58258 **DESCRIPTION**

58259 The *pthread\_mutex\_getprioceiling()* function shall return the current priority ceiling of the mutex.

58260 The *pthread\_mutex\_setprioceiling()* function shall attempt to lock the mutex as if by a call to  
58261 *pthread\_mutex\_lock()*, except that the process of locking the mutex need not adhere to the priority  
58262 protect protocol. On acquiring the mutex it shall change the mutex's priority ceiling and then  
58263 release the mutex as if by a call to *pthread\_mutex\_unlock()*. When the change is successful, the  
58264 previous value of the priority ceiling shall be returned in *old\_ceiling*.

58265 If the *pthread\_mutex\_setprioceiling()* function fails, the mutex priority ceiling shall not be  
58266 changed.

58267 **RETURN VALUE**

58268 If successful, the *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions shall  
58269 return zero; otherwise, an error number shall be returned to indicate the error.

58270 **ERRORS**

58271 These functions shall fail if:

- 58272 [EINVAL] The protocol attribute of *mutex* is PTHREAD\_PRIO\_NONE.
- 58273 [EPERM] The implementation requires appropriate privileges to perform the operation  
58274 and the caller does not have appropriate privileges.

58275 The *pthread\_mutex\_setprioceiling()* function shall fail if:

- 58276 [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
58277 locks for *mutex* has been exceeded.
- 58278 [EAGAIN] The mutex is a robust mutex and the system resources available for robust  
58279 mutexes owned would be exceeded.
- 58280 [EDEADLK] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK and the current  
58281 thread already owns the mutex.
- 58282 [EINVAL] The mutex was created with the protocol attribute having the value  
58283 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
58284 the mutex's current priority ceiling, and the implementation adheres to the  
58285 priority protect protocol in the process of locking the mutex.
- 58286 [ENOTRECOVERABLE]  
58287 The mutex is a robust mutex and the state protected by the mutex is not  
58288 recoverable.
- 58289 [EOWNERDEAD]  
58290 The mutex is a robust mutex and the process containing the previous owning  
58291 thread terminated while holding the mutex lock. The mutex lock shall be



58292 acquired by the calling thread and it is up to the new owner to make the state  
58293 consistent (see *pthread\_mutex\_lock()*).

58294 The *pthread\_mutex\_setprioceiling()* function may fail if:

58295 [EDEADLK] A deadlock condition was detected.

58296 [EINVAL] The priority requested by *prioceiling* is out of range.

58297 [EOWNERDEAD]

58298 The mutex is a robust mutex and the previous owning thread terminated  
58299 while holding the mutex lock. The mutex lock shall be acquired by the calling  
58300 thread and it is up to the new owner to make the state consistent (see  
58301 *pthread\_mutex\_lock()*).

58302 These functions shall not return an error code of [EINTR].

#### 58303 EXAMPLES

58304 None.

#### 58305 APPLICATION USAGE

58306 None.

#### 58307 RATIONALE

58308 None.

#### 58309 FUTURE DIRECTIONS

58310 None.

#### 58311 SEE ALSO

58312 *pthread\_mutex\_clocklock()*, *pthread\_mutex\_destroy()*, *pthread\_mutex\_lock()*

58313 XBD <pthread.h>

#### 58314 CHANGE HISTORY

58315 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

58316 Marked as part of the Realtime Threads Feature Group.

#### 58317 Issue 6

58318 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions are marked as  
58319 part of the Threads and Thread Priority Protection options.

58320 The [ENOSYS] error conditions have been removed.

58321 The *pthread\_mutex\_timedlock()* function is added to the SEE ALSO section for alignment with  
58322 IEEE Std 1003.1d-1999.

58323 The **restrict** keyword is added to the *pthread\_mutex\_getprioceiling()* and  
58324 *pthread\_mutex\_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

#### 58325 Issue 7

58326 SD5-XSH-ERN-39 is applied.

58327 Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a ``may fail''  
58328 error.

58329 SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a ``shall fail'' error case  
58330 for when the protocol attribute of *mutex* is PTHREAD\_PRIO\_NONE.

58331 The *pthread\_mutex\_getprioceiling()* and *pthread\_mutex\_setprioceiling()* functions are moved from  
58332 the Threads option to require support of either the Robust Mutex Priority Protection option or

58333 the Non-Robust Mutex Priority Protection option.

58334 The DESCRIPTION and ERRORS sections are updated to account properly for all of the various  
58335 mutex types.

58336 **Issue 8**

58337 Austin Group Defect 354 is applied, adding the [EAGAIN] error for exceeding system resources  
58338 available for robust mutexes owned.

58339 **NAME**

58340 pthread\_mutex\_init — destroy and initialize a mutex

58341 **SYNOPSIS**

58342 #include &lt;pthread.h&gt;

58343 int pthread\_mutex\_init(pthread\_mutex\_t \*restrict mutex,

58344 const pthread\_mutexattr\_t \*restrict attr);

58345 pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

58346 **DESCRIPTION**58347 Refer to *pthread\_mutex\_destroy()*.

58348 **NAME**

58349 pthread\_mutex\_lock, pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a  
58350 mutex

58351 **SYNOPSIS**

```
58352 #include <pthread.h>
58353 int pthread_mutex_lock(pthread_mutex_t *mutex);
58354 int pthread_mutex_trylock(pthread_mutex_t *mutex);
58355 int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

58356 **DESCRIPTION**

58357 The mutex object referenced by *mutex* shall be locked by a call to *pthread\_mutex\_lock()* that  
58358 returns zero or [EOWNERDEAD]. If the mutex is already locked by another thread, the calling  
58359 thread shall block until the mutex becomes available. This operation shall return with the mutex  
58360 object referenced by *mutex* in the locked state with the calling thread as its owner. If a thread  
58361 attempts to relock a mutex that it has already locked, *pthread\_mutex\_lock()* shall behave as  
58362 described in the **Relock** column of the following table. If a thread attempts to unlock a mutex  
58363 that it has not locked or a mutex which is unlocked, *pthread\_mutex\_unlock()* shall behave as  
58364 described in the **Unlock When Not Owner** column of the following table.

Mutex Type	Robustness	Relock	Unlock When Not Owner
NORMAL	non-robust	deadlock	undefined behavior
NORMAL	robust	deadlock	error returned
ERRORCHECK	either	error returned	error returned
RECURSIVE	either	recursive (see below)	error returned
DEFAULT	non-robust	undefined behavior†	undefined behavior†
DEFAULT	robust	undefined behavior†	error returned

58375 † If the mutex type is PTHREAD\_MUTEX\_DEFAULT, the behavior of *pthread\_mutex\_lock()*  
58376 may correspond to one of the three other standard mutex types as described in the table  
58377 above. If it does not correspond to one of those three, the behavior is undefined for the cases  
58378 marked †.

58379 Where the table indicates recursive behavior, the mutex shall maintain the concept of a lock  
58380 count. When a thread successfully acquires a mutex for the first time, the lock count shall be set  
58381 to one. Every time a thread relocks this mutex, the lock count shall be incremented by one. Each  
58382 time the thread unlocks the mutex, the lock count shall be decremented by one. When the lock  
58383 count reaches zero, the mutex shall become available for other threads to acquire.

58384 The *pthread\_mutex\_trylock()* function shall be equivalent to *pthread\_mutex\_lock()*, except that if  
58385 the mutex object referenced by *mutex* is currently locked (by any thread, including the current  
58386 thread), the call shall return immediately. If the mutex type is PTHREAD\_MUTEX\_RECURSIVE  
58387 and the mutex is currently owned by the calling thread, the mutex lock count shall be  
58388 incremented by one and the *pthread\_mutex\_trylock()* function shall immediately return success.

58389 The *pthread\_mutex\_unlock()* function shall release the mutex object referenced by *mutex*. The  
58390 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are  
58391 threads blocked on the mutex object referenced by *mutex* when *pthread\_mutex\_unlock()* is called,  
58392 resulting in the mutex becoming available, the scheduling policy shall determine which thread  
58393 shall acquire the mutex.

58394 (In the case of PTHREAD\_MUTEX\_RECURSIVE mutexes, the mutex shall become available  
58395 when the count reaches zero and the calling thread no longer has any locks on this mutex.)

58396 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the  
58397 thread shall resume waiting for the mutex as if it was not interrupted.

58398 If *mutex* is a robust mutex and the process containing the owning thread terminated while  
58399 holding the mutex lock, a call to *pthread\_mutex\_lock()* shall return the error value  
58400 [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding  
58401 the mutex lock, a call to *pthread\_mutex\_lock()* may return the error value [EOWNERDEAD] even  
58402 if the process in which the owning thread resides has not terminated. In these cases, the mutex  
58403 shall be locked by the calling thread but the state it protects is marked as inconsistent. The  
58404 application should ensure that the state is made consistent for reuse and when that is complete  
58405 call *pthread\_mutex\_consistent()*. If the application is unable to recover the state, it should unlock  
58406 the mutex without a prior call to *pthread\_mutex\_consistent()*, after which the mutex is marked  
58407 permanently unusable.

58408 If *mutex* does not refer to an initialized mutex object, the behavior of *pthread\_mutex\_lock()*,  
58409 *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()* is undefined.

#### 58410 RETURN VALUE

58411 If successful, the *pthread\_mutex\_lock()*, *pthread\_mutex\_trylock()*, and *pthread\_mutex\_unlock()*  
58412 functions shall return zero; otherwise, an error number shall be returned to indicate the error.

#### 58413 ERRORS

58414 The *pthread\_mutex\_lock()* and *pthread\_mutex\_trylock()* functions shall fail if:

58415 [EAGAIN] The mutex could not be acquired because the maximum number of recursive  
58416 locks for *mutex* has been exceeded.

58417 [EAGAIN] The mutex is a robust mutex and the system resources available for robust  
58418 mutexes owned would be exceeded.

58419 RPP|TPP [EINVAL] The *mutex* was created with the protocol attribute having the value  
58420 PTHREAD\_PRIO\_PROTECT and the calling thread's priority is higher than  
58421 the mutex's current priority ceiling.

58422 [ENOTRECOVERABLE]

58423 The state protected by the mutex is not recoverable.

58424 [EOWNERDEAD]

58425 The mutex is a robust mutex and the process containing the previous owning  
58426 thread terminated while holding the mutex lock. The mutex lock shall be  
58427 acquired by the calling thread and it is up to the new owner to make the state  
58428 consistent.

58429 The *pthread\_mutex\_lock()* function shall fail if:

58430 [EDEADLK] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK and the current  
58431 thread already owns the mutex.

58432 The *pthread\_mutex\_trylock()* function shall fail if:

58433 [EBUSY] The *mutex* could not be acquired because it was already locked.

58434 The *pthread\_mutex\_unlock()* function shall fail if:

58435 [EPERM] The mutex type is PTHREAD\_MUTEX\_ERRORCHECK or  
58436 PTHREAD\_MUTEX\_RECURSIVE, or the mutex is a robust mutex, and the  
58437 current thread does not own the mutex.

58438 The `pthread_mutex_lock()` and `pthread_mutex_trylock()` functions may fail if:  
58439 [EOWNERDEAD]  
58440 The mutex is a robust mutex and the previous owning thread terminated  
58441 while holding the mutex lock. The mutex lock shall be acquired by the calling  
58442 thread and it is up to the new owner to make the state consistent.

58443 The `pthread_mutex_lock()` function may fail if:  
58444 [EDEADLK] A deadlock condition was detected.  
58445 These functions shall not return an error code of [EINTR].

#### 58446 EXAMPLES

58447 None.

#### 58448 APPLICATION USAGE

58449 Applications that have assumed that non-zero return values are errors will need updating for  
58450 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting  
58451 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error  
58452 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If  
58453 an application is supposed to work with normal and robust mutexes it should check all return  
58454 values for error conditions and if necessary take appropriate action.

#### 58455 RATIONALE

58456 Mutex objects are intended to serve as a low-level primitive from which other thread  
58457 synchronization functions can be built. As such, the implementation of mutexes should be as  
58458 efficient as possible, and this has ramifications on the features available at the interface.

58459 The mutex functions and the particular default settings of the mutex attributes have been  
58460 motivated by the desire to not preclude fast, inlined implementations of mutex locking and  
58461 unlocking.

58462 Since most attributes only need to be checked when a thread is going to be blocked, the use of  
58463 attributes does not slow the (common) mutex-locking case.

58464 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it  
58465 would require storing the current thread ID when each mutex is locked, and this could incur  
58466 unacceptable levels of overhead. Similar arguments apply to a `mutex_tryunlock` operation.

58467 For further rationale on the extended mutex types, see XRAT [Threads Extensions](#) (on page 3815).

58468 If an implementation detects that the value specified by the `mutex` argument does not refer to an  
58469 initialized mutex object, it is recommended that the function should fail and report an [EINVAL]  
58470 error.

#### 58471 FUTURE DIRECTIONS

58472 None.

#### 58473 SEE ALSO

58474 [pthread\\_mutex\\_clocklock\(\)](#), [pthread\\_mutex\\_consistent\(\)](#), [pthread\\_mutex\\_destroy\(\)](#),  
58475 [pthread\\_mutexattr\\_getrobust\(\)](#)

58476 XBD Section 4.15.2 (on page 104), [<pthread.h>](#)

#### 58477 CHANGE HISTORY

58478 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**Issue 6**

58479 The `pthread_mutex_lock()`, `pthread_mutex_trylock()`, and `pthread_mutex_unlock()` functions are  
58480 marked as part of the Threads option.  
58481

58482 The following new requirements on POSIX implementations derive from alignment with the  
58483 Single UNIX Specification:

- 58484 • The behavior when attempting to relock a mutex is defined.

58485 The `pthread_mutex_timedlock()` function is added to the SEE ALSO section for alignment with  
58486 IEEE Std 1003.1d-1999.

58487 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/98 is applied, updating the ERRORS  
58488 section so that the [EDEADLK] error includes detection of a deadlock condition. The  
58489 RATIONALE section is also reworded to take into account non-XSI-conformant systems.

**Issue 7**

58490 SD5-XSH-ERN-43 is applied, marking the “shall fail” case of the [EINVAL] error as dependent  
58491 on the Thread Priority Protection option.  
58492

58493 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

58494 The `pthread_mutex_lock()`, `pthread_mutex_trylock()`, and `pthread_mutex_unlock()` functions are  
58495 moved from the Threads option to the Base.

58496 The following extended mutex types are moved from the XSI option to the Base:

```
58497     PTHREAD_MUTEX_NORMAL  
58498     PTHREAD_MUTEX_ERRORCHECK  
58499     PTHREAD_MUTEX_RECURSIVE  
58500     PTHREAD_MUTEX_DEFAULT
```

58501 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized  
58502 mutex.

58503 The ERRORS section is updated to account properly for all of the various mutex types.

58504 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0461 [121], XSH/TC1-2008/0462  
58505 [92,428], and XSH/TC1-2008/0463 [121] are applied.

**Issue 8**

58506 Austin Group Defect 354 is applied, adding the [EAGAIN] error for exceeding system resources  
58507 available for robust mutexes owned.  
58508

58509 Austin Group Defect 1115 is applied, changing “the thread” to “the calling thread”.

58510 **NAME**

58511 pthread\_mutex\_setprioceiling — change the priority ceiling of a mutex (**REALTIME**  
58512 **THREADS**)

58513 **SYNOPSIS**

```
58514 RPP|TPP #include <pthread.h>
58515 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
58516 int prioceiling, int *restrict old_ceiling);
```

58517 **DESCRIPTION**

58518 Refer to [pthread\\_mutex\\_getprioceiling\(\)](#).



58519 **NAME**

58520 pthread\_mutex\_timedlock — lock a mutex

58521 **SYNOPSIS**

58522 #include &lt;pthread.h&gt;

58523 int pthread\_mutex\_timedlock(pthread\_mutex\_t \*restrict mutex,  
58524 const struct timespec \*restrict abstime);58525 **DESCRIPTION**58526 Refer to [pthread\\_mutex\\_clocklock\(\)](#).

58527 **NAME**

58528 pthread\_mutex\_trylock, pthread\_mutex\_unlock — lock and unlock a mutex

58529 **SYNOPSIS**

58530 #include <pthread.h>

58531 int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

58532 int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

58533 **DESCRIPTION**

58534 Refer to *pthread\_mutex\_lock()*.

58535 **NAME**

58536 pthread\_mutexattr\_destroy, pthread\_mutexattr\_init — destroy and initialize the mutex  
58537 attributes object

58538 **SYNOPSIS**

```
58539 #include <pthread.h>
58540 int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
58541 int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

58542 **DESCRIPTION**

58543 The *pthread\_mutexattr\_destroy()* function shall destroy a mutex attributes object; the object  
58544 becomes, in effect, uninitialized. An implementation may cause *pthread\_mutexattr\_destroy()* to  
58545 set the object referenced by *attr* to an invalid value.

58546 A destroyed *attr* attributes object can be reinitialized using *pthread\_mutexattr\_init()*; the results of  
58547 otherwise referencing the object after it has been destroyed are undefined.

58548 The *pthread\_mutexattr\_init()* function shall initialize a mutex attributes object *attr* with the  
58549 default value for all of the attributes defined by the implementation.

58550 Results are undefined if *pthread\_mutexattr\_init()* is called specifying an already initialized *attr*  
58551 attributes object.

58552 After a mutex attributes object has been used to initialize one or more mutexes, any function  
58553 affecting the attributes object (including destruction) shall not affect any previously initialized  
58554 mutexes.

58555 The behavior is undefined if the value specified by the *attr* argument to  
58556 *pthread\_mutexattr\_destroy()* does not refer to an initialized mutex attributes object.

58557 **RETURN VALUE**

58558 Upon successful completion, *pthread\_mutexattr\_destroy()* and *pthread\_mutexattr\_init()* shall  
58559 return zero; otherwise, an error number shall be returned to indicate the error.

58560 **ERRORS**

58561 The *pthread\_mutexattr\_init()* function shall fail if:

58562 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

58563 These functions shall not return an error code of [EINTR].

58564 **EXAMPLES**

58565 None.

58566 **APPLICATION USAGE**

58567 None.

58568 **RATIONALE**

58569 If an implementation detects that the value specified by the *attr* argument to  
58570 *pthread\_mutexattr\_destroy()* does not refer to an initialized mutex attributes object, it is  
58571 recommended that the function should fail and report an [EINVAL] error.

58572 See *pthread\_attr\_destroy()* for a general explanation of attributes. Attributes objects allow  
58573 implementations to experiment with useful extensions and permit extension of this volume of  
58574 POSIX.1-2024 without changing the existing functions. Thus, they provide for future  
58575 extensibility of this volume of POSIX.1-2024 and reduce the temptation to standardize  
58576 prematurely on semantics that are not yet widely implemented or understood.

58577 Examples of possible additional mutex attributes that have been discussed are *spin\_only*,  
58578 *limited\_spin*, *no\_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:

58579 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes  
 58580 would transparently keep records of queue length, wait time, and so on.) Since there is not yet  
 58581 wide agreement on the usefulness of these resulting from shared implementation and usage  
 58582 experience, they are not yet specified in this volume of POSIX.1-2024. Mutex attributes objects,  
 58583 however, make it possible to test out these concepts for possible standardization at a later time.

#### 58584 **Mutex Attributes and Performance**

58585 Care has been taken to ensure that the default values of the mutex attributes have been defined  
 58586 such that mutexes initialized with the defaults have simple enough semantics so that the locking  
 58587 and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few  
 58588 other basic instructions).

58589 There is at least one implementation method that can be used to reduce the cost of testing at  
 58590 lock-time if a mutex has non-default attributes. One such method that an implementation can  
 58591 employ (and this can be made fully transparent to fully conforming POSIX applications) is to  
 58592 secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to  
 58593 lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were  
 58594 unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-  
 58595 default mutex. The underlying unlock operation is more complicated since the implementation  
 58596 never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending  
 58597 on the hardware, there may be certain optimizations that can be used so that whatever mutex  
 58598 attributes are considered “most frequently used” can be processed most efficiently.

#### 58599 **Process Shared Memory and Synchronization**

58600 The existence of memory mapping functions in this volume of POSIX.1-2024 leads to the  
 58601 possibility that an application may allocate the synchronization objects from this section in  
 58602 memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

58603 In order to permit such usage, while at the same time keeping the usual case (that is, usage  
 58604 within a single process) efficient, a *process-shared* option has been defined.

58605 If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the  
 58606 *process-shared* attribute can be used to indicate that mutexes or condition variables may be  
 58607 accessed by threads of multiple processes.

58608 The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared*  
 58609 attribute so that the most efficient forms of these synchronization objects are created by default.

58610 Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-*  
 58611 *shared* attribute may only be operated on by threads in the process that initialized them.  
 58612 Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-*  
 58613 *shared* attribute may be operated on by any thread in any process that has access to it. In  
 58614 particular, these processes may exist beyond the lifetime of the initializing process. For example,  
 58615 the following code implements a simple counting semaphore in a mapped file that may be used  
 58616 by many processes.

```
58617 /* sem.h */
58618 struct semaphore {
58619     pthread_mutex_t lock;
58620     pthread_cond_t nonzero;
58621     unsigned count;
58622 };
58623 typedef struct semaphore semaphore_t;
```

```

58624 semaphore_t *semaphore_create(char *semaphore_name);
58625 semaphore_t *semaphore_open(char *semaphore_name);
58626 void semaphore_post(semaphore_t *semap);
58627 void semaphore_wait(semaphore_t *semap);
58628 void semaphore_close(semaphore_t *semap);

58629 /* sem.c */
58630 #include <sys/types.h>
58631 #include <sys/stat.h>
58632 #include <sys/mman.h>
58633 #include <fcntl.h>
58634 #include <pthread.h>
58635 #include "sem.h"

58636 semaphore_t *
58637 semaphore_create(char *semaphore_name)
58638 {
58639     int fd;
58640     semaphore_t *semap;
58641     pthread_mutexattr_t psharedm;
58642     pthread_condattr_t psharedc;

58643     fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
58644     if (fd < 0)
58645         return (NULL);
58646     (void) ftruncate(fd, sizeof(semaphore_t));
58647     (void) pthread_mutexattr_init(&psharedm);
58648     (void) pthread_mutexattr_setpshared(&psharedm,
58649         PTHREAD_PROCESS_SHARED);
58650     (void) pthread_condattr_init(&psharedc);
58651     (void) pthread_condattr_setpshared(&psharedc,
58652         PTHREAD_PROCESS_SHARED);
58653     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
58654         PROT_READ | PROT_WRITE, MAP_SHARED,
58655         fd, 0);
58656     close (fd);
58657     (void) pthread_mutex_init(&semap->lock, &psharedm);
58658     (void) pthread_cond_init(&semap->nonzero, &psharedc);
58659     semap->count = 0;
58660     return (semap);
58661 }

58662 semaphore_t *
58663 semaphore_open(char *semaphore_name)
58664 {
58665     int fd;
58666     semaphore_t *semap;

58667     fd = open(semaphore_name, O_RDWR, 0666);
58668     if (fd < 0)
58669         return (NULL);
58670     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
58671         PROT_READ | PROT_WRITE, MAP_SHARED,
58672         fd, 0);

```

```

58673         close (fd);
58674         return (semap);
58675     }

58676     void
58677     semaphore_post (semaphore_t *semap)
58678     {
58679         pthread_mutex_lock (&semap->lock);
58680         if (semap->count == 0)
58681             pthread_cond_signal (&semap->nonzero);
58682         semap->count++;
58683         pthread_mutex_unlock (&semap->lock);
58684     }

58685     void
58686     semaphore_wait (semaphore_t *semap)
58687     {
58688         pthread_mutex_lock (&semap->lock);
58689         while (semap->count == 0)
58690             pthread_cond_wait (&semap->nonzero, &semap->lock);
58691         semap->count--;
58692         pthread_mutex_unlock (&semap->lock);
58693     }

58694     void
58695     semaphore_close (semaphore_t *semap)
58696     {
58697         munmap ((void *) semap, sizeof (semaphore_t));
58698     }

```

58699 The following code is for three separate processes that create, post, and wait on a semaphore in  
58700 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and  
58701 decrement the counting semaphore (waiting and waking as required) even though they did not  
58702 initialize the semaphore.

```

58703     /* create.c */
58704     #include "pthread.h"
58705     #include "sem.h"

58706     int
58707     main (void)
58708     {
58709         semaphore_t *semap;

58710         semap = semaphore_create ("/tmp/semaphore");
58711         if (semap == NULL)
58712             exit (1);
58713         semaphore_close (semap);
58714         return (0);
58715     }

58716     /* post.c */
58717     #include "pthread.h"
58718     #include "sem.h"

58719     int

```

```

58720     main(void)
58721     {
58722         semaphore_t *semaph;
58723
58724         semaph = semaphore_open("/tmp/semaphore");
58725         if (semaph == NULL)
58726             exit(1);
58727         semaphore_post(semaph);
58728         semaphore_close(semaph);
58729         return (0);
58730     }
58731
58732     /* wait.c */
58733     #include "pthread.h"
58734     #include "sem.h"
58735
58736     int
58737     main(void)
58738     {
58739         semaphore_t *semaph;
58740
58741         semaph = semaphore_open("/tmp/semaphore");
58742         if (semaph == NULL)
58743             exit(1);
58744         semaphore_wait(semaph);
58745         semaphore_close(semaph);
58746         return (0);
58747     }

```

**58744 FUTURE DIRECTIONS**

58745 None.

**58746 SEE ALSO**

58747 [pthread\\_cond\\_destroy\(\)](#), [pthread\\_create\(\)](#), [pthread\\_mutex\\_destroy\(\)](#)

58748 XBD [<pthread.h>](#)

**58749 CHANGE HISTORY**

58750 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**58751 Issue 6**

58752 The [pthread\\_mutexattr\\_destroy\(\)](#) and [pthread\\_mutexattr\\_init\(\)](#) functions are marked as part of the  
58753 Threads option.

58754 IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

**58755 Issue 7**

58756 The [pthread\\_mutexattr\\_destroy\(\)](#) and [pthread\\_mutexattr\\_init\(\)](#) functions are moved from the  
58757 Threads option to the Base.

58758 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
58759 results in undefined behavior.

**58760 Issue 8**

58761 Austin Group Defect 1195 is applied, changing `main()` to `main(void)`.

58762 **NAME**

58763 pthread\_mutexattr\_getprioceiling, pthread\_mutexattr\_setprioceiling — get and set the  
58764 prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

58765 **SYNOPSIS**

```
58766 RPP|TPP #include <pthread.h>
58767 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
58768     *restrict attr, int *restrict prioceiling);
58769 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
58770     int prioceiling);
```

58771 **DESCRIPTION**

58772 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions,  
58773 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to  
58774 by *attr* which was previously created by the function *pthread\_mutexattr\_init()*.

58775 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of  
58776 *prioceiling* are within the maximum range of priorities defined by **SCHED\_FIFO**.

58777 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum  
58778 priority level at which the critical section guarded by the mutex is executed. In order to avoid  
58779 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal  
58780 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are  
58781 within the maximum range of priorities defined under the **SCHED\_FIFO** scheduling policy.

58782 The behavior is undefined if the value specified by the *attr* argument to  
58783 *pthread\_mutexattr\_getprioceiling()* or *pthread\_mutexattr\_setprioceiling()* does not refer to an  
58784 initialized mutex attributes object.

58785 **RETURN VALUE**

58786 Upon successful completion, the *pthread\_mutexattr\_getprioceiling()* and  
58787 *pthread\_mutexattr\_setprioceiling()* functions shall return zero; otherwise, an error number shall be  
58788 returned to indicate the error.

58789 **ERRORS**

58790 These functions may fail if:

58791 [EINVAL] The value specified by *prioceiling* is invalid.

58792 [EPERM] The caller does not have the privilege to perform the operation.

58793 These functions shall not return an error code of [EINTR].

58794 **EXAMPLES**

58795 None.

58796 **APPLICATION USAGE**

58797 None.

58798 **RATIONALE**

58799 If an implementation detects that the value specified by the *attr* argument to  
58800 *pthread\_mutexattr\_getprioceiling()* or *pthread\_mutexattr\_setprioceiling()* does not refer to an  
58801 initialized mutex attributes object, it is recommended that the function should fail and report an  
58802 [EINVAL] error.



58803 **FUTURE DIRECTIONS**

58804 None.

58805 **SEE ALSO**58806 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*

58807 XBD &lt;pthread.h&gt;

58808 **CHANGE HISTORY**

58809 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

58810 Marked as part of the Realtime Threads Feature Group.

58811 **Issue 6**58812 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions are marked  
58813 as part of the Threads and Thread Priority Protection options.58814 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
58815 implementation does not support the Thread Priority Protection option.58816 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*  
58817 argument.58818 The **restrict** keyword is added to the *pthread\_mutexattr\_getprioceiling()* prototype for alignment  
58819 with the ISO/IEC 9899:1999 standard.58820 **Issue 7**58821 The *pthread\_mutexattr\_getprioceiling()* and *pthread\_mutexattr\_setprioceiling()* functions are moved  
58822 from the Threads option to require support of either the Robust Mutex Priority Protection option  
58823 or the Non-Robust Mutex Priority Protection option.58824 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
58825 results in undefined behavior.

58826 **NAME**

58827 pthread\_mutexattr\_getprotocol, pthread\_mutexattr\_setprotocol — get and set the protocol  
58828 attribute of the mutex attributes object (**REALTIME THREADS**)

58829 **SYNOPSIS**

```
58830 MC1 #include <pthread.h>
58831 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
58832     *restrict attr, int *restrict protocol);
58833 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
58834     int protocol);
```

58835 **DESCRIPTION**

58836 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions, respectively,  
58837 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was  
58838 previously created by the function *pthread\_mutexattr\_init()*.

58839 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of  
58840 *protocol* may be one of:

```
58841 RPI|TPI PTHREAD_PRIO_INHERIT
58842 MC1 PTHREAD_PRIO_NONE
58843 RPP|TPP PTHREAD_PRIO_PROTECT
```

58844 which are defined in the **<pthread.h>** header. The default value of the attribute shall be  
58845 PTHREAD\_PRIO\_NONE.

58846 When a thread owns a mutex with the PTHREAD\_PRIO\_NONE *protocol* attribute, its priority  
58847 and scheduling shall not be affected by its mutex ownership.

58848 RPI When a thread is blocking higher priority threads because of owning one or more robust  
58849 mutexes with the PTHREAD\_PRIO\_INHERIT *protocol* attribute, it shall execute at the higher of  
58850 its priority or the priority of the highest priority thread waiting on any of the robust mutexes  
58851 owned by this thread and initialized with this protocol.

58852 TPI When a thread is blocking higher priority threads because of owning one or more non-robust  
58853 mutexes with the PTHREAD\_PRIO\_INHERIT *protocol* attribute, it shall execute at the higher of  
58854 its priority or the priority of the highest priority thread waiting on any of the non-robust  
58855 mutexes owned by this thread and initialized with this protocol.

58856 RPP When a thread owns one or more robust mutexes initialized with the  
58857 PTHREAD\_PRIO\_PROTECT protocol, it shall execute at the higher of its priority or the highest  
58858 of the priority ceilings of all the robust mutexes owned by this thread and initialized with this  
58859 attribute, regardless of whether other threads are blocked on any of these robust mutexes or not.

58860 TPP When a thread owns one or more non-robust mutexes initialized with the  
58861 PTHREAD\_PRIO\_PROTECT protocol, it shall execute at the higher of its priority or the highest  
58862 of the priority ceilings of all the non-robust mutexes owned by this thread and initialized with  
58863 this attribute, regardless of whether other threads are blocked on any of these non-robust  
58864 mutexes or not.

58865 If a thread simultaneously owns several mutexes initialized with different protocols, it shall  
58866 execute at the highest of the priorities that it would have obtained by each of these protocols.

58867 RPI|TPI When a thread makes a call to *pthread\_mutex\_lock()*, the mutex was initialized with the protocol attribute having the value `PTHREAD_PRIO_INHERIT`, when the calling thread is blocked because the mutex is owned by another thread, that owner thread shall inherit the priority level of the calling thread as long as it continues to own the mutex. The implementation shall update its execution priority to the maximum of its assigned priority and all its inherited priorities. Furthermore, if this owner thread itself becomes blocked on another mutex with the *protocol* attribute having the value `PTHREAD_PRIO_INHERIT`, the same priority inheritance effect shall be propagated to this other owner thread, in a recursive manner.

58875 The behavior is undefined if the value specified by the *attr* argument to *pthread\_mutexattr\_getprotocol()* or *pthread\_mutexattr\_setprotocol()* does not refer to an initialized mutex attributes object.

#### 58878 RETURN VALUE

58879 Upon successful completion, the *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

#### 58882 ERRORS

58883 The *pthread\_mutexattr\_setprotocol()* function shall fail if:

58884 [ENOTSUP] The value specified by *protocol* is an unsupported value.

58885 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions may fail if:

58886 [EINVAL] The value specified by *protocol* is invalid.

58887 [EPERM] The caller does not have the privilege to perform the operation.

58888 These functions shall not return an error code of [EINTR].

#### 58889 EXAMPLES

58890 None.

#### 58891 APPLICATION USAGE

58892 None.

#### 58893 RATIONALE

58894 If an implementation detects that the value specified by the *attr* argument to *pthread\_mutexattr\_getprotocol()* or *pthread\_mutexattr\_setprotocol()* does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an [EINVAL] error.

#### 58898 FUTURE DIRECTIONS

58899 None.

#### 58900 SEE ALSO

58901 [\*pthread\\_cond\\_destroy\(\)\*](#), [\*pthread\\_create\(\)\*](#), [\*pthread\\_mutex\\_destroy\(\)\*](#)

58902 XBD <[pthread.h](#)>

#### 58903 CHANGE HISTORY

58904 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

58905 Marked as part of the Realtime Threads Feature Group.

#### 58906 Issue 6

58907 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions are marked as part of the Threads option and either the Thread Priority Protection or Thread Priority Inheritance options.

58910 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
58911 implementation does not support the Thread Priority Protection or Thread Priority Inheritance  
58912 options.

58913 The **restrict** keyword is added to the *pthread\_mutexattr\_getprotocol()* prototype for alignment  
58914 with the ISO/IEC 9899:1999 standard.

58915 **Issue 7**

58916 SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the  
58917 *protocol* attribute.

58918 SD5-XSH-ERN-188 is applied, updating the DESCRIPTION.

58919 The *pthread\_mutexattr\_getprotocol()* and *pthread\_mutexattr\_setprotocol()* functions are moved from  
58920 the Threads option to require support of either the Non-Robust Mutex Priority Protection option  
58921 or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection  
58922 option or the Robust Mutex Priority Inheritance option.

58923 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
58924 results in undefined behavior.

58925 **Issue 8**

58926 Austin Group Defect 1610 is applied, moving text relating to the effects of  
58927 PTHREAD\_PRIO\_INHERIT and PTHREAD\_PRIO\_PROTECT on scheduling queues to  
58928 [Scheduling Policies](#) (on page 531).

58929 **NAME**

58930 pthread\_mutexattr\_getpshared, pthread\_mutexattr\_setpshared — get and set the process-shared  
58931 attribute

58932 **SYNOPSIS**

```
58933 TSH #include <pthread.h>
58934 int pthread_mutexattr_getpshared(const pthread_mutexattr_t
58935     *restrict attr, int *restrict pshared);
58936 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
58937     int pshared);
```

58938 **DESCRIPTION**

58939 The *pthread\_mutexattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
58940 from the attributes object referenced by *attr*.

58941 The *pthread\_mutexattr\_setpshared()* function shall set the *process-shared* attribute in an initialized  
58942 attributes object referenced by *attr*.

58943 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a mutex to be  
58944 operated upon by any thread that has access to the memory where the mutex is allocated, even if  
58945 the mutex is allocated in memory that is shared by multiple processes. See [Section 2.9.9](#) (on page  
58946 548) for further requirements. The default value of the attribute shall be  
58947 `PTHREAD_PROCESS_PRIVATE`.

58948 The behavior is undefined if the value specified by the *attr* argument to  
58949 *pthread\_mutexattr\_getpshared()* or *pthread\_mutexattr\_setpshared()* does not refer to an initialized  
58950 mutex attributes object.

58951 **RETURN VALUE**

58952 Upon successful completion, *pthread\_mutexattr\_setpshared()* shall return zero; otherwise, an error  
58953 number shall be returned to indicate the error.

58954 Upon successful completion, *pthread\_mutexattr\_getpshared()* shall return zero and store the value  
58955 of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.  
58956 Otherwise, an error number shall be returned to indicate the error.

58957 **ERRORS**

58958 The *pthread\_mutexattr\_setpshared()* function may fail if:

58959 [EINVAL] The new value specified for the attribute is outside the range of legal values  
58960 for that attribute.

58961 These functions shall not return an error code of [EINTR].

58962 **EXAMPLES**

58963 None.

58964 **APPLICATION USAGE**

58965 None.

58966 **RATIONALE**

58967 If an implementation detects that the value specified by the *attr* argument to  
58968 *pthread\_mutexattr\_getpshared()* or *pthread\_mutexattr\_setpshared()* does not refer to an initialized  
58969 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]  
58970 error.

58971 **FUTURE DIRECTIONS**

58972 None.

58973 **SEE ALSO**58974 *pthread\_cond\_destroy()*, *pthread\_create()*, *pthread\_mutex\_destroy()*, *pthread\_mutexattr\_destroy()*

58975 XBD &lt;pthread.h&gt;

58976 **CHANGE HISTORY**

58977 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

58978 **Issue 6**58979 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions are marked as  
58980 part of the Threads and Thread Process-Shared Synchronization options.58981 The **restrict** keyword is added to the *pthread\_mutexattr\_getpshared()* prototype for alignment  
58982 with the ISO/IEC 9899:1999 standard.58983 **Issue 7**58984 The *pthread\_mutexattr\_getpshared()* and *pthread\_mutexattr\_setpshared()* functions are marked only  
58985 as part of the Thread Process-Shared Synchronization option as the Threads option is now part  
58986 of the Base.58987 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
58988 results in undefined behavior.58989 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0281 [972] and XSH/TC2-2008/0282  
58990 [757] are applied.

58991 **NAME**

58992 pthread\_mutexattr\_getrobust, pthread\_mutexattr\_setrobust — get and set the mutex robust  
58993 attribute

58994 **SYNOPSIS**

```
58995 #include <pthread.h>
58996 int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
58997     attr, int *restrict robust);
58998 int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
58999     int robust);
```

59000 **DESCRIPTION**

59001 The *pthread\_mutexattr\_getrobust()* and *pthread\_mutexattr\_setrobust()* functions, respectively, shall  
59002 get and set the mutex *robust* attribute. This attribute is set in the *robust* parameter. Valid values  
59003 for *robust* include:

59004 **PTHREAD\_MUTEX\_STALLED**

59005 No special actions are taken if the owner of the mutex is terminated while holding the  
59006 mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.  
59007 This is the default value.

59008 **PTHREAD\_MUTEX\_ROBUST**

59009 If the process containing the owning thread of a robust mutex terminates while holding the  
59010 mutex lock, the next thread that acquires the mutex shall be notified about the termination  
59011 by the return value [EOWNERDEAD] from the locking function. If the owning thread of a  
59012 robust mutex terminates while holding the mutex lock, the next thread that attempts to  
59013 acquire the mutex may be notified about the termination by the return value  
59014 [EOWNERDEAD]. The notified thread can then attempt to make the state protected by the  
59015 mutex consistent again, and if successful can mark the mutex state as consistent by calling  
59016 *pthread\_mutex\_consistent()*. After a subsequent successful call to *pthread\_mutex\_unlock()*, the  
59017 mutex lock shall be released and can be used normally by other threads. If the mutex is  
59018 unlocked without a call to *pthread\_mutex\_consistent()*, it shall be in a permanently unusable  
59019 state and all attempts to lock the mutex shall fail with the error [ENOTRECOVERABLE].  
59020 The only permissible operation on such a mutex is *pthread\_mutex\_destroy()*.

59021 The behavior is undefined if the value specified by the *attr* argument to  
59022 *pthread\_mutexattr\_getrobust()* or *pthread\_mutexattr\_setrobust()* does not refer to an initialized  
59023 mutex attributes object.

59024 **RETURN VALUE**

59025 Upon successful completion, the *pthread\_mutexattr\_getrobust()* function shall return zero and  
59026 store the value of the *robust* attribute of *attr* into the object referenced by the *robust* parameter.  
59027 Otherwise, an error value shall be returned to indicate the error. If successful, the  
59028 *pthread\_mutexattr\_setrobust()* function shall return zero; otherwise, an error number shall be  
59029 returned to indicate the error.

59030 **ERRORS**

59031 The *pthread\_mutexattr\_setrobust()* function shall fail if:

59032 [EINVAL] The value of *robust* is invalid.

59033 These functions shall not return an error code of [EINTR].

59034 **EXAMPLES**

59035 None.

59036 **APPLICATION USAGE**

59037 The actions required to make the state protected by the mutex consistent again are solely  
59038 dependent on the application. If it is not possible to make the state of a mutex consistent, robust  
59039 mutexes can be used to notify this situation by calling *pthread\_mutex\_unlock()* without a prior  
59040 call to *pthread\_mutex\_consistent()*.

59041 If the state is declared inconsistent by calling *pthread\_mutex\_unlock()* without a prior call to  
59042 *pthread\_mutex\_consistent()*, a possible approach could be to destroy the mutex and then  
59043 reinitialize it. However, it should be noted that this is possible only in certain situations where  
59044 the state protected by the mutex has to be reinitialized and coordination achieved with other  
59045 threads blocked on the mutex, because otherwise a call to a locking function with a reference to a  
59046 mutex object invalidated by a call to *pthread\_mutex\_destroy()* results in undefined behavior.

59047 **RATIONALE**

59048 If an implementation detects that the value specified by the *attr* argument to  
59049 *pthread\_mutexattr\_getrobust()* or *pthread\_mutexattr\_setrobust()* does not refer to an initialized  
59050 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]  
59051 error.

59052 **FUTURE DIRECTIONS**

59053 None.

59054 **SEE ALSO**59055 [\*pthread\\_mutex\\_consistent\(\)\*](#), [\*pthread\\_mutex\\_destroy\(\)\*](#), [\*pthread\\_mutex\\_lock\(\)\*](#)59056 XBD <[\*pthread.h\*](#)>59057 **CHANGE HISTORY**

59058 First released in Issue 7.

59059 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0283 [748] is applied.



59060 **NAME**

59061 pthread\_mutexattr\_gettype, pthread\_mutexattr\_settype — get and set the mutex type attribute

59062 **SYNOPSIS**

```
59063 #include <pthread.h>
59064 int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
59065 int *restrict type);
59066 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

59067 **DESCRIPTION**

59068 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions, respectively, shall get  
 59069 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The  
 59070 default value of the *type* attribute is PTHREAD\_MUTEX\_DEFAULT.

59071 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types  
 59072 include:

```
59073 PTHREAD_MUTEX_NORMAL
59074 PTHREAD_MUTEX_ERRORCHECK
59075 PTHREAD_MUTEX_RECURSIVE
59076 PTHREAD_MUTEX_DEFAULT
```

59077 The mutex type affects the behavior of calls which lock and unlock the mutex. See  
 59078 *pthread\_mutex\_lock()* for details. An implementation may map PTHREAD\_MUTEX\_DEFAULT  
 59079 to one of the other mutex types.

59080 The behavior is undefined if the value specified by the *attr* argument to  
 59081 *pthread\_mutexattr\_gettype()* or *pthread\_mutexattr\_settype()* does not refer to an initialized mutex  
 59082 attributes object.

59083 **RETURN VALUE**

59084 Upon successful completion, the *pthread\_mutexattr\_gettype()* function shall return zero and store  
 59085 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,  
 59086 an error shall be returned to indicate the error.

59087 If successful, the *pthread\_mutexattr\_settype()* function shall return zero; otherwise, an error  
 59088 number shall be returned to indicate the error.

59089 **ERRORS**

59090 The *pthread\_mutexattr\_settype()* function shall fail if:

59091 [EINVAL] The value *type* is invalid.

59092 These functions shall not return an error code of [EINTR].

59093 **EXAMPLES**

59094 None.

59095 **APPLICATION USAGE**

59096 It is advised that an application should not use a PTHREAD\_MUTEX\_RECURSIVE mutex with  
 59097 condition variables because the implicit unlock performed in a *pthread\_cond\_clockwait()*,  
 59098 *pthread\_cond\_timedwait()*, or *pthread\_cond\_wait()* call may not actually release the mutex (if it had  
 59099 been locked multiple times). If this happens, no other thread can satisfy the condition of the  
 59100 predicate.

59101 **RATIONALE**

59102 If an implementation detects that the value specified by the *attr* argument to  
59103 *pthread\_mutexattr\_gettype()* or *pthread\_mutexattr\_settype()* does not refer to an initialized mutex  
59104 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

59105 **FUTURE DIRECTIONS**

59106 None.

59107 **SEE ALSO**

59108 *pthread\_cond\_clockwait()*, *pthread\_mutex\_lock()*

59109 XBD <pthread.h>

59110 **CHANGE HISTORY**

59111 First released in Issue 5.

59112 **Issue 6**

59113 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for  
59114 *pthread\_mutexattr\_gettype()* is updated so that the first argument is of type **const**  
59115 **pthread\_mutexattr\_t\***.

59116 The **restrict** keyword is added to the *pthread\_mutexattr\_gettype()* prototype for alignment with  
59117 the ISO/IEC 9899:1999 standard.

59118 **Issue 7**

59119 The *pthread\_mutexattr\_gettype()* and *pthread\_mutexattr\_settype()* functions are moved from the  
59120 XSI option to the Base.

59121 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition  
59122 results in undefined behavior.

59123 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0464 [121] is applied.

59124 **Issue 8**

59125 Austin Group Defect 1216 is applied, adding *pthread\_cond\_clockwait()*.

59126 **NAME**

59127 pthread\_mutexattr\_init — initialize the mutex attributes object

59128 **SYNOPSIS**

59129 #include &lt;pthread.h&gt;

59130 int pthread\_mutexattr\_init(pthread\_mutexattr\_t \*attr);

59131 **DESCRIPTION**59132 Refer to *pthread\_mutexattr\_destroy()*.

59133 **NAME**

59134 pthread\_mutexattr\_setprioceiling — set the prioceiling attribute of the mutex attributes object  
59135 (**REALTIME THREADS**)

59136 **SYNOPSIS**

```
59137 RPP|TPP #include <pthread.h>  
59138 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,  
59139 int prioceiling);
```

59140 **DESCRIPTION**

59141 Refer to [pthread\\_mutexattr\\_getprioceiling\(\)](#).

59142 **NAME**

59143 pthread\_mutexattr\_setprotocol — set the protocol attribute of the mutex attributes object  
59144 (**REALTIME THREADS**)

59145 **SYNOPSIS**

```
59146 MC1 #include <pthread.h>  
59147 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,  
59148 int protocol);
```

59149 **DESCRIPTION**

59150 Refer to [pthread\\_mutexattr\\_getprotocol\(\)](#).

59151 **NAME**

59152 pthread\_mutexattr\_setpshared — set the process-shared attribute

59153 **SYNOPSIS**

```
59154 TSH #include <pthread.h>
59155 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
59156 int pshared);
```

59157 **DESCRIPTION**

59158 Refer to [pthread\\_mutexattr\\_getpshared\(\)](#).

59159 **NAME**

59160 pthread\_mutexattr\_setrobust — get and set the mutex robust attribute

59161 **SYNOPSIS**

59162 #include &lt;pthread.h&gt;

59163 int pthread\_mutexattr\_setrobust(pthread\_mutexattr\_t \*attr,  
59164 int robust);59165 **DESCRIPTION**59166 Refer to [pthread\\_mutexattr\\_getrobust\(\)](#).

59167 **NAME**

59168 pthread\_mutexattr\_settype — set the mutex type attribute

59169 **SYNOPSIS**

59170 #include <pthread.h>

59171 int pthread\_mutexattr\_settype(pthread\_mutexattr\_t \*attr, int type);

59172 **DESCRIPTION**

59173 Refer to *pthread\_mutexattr\_gettype()*.



59174 **NAME**

59175 pthread\_once — dynamic package initialization

59176 **SYNOPSIS**

```
59177 #include <pthread.h>
59178 int pthread_once(pthread_once_t *once_control,
59179                 void (*init_routine)(void));
59180 pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

59181 **DESCRIPTION**

59182 The first call to *pthread\_once()* by any thread in a process, with a given *once\_control*, shall call the  
 59183 *init\_routine* with no arguments. Subsequent calls of *pthread\_once()* with the same *once\_control*  
 59184 shall not call the *init\_routine*. On return from *pthread\_once()*, *init\_routine* shall have completed.  
 59185 The *once\_control* parameter shall determine whether the associated initialization routine has been  
 59186 called.

59187 The *pthread\_once()* function is not a cancellation point. However, if *init\_routine* is a cancellation  
 59188 point and is canceled, the effect on *once\_control* shall be as if *pthread\_once()* was never called.

59189 If the call to *init\_routine* is terminated by a call to *longjmp()* or *siglongjmp()*, the behavior is  
 59190 undefined.

59191 The constant PTHREAD\_ONCE\_INIT is defined in the **<pthread.h>** header.

59192 The behavior of *pthread\_once()* is undefined if *once\_control* has automatic storage duration or is  
 59193 not initialized by PTHREAD\_ONCE\_INIT.

59194 **RETURN VALUE**

59195 Upon successful completion, *pthread\_once()* shall return zero; otherwise, an error number shall  
 59196 be returned to indicate the error.

59197 **ERRORS**

59198 The *pthread\_once()* function shall not return an error code of [EINTR].

59199 **EXAMPLES**

59200 None.

59201 **APPLICATION USAGE**

59202 If *init\_routine* recursively calls *pthread\_once()* with the same *once\_control*, the recursive call will  
 59203 not call the specified *init\_routine*, and thus the specified *init\_routine* will not complete, and thus  
 59204 the recursive call to *pthread\_once()* will not return. Use of *longjmp()* or *siglongjmp()* within an  
 59205 *init\_routine* to jump to a point outside of *init\_routine* prevents *init\_routine* from returning.

59206 **RATIONALE**

59207 Some C libraries are designed for dynamic initialization. That is, the global initialization for the  
 59208 library is performed when the first procedure in the library is called. In a single-threaded  
 59209 program, this is normally implemented using a static variable whose value is checked on entry  
 59210 to a routine, as follows:

```
59211 static int random_is_initialized = 0;
59212 extern void initialize_random(void);
59213 int random_function()
59214 {
59215     if (random_is_initialized == 0) {
59216         initialize_random();
59217         random_is_initialized = 1;
59218     }
```

```

59219     ... /* Operations performed after initialization. */
59220     }

```

59221 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,  
 59222 library initialization has to be accomplished by an explicit call to a library-exported initialization  
 59223 function prior to any use of the library.

59224 For dynamic library initialization in a multi-threaded process, if an initialization flag is used the  
 59225 flag needs to be protected against modification by multiple threads simultaneously calling into  
 59226 the library. This can be done by using a mutex (initialized by assigning  
 59227 PTHREAD\_MUTEX\_INITIALIZER). However, the better solution is to use *pthread\_once()* which  
 59228 is designed for exactly this purpose, as follows:

```

59229 #include <pthread.h>
59230 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
59231 extern void initialize_random(void);

59232 int random_function()
59233 {
59234     (void) pthread_once(&random_is_initialized, initialize_random);
59235     ... /* Operations performed after initialization. */
59236 }

```

59237 If an implementation detects that the value specified by the *once\_control* argument to  
 59238 *pthread\_once()* does not refer to a **pthread\_once\_t** object initialized by PTHREAD\_ONCE\_INIT,  
 59239 it is recommended that the function should fail and report an [EINVAL] error.

#### 59240 FUTURE DIRECTIONS

59241 None.

#### 59242 SEE ALSO

59243 XBD [<pthread.h>](#)

#### 59244 CHANGE HISTORY

59245 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

#### 59246 Issue 6

59247 The *pthread\_once()* function is marked as part of the Threads option.

59248 The [EINVAL] error is added as a “may fail” case for if either argument is invalid.

#### 59249 Issue 7

59250 The *pthread\_once()* function is moved from the Threads option to the Base.

59251 The [EINVAL] error for an uninitialized **pthread\_once\_t** object is removed; this condition results  
 59252 in undefined behavior.

59253 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0284 [863], XSH/TC2-2008/0285 [874],  
 59254 XSH/TC2-2008/0286 [874], and XSH/TC2-2008/0287 [747] are applied.

#### 59255 Issue 8

59256 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

59257 **NAME**

59258 pthread\_rwlock\_clockrdlock, pthread\_rwlock\_timedrdlock — lock a read-write lock for reading

59259 **SYNOPSIS**

```
59260 #include <pthread.h>
59261 int pthread_rwlock_clockrdlock(pthread_rwlock_t *restrict rlock,
59262     clockid_t clock_id, const struct timespec *restrict abstime);
59263 int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rlock,
59264     const struct timespec *restrict abstime);
```

59265 **DESCRIPTION**

59266 The *pthread\_rwlock\_clockrdlock()* and *pthread\_rwlock\_timedrdlock()* functions shall apply a read  
59267 lock to the read-write lock referenced by *rlock* as in the *pthread\_rwlock\_rdlock()* function.  
59268 However, if the lock cannot be acquired without waiting for other threads to unlock the lock,  
59269 this wait shall be terminated when the specified timeout expires. The timeout shall expire when  
59270 the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are  
59271 based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time  
59272 specified by *abstime* has already been passed at the time of the call.

59273 For *pthread\_rwlock\_timedrdlock()*, the timeout shall be based on the CLOCK\_REALTIME clock.  
59274 For *pthread\_rwlock\_clockrdlock()*, the timeout shall be based on the clock specified by the *clock\_id*  
59275 argument. The resolution of the timeout shall be the resolution of the clock on which it is based.  
59276 Implementations shall support passing CLOCK\_REALTIME and CLOCK\_MONOTONIC to  
59277 *pthread\_rwlock\_clockrdlock()* as the *clock\_id* argument.

59278 Under no circumstances shall the function fail with a timeout if the lock can be acquired  
59279 immediately. The validity of the *abstime* parameter need not be checked if the lock can be  
59280 immediately acquired.

59281 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
59282 write lock via a call to *pthread\_rwlock\_clockrdlock()* or *pthread\_rwlock\_timedrdlock()*, upon return  
59283 from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

59284 The calling thread may deadlock if at the time the call is made it holds a write lock on *rlock*.  
59285 The results are undefined if these functions are called with an uninitialized read-write lock.

59286 **RETURN VALUE**

59287 The *pthread\_rwlock\_clockrdlock()* and *pthread\_rwlock\_timedrdlock()* functions shall return zero if  
59288 the lock for reading on the read-write lock object referenced by *rlock* is acquired. Otherwise, an  
59289 error number shall be returned to indicate the error.

59290 **ERRORS**

59291 The *pthread\_rwlock\_clockrdlock()* and *pthread\_rwlock\_timedrdlock()* functions shall fail if:

59292 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

59293 The *pthread\_rwlock\_clockrdlock()* and *pthread\_rwlock\_timedrdlock()* functions may fail if:

59294 [EAGAIN] The read lock could not be acquired because the maximum number of read  
59295 locks for lock would be exceeded.

59296 [EDEADLK] A deadlock condition was detected or the calling thread already holds a write  
59297 lock on *rlock*.

59298 [EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000  
59299 million, or the *pthread\_rwlock\_clockrdlock()* function was passed an invalid or  
59300 unsupported *clock\_id* value.

59301 These functions shall not return an error code of [EINTR].

59302 **EXAMPLES**

59303 None.

59304 **APPLICATION USAGE**59305 Applications using these functions may be subject to priority inversion, as discussed in XBD  
59306 [Section 3.275](#) (on page 72).59307 **RATIONALE**59308 If an implementation detects that the value specified by the *rwlock* argument to  
59309 *pthread\_rwlock\_clockrdlock()* or *pthread\_rwlock\_timedrdlock()* does not refer to an initialized read-  
59310 write lock object, it is recommended that the function should fail and report an [EINVAL] error.59311 **FUTURE DIRECTIONS**

59312 None.

59313 **SEE ALSO**59314 [pthread\\_rwlock\\_clockwrlock\(\)](#), [pthread\\_rwlock\\_destroy\(\)](#), [pthread\\_rwlock\\_rdlock\(\)](#),  
59315 [pthread\\_rwlock\\_trywrlock\(\)](#), [pthread\\_rwlock\\_unlock\(\)](#)59316 XBD [Section 3.275](#) (on page 72), [Section 4.15.2](#) (on page 104), [<pthread.h>](#), [<time.h>](#)59317 **CHANGE HISTORY**

59318 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

59319 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/102 is applied, updating the ERRORS  
59320 section so that the [EDEADLK] error includes detection of a deadlock condition.59321 **Issue 7**59322 The *pthread\_rwlock\_timedrdlock()* function is moved from the Timeouts option to the Base.59323 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
59324 in undefined behavior.59325 **Issue 8**59326 Austin Group Defect 592 is applied, removing text relating to [<time.h>](#) from the SYNOPSIS and  
59327 DESCRIPTION sections.59328 Austin Group Defects 1216 and 1472 are applied, adding *pthread\_rwlock\_clockrdlock()*.

59329 **NAME**

59330 pthread\_rwlock\_clockwrlock, pthread\_rwlock\_timedwrlock — lock a read-write lock for writing

59331 **SYNOPSIS**

```
59332 #include <pthread.h>
59333 int pthread_rwlock_clockwrlock(pthread_rwlock_t *restrict rlock,
59334     clockid_t clock_id, const struct timespec *restrict abstime);
59335 int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rlock,
59336     const struct timespec *restrict abstime);
```

59337 **DESCRIPTION**

59338 The *pthread\_rwlock\_clockwrlock()* and *pthread\_rwlock\_timedwrlock()* functions shall apply a write  
 59339 lock to the read-write lock referenced by *rlock* as in the *pthread\_rwlock\_wrlock()* function.  
 59340 However, if the lock cannot be acquired without waiting for other threads to unlock the lock,  
 59341 this wait shall be terminated when the specified timeout expires. The timeout shall expire when  
 59342 the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are  
 59343 based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time  
 59344 specified by *abstime* has already been passed at the time of the call.

59345 For *pthread\_rwlock\_timedwrlock()*, the timeout shall be based on the CLOCK\_REALTIME clock.  
 59346 For *pthread\_rwlock\_clockwrlock()*, the timeout shall be based on the clock specified by the *clock\_id*  
 59347 argument. The resolution of the timeout shall be the resolution of the clock on which it is based.  
 59348 Implementations shall support passing CLOCK\_REALTIME and CLOCK\_MONOTONIC to  
 59349 *pthread\_rwlock\_clockwrlock()* as the *clock\_id* argument.

59350 Under no circumstances shall the function fail with a timeout if the lock can be acquired  
 59351 immediately. The validity of the *abstime* parameter need not be checked if the lock can be  
 59352 immediately acquired.

59353 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-  
 59354 write lock via a call to *pthread\_rwlock\_clockwrlock()* or *pthread\_rwlock\_timedwrlock()*, upon return  
 59355 from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

59356 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The  
 59357 results are undefined if these functions are called with an uninitialized read-write lock.

59358 **RETURN VALUE**

59359 The *pthread\_rwlock\_clockwrlock()* and *pthread\_rwlock\_timedwrlock()* functions shall return zero if  
 59360 the lock for writing on the read-write lock object referenced by *rlock* is acquired. Otherwise, an  
 59361 error number shall be returned to indicate the error.

59362 **ERRORS**

59363 The *pthread\_rwlock\_clockwrlock()* and *pthread\_rwlock\_timedwrlock()* functions shall fail if:

59364 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

59365 The *pthread\_rwlock\_clockwrlock()* and *pthread\_rwlock\_timedwrlock()* functions may fail if:

59366 [EDEADLK] A deadlock condition was detected or the calling thread already holds the  
 59367 *rlock*.

59368 [EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000  
 59369 million, or the *pthread\_rwlock\_clockwrlock()* function was passed an invalid or  
 59370 unsupported *clock\_id* value.

59371 These functions shall not return an error code of [EINTR].

59372 **EXAMPLES**

59373 None.

59374 **APPLICATION USAGE**59375 Applications using these functions may be subject to priority inversion, as discussed in XBD  
59376 [Section 3.275](#) (on page 72).59377 **RATIONALE**59378 If an implementation detects that the value specified by the *rwlock* argument to  
59379 *pthread\_rwlock\_clockwrlock()* or *pthread\_rwlock\_timedwrlock()* does not refer to an initialized read-  
59380 write lock object, it is recommended that the function should fail and report an [EINVAL] error.59381 **FUTURE DIRECTIONS**

59382 None.

59383 **SEE ALSO**59384 [pthread\\_rwlock\\_clockrdlock\(\)](#), [pthread\\_rwlock\\_destroy\(\)](#), [pthread\\_rwlock\\_rdlock\(\)](#),  
59385 [pthread\\_rwlock\\_trywrlock\(\)](#), [pthread\\_rwlock\\_unlock\(\)](#)59386 XBD [Section 3.275](#) (on page 72), [Section 4.15.2](#) (on page 104), [<pthread.h>](#), [<time.h>](#)59387 **CHANGE HISTORY**

59388 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

59389 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/103 is applied, updating the ERRORS  
59390 section so that the [EDEADLK] error includes detection of a deadlock condition.59391 **Issue 7**59392 The *pthread\_rwlock\_timedwrlock()* function is moved from the Timeouts option to the Base.59393 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
59394 in undefined behavior.59395 **Issue 8**59396 Austin Group Defect 592 is applied, removing text relating to [<time.h>](#) from the SYNOPSIS and  
59397 DESCRIPTION sections.59398 Austin Group Defects 1216 and 1472 are applied, adding *pthread\_rwlock\_clockwrlock()*.

59399 **NAME**

59400 pthread\_rwlock\_destroy, pthread\_rwlock\_init — destroy and initialize a read-write lock object

59401 **SYNOPSIS**

```
59402 #include <pthread.h>
59403 int pthread_rwlock_destroy(pthread_rwlock_t *rwlck);
59404 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlck,
59405     const pthread_rwlockattr_t *restrict attr);
59406 pthread_rwlock_t rwlck = PTHREAD_RWLOCK_INITIALIZER;
```

59407 **DESCRIPTION**

59408 The *pthread\_rwlock\_destroy()* function shall destroy the read-write lock object referenced by  
 59409 *rwlck* and release any resources used by the lock. The effect of subsequent use of the lock is  
 59410 undefined until the lock is reinitialized by another call to *pthread\_rwlock\_init()*. An  
 59411 implementation may cause *pthread\_rwlock\_destroy()* to set the object referenced by *rwlck* to an  
 59412 invalid value. Results are undefined if *pthread\_rwlock\_destroy()* is called when any thread holds  
 59413 *rwlck*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

59414 The *pthread\_rwlock\_init()* function shall allocate any resources required to use the read-write lock  
 59415 referenced by *rwlck* and initializes the lock to an unlocked state with attributes referenced by  
 59416 *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same as  
 59417 passing the address of a default read-write lock attributes object. Once initialized, the lock can be  
 59418 used any number of times without being reinitialized. Results are undefined if  
 59419 *pthread\_rwlock\_init()* is called specifying an already initialized read-write lock. Results are  
 59420 undefined if a read-write lock is used without first being initialized.

59421 If the *pthread\_rwlock\_init()* function fails, *rwlck* shall not be initialized and the contents of *rwlck*  
 59422 are undefined.

59423 See [Section 2.9.9](#) (on page 548) for further requirements.

59424 In cases where default read-write lock attributes are appropriate, the macro  
 59425 PTHREAD\_RWLOCK\_INITIALIZER can be used to initialize read-write locks. The effect shall  
 59426 be equivalent to dynamic initialization by a call to *pthread\_rwlock\_init()* with the *attr* parameter  
 59427 specified as NULL, except that no error checks are performed.

59428 The behavior is undefined if the value specified by the *attr* argument to *pthread\_rwlock\_init()*  
 59429 does not refer to an initialized read-write lock attributes object.

59430 **RETURN VALUE**

59431 If successful, the *pthread\_rwlock\_destroy()* and *pthread\_rwlock\_init()* functions shall return zero;  
 59432 otherwise, an error number shall be returned to indicate the error.

59433 **ERRORS**

59434 The *pthread\_rwlock\_init()* function shall fail if:

59435 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize  
 59436 another read-write lock.

59437 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

59438 [EPERM] The caller does not have the privilege to perform the operation.

59439 These functions shall not return an error code of [EINTR].

59440 **EXAMPLES**

59441 None.

59442 **APPLICATION USAGE**59443 Applications using these and related read-write lock functions may be subject to priority  
59444 inversion, as discussed in XBD [Section 3.275](#) (on page 72).59445 **RATIONALE**59446 If an implementation detects that the value specified by the *rwlock* argument to  
59447 *pthread\_rwlock\_destroy()* does not refer to an initialized read-write lock object, it is recommended  
59448 that the function should fail and report an [EINVAL] error.59449 If an implementation detects that the value specified by the *attr* argument to  
59450 *pthread\_rwlock\_init()* does not refer to an initialized read-write lock attributes object, it is  
59451 recommended that the function should fail and report an [EINVAL] error.59452 If an implementation detects that the value specified by the *rwlock* argument to  
59453 *pthread\_rwlock\_destroy()* or *pthread\_rwlock\_init()* refers to a locked read-write lock object, or  
59454 detects that the value specified by the *rwlock* argument to *pthread\_rwlock\_init()* refers to an  
59455 already initialized read-write lock object, it is recommended that the function should fail and  
59456 report an [EBUSY] error.59457 **FUTURE DIRECTIONS**

59458 None.

59459 **SEE ALSO**59460 [pthread\\_rwlock\\_clockrdlock\(\)](#), [pthread\\_rwlock\\_clockwrlock\(\)](#), [pthread\\_rwlock\\_rdlock\(\)](#),  
59461 [pthread\\_rwlock\\_trywrlock\(\)](#), [pthread\\_rwlock\\_unlock\(\)](#)59462 [XBD Section 3.275](#) (on page 72), [<pthread.h>](#)59463 **CHANGE HISTORY**

59464 First released in Issue 5.

59465 **Issue 6**

59466 The following changes are made for alignment with IEEE Std 1003.1j-2000:

59467 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
59468 now part of the Threads option (previously it was part of the Read-Write Locks option in  
59469 IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also  
59470 deleted from the SYNOPSIS.

59471 • The DESCRIPTION is updated as follows:

59472 — It explicitly notes allocation of resources upon initialization of a read-write lock  
59473 object.59474 — A paragraph is added specifying that copies of read-write lock objects may not be  
59475 used.59476 • An [EINVAL] error is added to the ERRORS section for *pthread\_rwlock\_init()*, indicating  
59477 that the *rwlock* value is invalid.

59478 • The SEE ALSO section is updated.

59479 The **restrict** keyword is added to the *pthread\_rwlock\_init()* prototype for alignment with the  
59480 ISO/IEC 9899:1999 standard.59481 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION  
59482 USAGE relating to priority inversion.



59483 **Issue 7**

59484 Austin Group Interpretation 1003.1-2001 #048 is applied, adding the  
59485 PTHREAD\_RWLOCK\_INITIALIZER macro.

59486 The *pthread\_rwlock\_destroy()* and *pthread\_rwlock\_init()* functions are moved from the Threads  
59487 option to the Base.

59488 The [EINVAL] error for an uninitialized read-write lock object or read-write lock attributes  
59489 object is removed; this condition results in undefined behavior.

59490 The [EBUSY] error for a locked read-write lock object or an already initialized read-write lock  
59491 object is removed; this condition results in undefined behavior.

59492 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0465 [70] is applied.

59493 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0288 [972] and XSH/TC2-2008/0289  
59494 [758] are applied.

59495 **NAME**

59496 pthread\_rwlock\_rdlock, pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

59497 **SYNOPSIS**

59498 #include &lt;pthread.h&gt;

59499 int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);

59500 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

59501 **DESCRIPTION**59502 The *pthread\_rwlock\_rdlock()* function shall apply a read lock to the read-write lock referenced by  
59503 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are  
59504 no writers blocked on the lock.59505 TPS If the Thread Execution Scheduling option is supported, and the threads that hold or are blocked  
59506 on the lock are executing with the scheduling policies SCHED\_FIFO or SCHED\_RR, the calling  
59507 thread shall not acquire the lock if a writer holds the lock or if the calling thread does not  
59508 already hold a read lock and writers of higher or equal priority are blocked on the lock;  
59509 otherwise, the calling thread shall acquire the lock.59510 TPS TSP If the Thread Execution Scheduling option is supported, and the threads that hold or are blocked  
59511 on the lock are executing with the SCHED\_SPORADIC scheduling policy, the calling thread  
59512 shall not acquire the lock if a writer holds the lock or if the calling thread does not already hold a  
59513 read lock and writers of higher or equal priority are blocked on the lock; otherwise, the calling  
59514 thread shall acquire the lock.59515 If the Thread Execution Scheduling option is not supported, it is implementation-defined  
59516 whether the calling thread acquires the lock when a writer does not hold the lock and there are  
59517 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the  
59518 read lock.59519 Whether the Thread Execution Scheduling option is supported or not, if the read lock is not  
59520 acquired, the calling thread shall block until it can acquire the lock. The calling thread may  
59521 deadlock if at the time the call is made it holds a write lock.59522 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the  
59523 *pthread\_rwlock\_rdlock()* function *n* times). If so, the application shall ensure that the thread  
59524 performs matching unlocks (that is, it calls the *pthread\_rwlock\_unlock()* function *n* times).59525 The maximum number of simultaneous read locks that an implementation guarantees can be  
59526 applied to a read-write lock shall be implementation-defined. The *pthread\_rwlock\_rdlock()*  
59527 function may fail if this maximum would be exceeded.59528 The *pthread\_rwlock\_tryrdlock()* function shall apply a read lock as in the *pthread\_rwlock\_rdlock()*  
59529 function, with the exception that the function shall fail if the equivalent *pthread\_rwlock\_rdlock()*  
59530 call would have blocked the calling thread. In no case shall the *pthread\_rwlock\_tryrdlock()*  
59531 function ever block; it always either acquires the lock or fails and returns immediately.

59532 Results are undefined if any of these functions are called with an uninitialized read-write lock.

59533 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the  
59534 signal handler the thread resumes waiting for the read-write lock for reading as if it was not  
59535 interrupted.59536 **RETURN VALUE**59537 If successful, the *pthread\_rwlock\_rdlock()* function shall return zero; otherwise, an error number  
59538 shall be returned to indicate the error.59539 The *pthread\_rwlock\_tryrdlock()* function shall return zero if the lock for reading on the read-write

59540 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to  
59541 indicate the error.

#### 59542 ERRORS

59543 The *pthread\_rwlock\_tryrdlock()* function shall fail if:

59544 [EBUSY] The read-write lock could not be acquired for reading because a writer holds  
59545 the lock or a writer with the appropriate priority was blocked on it.

59546 The *pthread\_rwlock\_rdlock()* and *pthread\_rwlock\_tryrdlock()* functions may fail if:

59547 [EAGAIN] The read lock could not be acquired because the maximum number of read  
59548 locks for *rwlock* has been exceeded.

59549 The *pthread\_rwlock\_rdlock()* function may fail if:

59550 [EDEADLK] A deadlock condition was detected or the current thread already owns the  
59551 read-write lock for writing.

59552 These functions shall not return an error code of [EINTR].

#### 59553 EXAMPLES

59554 None.

#### 59555 APPLICATION USAGE

59556 Applications using these functions may be subject to priority inversion, as discussed in XBD  
59557 [Section 3.275](#) (on page 72).

#### 59558 RATIONALE

59559 If an implementation detects that the value specified by the *rwlock* argument to  
59560 *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_tryrdlock()* does not refer to an initialized read-write  
59561 lock object, it is recommended that the function should fail and report an [EINVAL] error.

#### 59562 FUTURE DIRECTIONS

59563 None.

#### 59564 SEE ALSO

59565 [pthread\\_rwlock\\_clockrdlock\(\)](#), [pthread\\_rwlock\\_clockwrlock\(\)](#), [pthread\\_rwlock\\_destroy\(\)](#),  
59566 [pthread\\_rwlock\\_trywrlock\(\)](#), [pthread\\_rwlock\\_unlock\(\)](#)

59567 XBD [Section 3.275](#) (on page 72), [Section 4.15.2](#) (on page 104), [<pthread.h>](#)

#### 59568 CHANGE HISTORY

59569 First released in Issue 5.

#### 59570 Issue 6

59571 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 59572 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
59573 now part of the Threads option (previously it was part of the Read-Write Locks option in  
59574 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 59575 • The DESCRIPTION is updated as follows:
  - 59576 — Conditions under which writers have precedence over readers are specified.
  - 59577 — Failure of *pthread\_rwlock\_tryrdlock()* is clarified.
  - 59578 — A paragraph on the maximum number of read locks is added.

59579                   • In the ERRORS sections, [EBUSY] is modified to take into account write priority, and  
59580                   [EDEADLK] is deleted as a *pthread\_rwlock\_tryrdlock()* error.

59581                   • The SEE ALSO section is updated.

59582                   IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/101 is applied, updating the ERRORS  
59583                   section so that the [EDEADLK] error includes detection of a deadlock condition.

59584   **Issue 7**

59585                   The *pthread\_rwlock\_rdlock()* and *pthread\_rwlock\_tryrdlock()* functions are moved from the Threads  
59586                   option to the Base.

59587                   The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
59588                   in undefined behavior.

59589   **Issue 8**

59590                   Austin Group Defect 1111 is applied, changing “threads involved in the lock” to “threads that  
59591                   hold or are blocked on the lock” and clarifying that a requirement when the read lock is not  
59592                   acquired applies regardless of whether or not the Thread Execution Scheduling option is  
59593                   supported.

59594 **NAME**

59595 pthread\_rwlock\_timedrdlock — lock a read-write lock for reading

59596 **SYNOPSIS**

59597 #include &lt;pthread.h&gt;

59598 int pthread\_rwlock\_timedrdlock(pthread\_rwlock\_t \*restrict *rwlock*,  
59599 const struct timespec \*restrict *abstime*);59600 **DESCRIPTION**59601 Refer to [pthread\\_rwlock\\_clockrdlock\(\)](#).

59602 **NAME**

59603 pthread\_rwlock\_timedwrlock — lock a read-write lock for writing

59604 **SYNOPSIS**

59605 #include <pthread.h>

59606 int pthread\_rwlock\_timedwrlock(pthread\_rwlock\_t \*restrict *rwlock*,  
59607 const struct timespec \*restrict *abstime*);

59608 **DESCRIPTION**

59609 Refer to [pthread\\_rwlock\\_clockwrlock\(\)](#).

59610 **NAME**

59611 pthread\_rwlock\_tryrdlock — lock a read-write lock object for reading

59612 **SYNOPSIS**

59613 #include &lt;pthread.h&gt;

59614 int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

59615 **DESCRIPTION**59616 Refer to *pthread\_rwlock\_rdlock()*.

59617 **NAME**

59618 pthread\_rwlock\_trywrlock, pthread\_rwlock\_wrlock — lock a read-write lock object for writing

59619 **SYNOPSIS**

59620 #include <pthread.h>

59621 int pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*rlock);

59622 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rlock);

59623 **DESCRIPTION**

59624 The *pthread\_rwlock\_trywrlock()* function shall apply a write lock like the *pthread\_rwlock\_wrlock()*  
59625 function, with the exception that the function shall fail if any thread currently holds *rlock* (for  
59626 reading or writing).

59627 The *pthread\_rwlock\_wrlock()* function shall apply a write lock to the read-write lock referenced by  
59628 *rlock*. The calling thread shall acquire the write lock if no thread (reader or writer) holds the  
59629 read-write lock *rlock*. Otherwise, if another thread holds the read-write lock *rlock*, the calling  
59630 thread shall block until it can acquire the lock. If a deadlock condition occurs or the calling  
59631 thread already owns the read-write lock for writing or reading, the call shall either deadlock or  
59632 return [EDEADLK].

59633 Results are undefined if any of these functions are called with an uninitialized read-write lock.

59634 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the  
59635 signal handler the thread resumes waiting for the read-write lock for writing as if it was not  
59636 interrupted.

59637 **RETURN VALUE**

59638 The *pthread\_rwlock\_trywrlock()* function shall return zero if the lock for writing on the read-write  
59639 lock object referenced by *rlock* is acquired. Otherwise, an error number shall be returned to  
59640 indicate the error.

59641 If successful, the *pthread\_rwlock\_wrlock()* function shall return zero; otherwise, an error number  
59642 shall be returned to indicate the error.

59643 **ERRORS**

59644 The *pthread\_rwlock\_trywrlock()* function shall fail if:

59645 [EBUSY] The read-write lock could not be acquired for writing because it was already  
59646 locked for reading or writing.

59647 The *pthread\_rwlock\_wrlock()* function may fail if:

59648 [EDEADLK] A deadlock condition was detected or the current thread already owns the  
59649 read-write lock for writing or reading.

59650 These functions shall not return an error code of [EINTR].

59651 **EXAMPLES**

59652 None.

59653 **APPLICATION USAGE**

59654 Applications using these functions may be subject to priority inversion, as discussed in XBD  
59655 [Section 3.275](#) (on page 72).

59656 **RATIONALE**

59657 If an implementation detects that the value specified by the *rlock* argument to  
59658 *pthread\_rwlock\_trywrlock()* or *pthread\_rwlock\_wrlock()* does not refer to an initialized read-write  
59659 lock object, it is recommended that the function should fail and report an [EINVAL] error.



59660 **FUTURE DIRECTIONS**

59661 None.

59662 **SEE ALSO**59663 *pthread\_rwlock\_clockrdlock()*, *pthread\_rwlock\_clockwrlock()*, *pthread\_rwlock\_destroy()*,  
59664 *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_unlock()*

59665 XBD Section 3.275 (on page 72), Section 4.15.2 (on page 104), &lt;pthread.h&gt;

59666 **CHANGE HISTORY**

59667 First released in Issue 5.

59668 **Issue 6**

59669 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 59670
- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
59671 now part of the Threads option (previously it was part of the Read-Write Locks option in  
59672 IEEE Std 1003.1j-2000 and also part of the XSI extension).
  - The [EDEADLK] error is deleted as a *pthread\_rwlock\_trywrlock()* error.  
59673
  - The SEE ALSO section is updated.  
59674

59675 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/104 is applied, updating the ERRORS  
59676 section so that the [EDEADLK] error includes detection of a deadlock condition.59677 **Issue 7**59678 The *pthread\_rwlock\_trywrlock()* and *pthread\_rwlock\_wrlock()* functions are moved from the  
59679 Threads option to the Base.59680 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
59681 in undefined behavior.59682 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0290 [720] and XSH/TC2-2008/0291  
59683 [722] are applied.

59684 **NAME**

59685 pthread\_rwlock\_unlock — unlock a read-write lock object

59686 **SYNOPSIS**

59687 #include &lt;pthread.h&gt;

59688 int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rlock);

59689 **DESCRIPTION**

59690 The *pthread\_rwlock\_unlock()* function shall release a lock held on the read-write lock object  
59691 referenced by *rlock*. Results are undefined if the read-write lock *rlock* is not held by the  
59692 calling thread.

59693 If this function is called to release a read lock from the read-write lock object and there are other  
59694 read locks currently held on this read-write lock object, the read-write lock object remains in the  
59695 read locked state. If this function releases the last read lock for this read-write lock object, the  
59696 read-write lock object shall be put in the unlocked state with no owners.

59697 If this function is called to release a write lock for this read-write lock object, the read-write lock  
59698 object shall be put in the unlocked state.

59699 If there are threads blocked on the lock when it becomes available, the scheduling policy shall  
59700 TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is  
59701 supported, when threads executing with the scheduling policies SCHED\_FIFO, SCHED\_RR, or  
59702 SCHED\_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when  
59703 the lock becomes available. For equal priority threads, write locks shall take precedence over  
59704 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-  
59705 defined whether write locks take precedence over read locks.

59706 Results are undefined if this function is called with an uninitialized read-write lock.

59707 **RETURN VALUE**

59708 If successful, the *pthread\_rwlock\_unlock()* function shall return zero; otherwise, an error number  
59709 shall be returned to indicate the error.

59710 **ERRORS**

59711 The *pthread\_rwlock\_unlock()* function shall not return an error code of [EINTR].

59712 **EXAMPLES**

59713 None.

59714 **APPLICATION USAGE**

59715 None.

59716 **RATIONALE**

59717 If an implementation detects that the value specified by the *rlock* argument to  
59718 *pthread\_rwlock\_unlock()* does not refer to an initialized read-write lock object, it is recommended  
59719 that the function should fail and report an [EINVAL] error.

59720 If an implementation detects that the value specified by the *rlock* argument to  
59721 *pthread\_rwlock\_unlock()* refers to a read-write lock object for which the current thread does not  
59722 hold a lock, it is recommended that the function should fail and report an [EPERM] error.

59723 **FUTURE DIRECTIONS**

59724 None.

59725 **SEE ALSO**

59726 *pthread\_rwlock\_clockrdlock()*, *pthread\_rwlock\_clockwrlock()*, *pthread\_rwlock\_destroy()*,  
59727 *pthread\_rwlock\_rdlock()*, *pthread\_rwlock\_trywrlock()*

59728 XBD Section 4.15.2 (on page 104), **<pthread.h>**

59729 **CHANGE HISTORY**

59730 First released in Issue 5.

59731 **Issue 6**

59732 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 59733 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is  
59734 now part of the Threads option (previously it was part of the Read-Write Locks option in  
59735 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 59736 • The DESCRIPTION is updated as follows:
  - 59737 — The conditions under which writers have precedence over readers are specified.
  - 59738 — The concept of read-write lock owner is deleted.
- 59739 • The SEE ALSO section is updated.

59740 **Issue 7**

59741 SD5-XSH-ERN-183 is applied.

59742 The *pthread\_rwlock\_unlock()* function is moved from the Threads option to the Base.

59743 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results  
59744 in undefined behavior.

59745 The [EPERM] error for a read-write lock object for which the current thread does not hold a lock  
59746 is removed; this condition results in undefined behavior.

59747 **NAME**

59748 pthread\_rwlock\_wrlock — lock a read-write lock object for writing

59749 **SYNOPSIS**

59750 #include <pthread.h>

59751 int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock);

59752 **DESCRIPTION**

59753 Refer to *pthread\_rwlock\_trywrlock()*.

59754 **NAME**

59755 pthread\_rwlockattr\_destroy, pthread\_rwlockattr\_init — destroy and initialize the read-write  
59756 lock attributes object

59757 **SYNOPSIS**

```
59758 #include <pthread.h>
59759
59759 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
59760 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

59761 **DESCRIPTION**

59762 The *pthread\_rwlockattr\_destroy()* function shall destroy a read-write lock attributes object. A  
59763 destroyed *attr* attributes object can be reinitialized using *pthread\_rwlockattr\_init()*; the results of  
59764 otherwise referencing the object after it has been destroyed are undefined. An implementation  
59765 may cause *pthread\_rwlockattr\_destroy()* to set the object referenced by *attr* to an invalid value.

59766 The *pthread\_rwlockattr\_init()* function shall initialize a read-write lock attributes object *attr* with  
59767 the default value for all of the attributes defined by the implementation.

59768 Results are undefined if *pthread\_rwlockattr\_init()* is called specifying an already initialized *attr*  
59769 attributes object.

59770 After a read-write lock attributes object has been used to initialize one or more read-write locks,  
59771 any function affecting the attributes object (including destruction) shall not affect any previously  
59772 initialized read-write locks.

59773 The behavior is undefined if the value specified by the *attr* argument to  
59774 *pthread\_rwlockattr\_destroy()* does not refer to an initialized read-write lock attributes object.

59775 **RETURN VALUE**

59776 If successful, the *pthread\_rwlockattr\_destroy()* and *pthread\_rwlockattr\_init()* functions shall return  
59777 zero; otherwise, an error number shall be returned to indicate the error.

59778 **ERRORS**

59779 The *pthread\_rwlockattr\_init()* function shall fail if:

59780 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

59781 These functions shall not return an error code of [EINTR].

59782 **EXAMPLES**

59783 None.

59784 **APPLICATION USAGE**

59785 None.

59786 **RATIONALE**

59787 If an implementation detects that the value specified by the *attr* argument to  
59788 *pthread\_rwlockattr\_destroy()* does not refer to an initialized read-write lock attributes object, it is  
59789 recommended that the function should fail and report an [EINVAL] error.

59790 **FUTURE DIRECTIONS**

59791 None.

59792 **SEE ALSO**

59793 [pthread\\_rwlock\\_destroy\(\)](#), [pthread\\_rwlockattr\\_getpshared\(\)](#)

59794 XBD [<pthread.h>](#)

59795 **CHANGE HISTORY**

59796 First released in Issue 5.

59797 **Issue 6**

59798 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 59799 • The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
- 59800 now part of the Threads option (previously it was part of the Read-Write Locks option in
- 59801 IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 59802 • The SEE ALSO section is updated.

59803 **Issue 7**59804 The *pthread\_rwlockattr\_destroy()* and *pthread\_rwlockattr\_init()* functions are moved from the

59805 Threads option to the Base.

59806 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this

59807 condition results in undefined behavior.

59808 **NAME**

59809 pthread\_rwlockattr\_getpshared, pthread\_rwlockattr\_setpshared — get and set the process-  
 59810 shared attribute of the read-write lock attributes object

59811 **SYNOPSIS**

```
59812 TSH #include <pthread.h>
59813
59814 int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
59815 *restrict attr, int *restrict pshared);
59816 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
59817 int pshared);
```

59817 **DESCRIPTION**

59818 The *pthread\_rwlockattr\_getpshared()* function shall obtain the value of the *process-shared* attribute  
 59819 from the initialized attributes object referenced by *attr*. The *pthread\_rwlockattr\_setpshared()*  
 59820 function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

59821 The *process-shared* attribute shall be set to `PTHREAD_PROCESS_SHARED` to permit a read-write  
 59822 lock to be operated upon by any thread that has access to the memory where the read-write lock  
 59823 is allocated, even if the read-write lock is allocated in memory that is shared by multiple  
 59824 processes. See [Section 2.9.9](#) (on page 548) for further requirements. The default value of the  
 59825 *process-shared* attribute shall be `PTHREAD_PROCESS_PRIVATE`.

59826 Additional attributes, their default values, and the names of the associated functions to get and  
 59827 set those attribute values are implementation-defined.

59828 The behavior is undefined if the value specified by the *attr* argument to  
 59829 *pthread\_rwlockattr\_getpshared()* or *pthread\_rwlockattr\_setpshared()* does not refer to an initialized  
 59830 read-write lock attributes object.

59831 **RETURN VALUE**

59832 Upon successful completion, the *pthread\_rwlockattr\_getpshared()* function shall return zero and  
 59833 store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared*  
 59834 parameter. Otherwise, an error number shall be returned to indicate the error.

59835 If successful, the *pthread\_rwlockattr\_setpshared()* function shall return zero; otherwise, an error  
 59836 number shall be returned to indicate the error.

59837 **ERRORS**

59838 The *pthread\_rwlockattr\_setpshared()* function may fail if:

59839 [EINVAL] The new value specified for the attribute is outside the range of legal values  
 59840 for that attribute.

59841 These functions shall not return an error code of [EINTR].

59842 **EXAMPLES**

59843 None.

59844 **APPLICATION USAGE**

59845 None.

59846 **RATIONALE**

59847 None.

59848 **FUTURE DIRECTIONS**

59849 None.

59850 **SEE ALSO**59851 *pthread\_rwlock\_destroy()*, *pthread\_rwlockattr\_destroy()*

59852 XBD &lt;pthread.h&gt;

59853 **CHANGE HISTORY**

59854 First released in Issue 5.

59855 **Issue 6**

59856 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 59857 • The margin code in the SYNOPSIS is changed to THR TSH to indicate that the
- 59858 functionality is now part of the Threads option (previously it was part of the Read-Write
- 59859 Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 59860 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 59861 • The SEE ALSO section is updated.

59862 The **restrict** keyword is added to the *pthread\_rwlockattr\_getpshared()* prototype for alignment  
59863 with the ISO/IEC 9899:1999 standard.

59864 **Issue 7**

59865 The *pthread\_rwlockattr\_getpshared()* and *pthread\_rwlockattr\_setpshared()* functions are marked  
59866 only as part of the Thread Process-Shared Synchronization option as the Threads option is now  
59867 part of the Base.

59868 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this  
59869 condition results in undefined behavior.

59870 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0292 [972] and XSH/TC2-2008/0293  
59871 [757] are applied.



59872 **NAME**

59873 pthread\_rwlockattr\_init — initialize the read-write lock attributes object

59874 **SYNOPSIS**

59875 #include <pthread.h>

59876 int pthread\_rwlockattr\_init(pthread\_rwlockattr\_t \*attr);

59877 **DESCRIPTION**

59878 Refer to *pthread\_rwlockattr\_destroy()*.

59879 **NAME**

59880 pthread\_rwlockattr\_setpshared — set the process-shared attribute of the read-write lock  
59881 attributes object

59882 **SYNOPSIS**

```
59883 TSH #include <pthread.h>  
59884 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,  
59885 int pshared);
```

59886 **DESCRIPTION**

59887 Refer to [pthread\\_rwlockattr\\_getpshared\(\)](#).

59888 **NAME**

59889 pthread\_self — get the calling thread ID

59890 **SYNOPSIS**

59891 #include &lt;pthread.h&gt;

59892 pthread\_t pthread\_self(void);

59893 **DESCRIPTION**59894 The *pthread\_self()* function shall return the thread ID of the calling thread.59895 **RETURN VALUE**59896 The *pthread\_self()* function shall always be successful and no return value is reserved to indicate  
59897 an error.59898 **ERRORS**

59899 No errors are defined.

59900 **EXAMPLES**

59901 None.

59902 **APPLICATION USAGE**

59903 None.

59904 **RATIONALE**59905 The *pthread\_self()* function provides a capability similar to the *getpid()* function for processes  
59906 and the rationale is the same: the creation call does not provide the thread ID to the created  
59907 thread.59908 **FUTURE DIRECTIONS**

59909 None.

59910 **SEE ALSO**59911 *pthread\_create()*, *pthread\_equal()*

59912 XBD &lt;pthread.h&gt;

59913 **CHANGE HISTORY**

59914 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

59915 **Issue 6**59916 The *pthread\_self()* function is marked as part of the Threads option.59917 **Issue 7**

59918 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section.

59919 The *pthread\_self()* function is moved from the Threads option to the Base.

59920 **NAME**

59921 pthread\_setcancelstate, pthread\_setcanceltype, pthread\_testcancel — set cancelability state

59922 **SYNOPSIS**

```
59923 #include <pthread.h>
59924 int pthread_setcancelstate(int state, int *oldstate);
59925 int pthread_setcanceltype(int type, int *oldtype);
59926 void pthread_testcancel(void);
```

59927 **DESCRIPTION**

59928 The *pthread\_setcancelstate()* function shall atomically both set the calling thread's cancelability state to the indicated *state* and return the previous cancelability state at the location referenced by *oldstate*. Legal values for *state* are PTHREAD\_CANCEL\_ENABLE and PTHREAD\_CANCEL\_DISABLE.

59932 The *pthread\_setcanceltype()* function shall atomically both set the calling thread's cancelability type to the indicated *type* and return the previous cancelability type at the location referenced by *oldtype*. Legal values for *type* are PTHREAD\_CANCEL\_DEFERRED and PTHREAD\_CANCEL\_ASYNCCHRONOUS.

59936 The cancelability state and type of any newly created threads, including the thread in which *main()* was first invoked, shall be PTHREAD\_CANCEL\_ENABLE and PTHREAD\_CANCEL\_DEFERRED respectively.

59939 The *pthread\_testcancel()* function shall create a cancellation point in the calling thread. The *pthread\_testcancel()* function shall have no effect if cancelability is disabled.

59941 The *pthread\_setcancelstate()* function shall be async-signal-safe.

59942 **RETURN VALUE**

59943 If successful, the *pthread\_setcancelstate()* and *pthread\_setcanceltype()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

59945 **ERRORS**

59946 The *pthread\_setcancelstate()* function may fail if:

59947 [EINVAL] The specified state is not PTHREAD\_CANCEL\_ENABLE or  
59948 PTHREAD\_CANCEL\_DISABLE.

59949 The *pthread\_setcanceltype()* function may fail if:

59950 [EINVAL] The specified type is not PTHREAD\_CANCEL\_DEFERRED or  
59951 PTHREAD\_CANCEL\_ASYNCCHRONOUS.

59952 These functions shall not return an error code of [EINTR].

59953 **EXAMPLES**

59954 None.

59955 **APPLICATION USAGE**

59956 In order to write a signal handler for an asynchronous signal which can run safely in a cancellable thread, *pthread\_setcancelstate()* must be used to disable cancellation for the duration of any calls that the signal handler makes which are cancellation points. However, earlier versions of the standard did not permit strictly conforming applications to call *pthread\_setcancelstate()* from a signal handler since it was not required to be async-signal-safe. On non-conforming implementations where *pthread\_setcancelstate()* is not async-signal-safe, alternatives are to ensure either that the corresponding signals are blocked during execution of functions that are not async-cancel-safe or that cancellation is disabled during times when those signals could be delivered.

**59965 RATIONALE**

59966 The `pthread_setcancelstate()` and `pthread_setcanceltype()` functions control the points at which a  
59967 thread may be asynchronously canceled. For cancellation control to be usable in modular  
59968 fashion, some rules need to be followed.

59969 An object can be considered to be a generalization of a procedure. It is a set of procedures and  
59970 global variables written as a unit and called by clients not known by the object. Objects may  
59971 depend on other objects.

59972 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On  
59973 exit from an object, the cancelability state should always be restored to its value on entry to the  
59974 object.

59975 This follows from a modularity argument: if the client of an object (or the client of an object that  
59976 uses that object) has disabled cancelability, it is because the client does not want to be concerned  
59977 about cleaning up if the thread is canceled while executing some sequence of actions. If an object  
59978 is called in such a state and it enables cancelability and a cancellation request is pending for that  
59979 thread, then the thread is canceled, contrary to the wish of the client that disabled.

59980 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry  
59981 to an object. But as with the cancelability state, on exit from an object the cancelability type  
59982 should always be restored to its value on entry to the object.

59983 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously  
59984 cancelable.

**59985 FUTURE DIRECTIONS**

59986 None.

**59987 SEE ALSO**

59988 [\*pthread\\_cancel\(\)\*](#)

59989 XBD [\*\*<pthread.h>\*\*](#)

**59990 CHANGE HISTORY**

59991 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

**59992 Issue 6**

59993 The `pthread_setcancelstate()`, `pthread_setcanceltype()`, and `pthread_testcancel()` functions are marked  
59994 as part of the Threads option.

**59995 Issue 7**

59996 The `pthread_setcancelstate()`, `pthread_setcanceltype()`, and `pthread_testcancel()` functions are moved  
59997 from the Threads option to the Base.

59998 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0294 [622] and XSH/TC2-2008/0295  
59999 [615] are applied.

**60000 Issue 8**

60001 Austin Group Defect 841 is applied, requiring `pthread_setcancelstate()` to be async-signal-safe.

60002 **NAME**

60003 pthread\_setschedparam — dynamic thread scheduling parameters access (**REALTIME**  
60004 **THREADS**)

60005 **SYNOPSIS**

```
60006 TPS #include <pthread.h>  
60007 int pthread_setschedparam(pthread_t thread, int policy,  
60008     const struct sched_param *param);
```

60009 **DESCRIPTION**

60010 Refer to [pthread\\_getschedparam\(\)](#).

60011 **NAME**

60012 pthread\_setschedprio — dynamic thread scheduling parameters access (**REALTIME**  
60013 **THREADS**)

60014 **SYNOPSIS**

```
60015 TPS #include <pthread.h>  
60016 int pthread_setschedprio(pthread_t thread, int prio);
```

60017 **DESCRIPTION**

60018 The *pthread\_setschedprio()* function shall set the scheduling priority for the thread whose thread  
60019 ID is given by *thread* to the value given by *prio*. See [Scheduling Policies](#) (on page 531) for a  
60020 description on how this function call affects the ordering of the thread in the thread list for its  
60021 new priority.

60022 If the *pthread\_setschedprio()* function fails, the scheduling priority of the target thread shall not be  
60023 changed.

60024 **RETURN VALUE**

60025 If successful, the *pthread\_setschedprio()* function shall return zero; otherwise, an error number  
60026 shall be returned to indicate the error.

60027 **ERRORS**

60028 The *pthread\_setschedprio()* function may fail if:

60029 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.

60030 [EPERM] The caller does not have appropriate privileges to set the scheduling priority  
60031 of the specified thread.

60032 The *pthread\_setschedprio()* function shall not return an error code of [EINTR].

60033 **EXAMPLES**

60034 None.

60035 **APPLICATION USAGE**

60036 None.

60037 **RATIONALE**

60038 The *pthread\_setschedprio()* function provides a way for an application to temporarily raise its  
60039 priority and then lower it again, without having the undesired side-effect of yielding to other  
60040 threads of the same priority. This is necessary if the application is to implement its own  
60041 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This  
60042 capability is especially important if the implementation does not support the Thread Priority  
60043 Protection or Thread Priority Inheritance options, but even if those options are supported it is  
60044 needed if the application is to bound priority inheritance for other resources, such as  
60045 semaphores.

60046 The standard developers considered that while it might be preferable conceptually to solve this  
60047 problem by modifying the specification of *pthread\_setschedparam()*, it was too late to make such a  
60048 change, as there may be implementations that would need to be changed. Therefore, this new  
60049 function was introduced.

60050 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended  
60051 that the function should fail and report an [ESRCH] error.

60052 **FUTURE DIRECTIONS**

60053 None.

60054 **SEE ALSO**

60055 [Scheduling Policies](#) (on page 531), [pthread\\_getschedparam\(\)](#)

60056 XBD [<pthread.h>](#)

60057 **CHANGE HISTORY**

60058 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

60059 **Issue 7**

60060 The `pthread_setschedprio()` function is marked only as part of the Thread Execution Scheduling option as the Threads option is now part of the Base.

60062 Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [EPERM] error.

60063 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

60064 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0466 [314] is applied.

60065 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0296 [757] is applied.



60066 **NAME**

60067 pthread\_setspecific — thread-specific data management

60068 **SYNOPSIS**

60069 #include &lt;pthread.h&gt;

60070 int pthread\_setspecific(pthread\_key\_t *key*, const void \**value*);60071 **DESCRIPTION**60072 Refer to *pthread\_getspecific()*.

60073 **NAME**

60074 pthread\_sigmask, sigprocmask — examine and change blocked signals

60075 **SYNOPSIS**

```
60076 CX #include <signal.h>
60077 int pthread_sigmask(int how, const sigset_t *restrict set,
60078 sigset_t *restrict oset);
60079 int sigprocmask(int how, const sigset_t *restrict set,
60080 sigset_t *restrict oset);
```

60081 **DESCRIPTION**60082 The *pthread\_sigmask()* function shall examine or change (or both) the calling thread's signal  
60083 mask.60084 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the  
60085 currently blocked set.60086 The argument *how* indicates the way in which the set is changed, and the application shall  
60087 ensure it consists of one of the following values:60088 SIG\_BLOCK The resulting set shall be the union of the current set and the signal set  
60089 pointed to by *set*.60090 SIG\_SETMASK The resulting set shall be the signal set pointed to by *set*.60091 SIG\_UNBLOCK The resulting set shall be the intersection of the current set and the  
60092 complement of the signal set pointed to by *set*.60093 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location  
60094 pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the  
60095 thread's signal mask shall be unchanged; thus the call can be used to enquire about currently  
60096 blocked signals.60097 If the argument *set* is not a null pointer, after *pthread\_sigmask()* changes the currently blocked set  
60098 of signals it shall determine whether there are any pending unblocked signals; if there are any,  
60099 then at least one of those signals shall be delivered before the call to *pthread\_sigmask()* returns.60100 It is not possible to block those signals which cannot be ignored. This shall be enforced by the  
60101 system without causing an error to be indicated.60102 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked,  
60103 the result is undefined, unless the signal was generated by the action of another process, or by  
60104 one of the functions *kill()*, *pthread\_kill()*, *raise()*, or *sigqueue()*.60105 If *pthread\_sigmask()* fails, the thread's signal mask shall not be changed.60106 The *sigprocmask()* function shall be equivalent to *pthread\_sigmask()*, except that its behavior is  
60107 unspecified if called from a multi-threaded process, and on error it returns  $-1$  and sets *errno* to  
60108 the error number instead of returning the error number directly.60109 **RETURN VALUE**60110 Upon successful completion, *pthread\_sigmask()* shall return 0; otherwise, it shall return the  
60111 corresponding error number.60112 Upon successful completion, *sigprocmask()* shall return 0; otherwise,  $-1$  shall be returned and  
60113 *errno* shall be set to indicate the error.

60114 **ERRORS**

60115 These functions shall fail if:

60116 [EINVAL] The *set* argument is not a null pointer and the value of the *how* argument is not  
 60117 equal to one of the defined values.

60118 These functions shall not return an error code of [EINTR].

60119 **EXAMPLES**60120 **Signaling in a Multi-Threaded Process**

60121 This example shows the use of *pthread\_sigmask()* in order to deal with signals in a multi-  
 60122 threaded process. It provides a fairly general framework that could be easily adapted/extended.

```

60123 #include <stdio.h>
60124 #include <stdlib.h>
60125 #include <pthread.h>
60126 #include <signal.h>
60127 #include <string.h>
60128 #include <errno.h>
60129 ...
60130 static sigset_t  signal_mask; /* signals to block          */
60131 int main (int argc, char *argv[])
60132 {
60133     pthread_t  sig_thr_id;      /* signal handler thread ID */
60134     int        rc;              /* return code               */
60135     sigemptyset (&signal_mask);
60136     sigaddset (&signal_mask, SIGINT);
60137     sigaddset (&signal_mask, SIGTERM);
60138     rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
60139     if (rc != 0) {
60140         /* handle error */
60141         ...
60142     }
60143     /* any newly created threads inherit the signal mask */
60144     rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
60145     if (rc != 0) {
60146         /* handle error */
60147         ...
60148     }
60149     /* APPLICATION CODE */
60150     ...
60151 }
60152 void *signal_thread (void *arg)
60153 {
60154     int        sig_caught;      /* signal caught            */
60155     int        rc;              /* returned code            */
60156     rc = sigwait (&signal_mask, &sig_caught);
60157     if (rc != 0) {
60158         /* handle error */

```

```

60159     }
60160     switch (sig_caught)
60161     {
60162     case SIGINT:      /* process SIGINT */
60163         ...
60164         break;
60165     case SIGTERM:    /* process SIGTERM */
60166         ...
60167         break;
60168     default:         /* should normally not happen */
60169         fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
60170         break;
60171     }
60172 }

```

### 60173 APPLICATION USAGE

60174 Although *pthread\_sigmask()* has to deliver at least one of any pending unblocked signals that  
60175 exist after it has changed the currently blocked set of signals, there is no requirement that the  
60176 delivered signal(s) include any that were unblocked by the change. If one or more signals that  
60177 were already unblocked become pending (see [Section 2.4.1](#), on page 513) during the period the  
60178 *pthread\_setmask()* call is executing, the signal(s) delivered before the call returns might include  
60179 only those signals.

### 60180 RATIONALE

60181 When a thread's signal mask is changed in a signal-catching function that is installed by  
60182 *sigaction()*, the restoration of the signal mask on return from the signal-catching function  
60183 overrides that change (see *sigaction()*). If the signal-catching function was installed with  
60184 *signal()*, it is unspecified whether this occurs.

60185 See *kill()* for a discussion of the requirement on delivery of signals.

### 60186 FUTURE DIRECTIONS

60187 None.

### 60188 SEE ALSO

60189 *exec*, *kill()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,  
60190 *sigpending()*, *sigqueue()*, *sigsuspend()*

60191 XBD [<signal.h>](#)

### 60192 CHANGE HISTORY

60193 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 60194 Issue 5

60195 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

60196 The *pthread\_sigmask()* function is added for alignment with the POSIX Threads Extension.

#### 60197 Issue 6

60198 The *pthread\_sigmask()* function is marked as part of the Threads option.

60199 The SYNOPSIS for *sigprocmask()* is marked as a CX extension to note that the presence of this  
60200 function in the [<signal.h>](#) header is an extension to the ISO C standard.

60201 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

60202           • The DESCRIPTION is updated to explicitly state the functions which may generate the  
60203           signal.

60204           The normative text is updated to avoid use of the term “must” for application requirements.

60205           The **restrict** keyword is added to the *pthread\_sigmask()* and *sigprocmask()* prototypes for  
60206           alignment with the ISO/IEC 9899:1999 standard.

60207           IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/105 is applied, updating “process’ signal  
60208           mask” to “thread’s signal mask” in the DESCRIPTION and RATIONALE sections.

60209           IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/106 is applied, adding the example to the  
60210           EXAMPLES section.

60211   **Issue 7**

60212           The *pthread\_sigmask()* function is moved from the Threads option to the Base.

60213           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0467 [319] is applied.

60214   **Issue 8**

60215           Austin Group Defect 1132 is applied, clarifying the [EINVAL] error.

60216           Austin Group Defect 1636 is applied, clarifying the exceptions to the equivalence of  
60217           *pthread\_sigmask()* and *sigprocmask()*.

60218           Austin Group Defect 1731 is applied, clarifying that although *pthread\_sigmask()* has to deliver at  
60219           least one of any pending unblocked signals that exist after it has changed the currently blocked  
60220           set of signals, there is no requirement that the delivered signal(s) include any that were  
60221           unblocked by the change.

60222 **NAME**

60223 pthread\_spin\_destroy, pthread\_spin\_init — destroy or initialize a spin lock object

60224 **SYNOPSIS**

60225 #include &lt;pthread.h&gt;

60226 int pthread\_spin\_destroy(pthread\_spinlock\_t \*lock);

60227 int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

60228 **DESCRIPTION**

60229 The *pthread\_spin\_destroy()* function shall destroy the spin lock referenced by *lock* and release any  
 60230 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is  
 60231 reinitialized by another call to *pthread\_spin\_init()*. The results are undefined if  
 60232 *pthread\_spin\_destroy()* is called when a thread holds the lock, or if this function is called with an  
 60233 uninitialized thread spin lock.

60234 The *pthread\_spin\_init()* function shall allocate any resources required to use the spin lock  
 60235 referenced by *lock* and initialize the lock to an unlocked state.

60236 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is  
 60237 PTHREAD\_PROCESS\_SHARED, the implementation shall permit the spin lock to be operated  
 60238 upon by any thread that has access to the memory where the spin lock is allocated, even if it is  
 60239 allocated in memory that is shared by multiple processes.

60240 See [Section 2.9.9](#) (on page 548) for further requirements.

60241 The results are undefined if *pthread\_spin\_init()* is called specifying an already initialized spin  
 60242 lock. The results are undefined if a spin lock is used without first being initialized.

60243 If the *pthread\_spin\_init()* function fails, the lock is not initialized and the contents of *lock* are  
 60244 undefined.

60245 Only the object referenced by *lock* may be used for performing synchronization.

60246 The result of referring to copies of that object in calls to *pthread\_spin\_destroy()*,  
 60247 *pthread\_spin\_lock()*, *pthread\_spin\_trylock()*, or *pthread\_spin\_unlock()* is undefined.

60248 **RETURN VALUE**

60249 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
 60250 be returned to indicate the error.

60251 **ERRORS**

60252 The *pthread\_spin\_init()* function shall fail if:

60253 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

60254 [ENOMEM] Insufficient memory exists to initialize the lock.

60255 These functions shall not return an error code of [EINTR].

60256 **EXAMPLES**

60257 None.

60258 **APPLICATION USAGE**

60259 None.

60260 **RATIONALE**

60261 If an implementation detects that the value specified by the *lock* argument to  
 60262 *pthread\_spin\_destroy()* does not refer to an initialized spin lock object, it is recommended that the  
 60263 function should fail and report an [EINVAL] error.

60264 If an implementation detects that the value specified by the *lock* argument to

60265 *pthread\_spin\_destroy()* or *pthread\_spin\_init()* refers to a locked spin lock object, or detects that the  
60266 value specified by the *lock* argument to *pthread\_spin\_init()* refers to an already initialized spin  
60267 lock object, it is recommended that the function should fail and report an [EBUSY] error.

60268 **FUTURE DIRECTIONS**

60269 None.

60270 **SEE ALSO**

60271 *pthread\_spin\_lock()*, *pthread\_spin\_unlock()*

60272 XBD <pthread.h>

60273 **CHANGE HISTORY**

60274 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

60275 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

60276 **Issue 7**

60277 The *pthread\_spin\_destroy()* and *pthread\_spin\_init()* functions are moved from the Spin Locks  
60278 option to the Base.

60279 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in  
60280 undefined behavior.

60281 The [EBUSY] error for a locked spin lock object or an already initialized spin lock object is  
60282 removed; this condition results in undefined behavior.

60283 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0297 [972] is applied.

60284 **NAME**

60285 pthread\_spin\_lock, pthread\_spin\_trylock — lock a spin lock object

60286 **SYNOPSIS**

60287 #include &lt;pthread.h&gt;

60288 int pthread\_spin\_lock(pthread\_spinlock\_t \*lock);

60289 int pthread\_spin\_trylock(pthread\_spinlock\_t \*lock);

60290 **DESCRIPTION**

60291 The *pthread\_spin\_lock()* function shall lock the spin lock referenced by *lock*. The calling thread  
60292 shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is,  
60293 shall not return from the *pthread\_spin\_lock()* call) until the lock becomes available. The results are  
60294 undefined if the calling thread holds the lock at the time the call is made. The  
60295 *pthread\_spin\_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any  
60296 thread. Otherwise, the function shall fail.

60297 The results are undefined if any of these functions is called with an uninitialized spin lock.

60298 **RETURN VALUE**

60299 Upon successful completion, these functions shall return zero; otherwise, an error number shall  
60300 be returned to indicate the error.

60301 **ERRORS**

60302 The *pthread\_spin\_lock()* function may fail if:

60303 [EDEADLK] A deadlock condition was detected.

60304 The *pthread\_spin\_trylock()* function shall fail if:

60305 [EBUSY] A thread currently holds the lock.

60306 These functions shall not return an error code of [EINTR].

60307 **EXAMPLES**

60308 None.

60309 **APPLICATION USAGE**

60310 Applications using this function may be subject to priority inversion, as discussed in XBD  
60311 [Section 3.275](#) (on page 72).

60312 **RATIONALE**

60313 If an implementation detects that the value specified by the *lock* argument to *pthread\_spin\_lock()*  
60314 or *pthread\_spin\_trylock()* does not refer to an initialized spin lock object, it is recommended that  
60315 the function should fail and report an [EINVAL] error.

60316 If an implementation detects that the value specified by the *lock* argument to *pthread\_spin\_lock()*  
60317 refers to a spin lock object for which the calling thread already holds the lock, it is recommended  
60318 that the function should fail and report an [EDEADLK] error.

60319 **FUTURE DIRECTIONS**

60320 None.

60321 **SEE ALSO**

60322 [pthread\\_spin\\_destroy\(\)](#), [pthread\\_spin\\_unlock\(\)](#)

60323 XBD [Section 3.275](#) (on page 72), [Section 4.15.2](#) (on page 104), [<pthread.h>](#)



60324 **CHANGE HISTORY**

60325 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

60326 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

60327 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/107 is applied, updating the ERRORS  
60328 section so that the [EDEADLK] error includes detection of a deadlock condition.

60329 **Issue 7**

60330 The `pthread_spin_lock()` and `pthread_spin_trylock()` functions are moved from the Spin Locks  
60331 option to the Base.

60332 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in  
60333 undefined behavior.

60334 The [EDEADLK] error for a spin lock object for which the calling thread already holds the lock is  
60335 removed; this condition results in undefined behavior.

60336 **NAME**

60337 pthread\_spin\_unlock — unlock a spin lock object

60338 **SYNOPSIS**

60339 #include &lt;pthread.h&gt;

60340 int pthread\_spin\_unlock(pthread\_spinlock\_t \*lock);

60341 **DESCRIPTION**60342 The *pthread\_spin\_unlock()* function shall release the spin lock referenced by *lock* which was  
60343 locked via the *pthread\_spin\_lock()* or *pthread\_spin\_trylock()* functions.

60344 The results are undefined if the lock is not held by the calling thread.

60345 If there are threads spinning on the lock when *pthread\_spin\_unlock()* is called, the lock becomes  
60346 available and an unspecified spinning thread shall acquire the lock.

60347 The results are undefined if this function is called with an uninitialized thread spin lock.

60348 **RETURN VALUE**60349 Upon successful completion, the *pthread\_spin\_unlock()* function shall return zero; otherwise, an  
60350 error number shall be returned to indicate the error.60351 **ERRORS**

60352 This function shall not return an error code of [EINTR].

60353 **EXAMPLES**

60354 None.

60355 **APPLICATION USAGE**

60356 None.

60357 **RATIONALE**60358 If an implementation detects that the value specified by the *lock* argument to  
60359 *pthread\_spin\_unlock()* does not refer to an initialized spin lock object, it is recommended that the  
60360 function should fail and report an [EINVAL] error.60361 If an implementation detects that the value specified by the *lock* argument to  
60362 *pthread\_spin\_unlock()* refers to a spin lock object for which the current thread does not hold the  
60363 lock, it is recommended that the function should fail and report an [EPERM] error.60364 **FUTURE DIRECTIONS**

60365 None.

60366 **SEE ALSO**60367 *pthread\_spin\_destroy()*, *pthread\_spin\_lock()*

60368 XBD Section 4.15.2 (on page 104), &lt;pthread.h&gt;

60369 **CHANGE HISTORY**

60370 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

60371 In the SYNOPSIS, the inclusion of &lt;sys/types.h&gt; is no longer required.

60372 **Issue 7**60373 The *pthread\_spin\_unlock()* function is moved from the Spin Locks option to the Base.60374 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in  
60375 undefined behavior.

60376  
60377

The [EPERM] error for a spin lock object for which the current thread does not hold the lock is removed; this condition results in undefined behavior.

60378 **NAME**

60379 pthread\_testcancel — set cancelability state

60380 **SYNOPSIS**

60381 #include <pthread.h>

60382 void pthread\_testcancel(void);

60383 **DESCRIPTION**

60384 Refer to *pthread\_setcancelstate()*.

60385 **NAME**

60386 ptsname, ptsname\_r — get name of the subsidiary pseudo-terminal device

60387 **SYNOPSIS**

```
60388 XSI      #include <stdlib.h>
60389      char *ptsname(int fildes);
60390      int ptsname_r(int fildes, char *name, size_t namesize);
```

60391 **DESCRIPTION**

60392 The *ptsname()* function shall return the name of the subsidiary pseudo-terminal device  
 60393 associated with a manager pseudo-terminal device. The *fildes* argument is a file descriptor that  
 60394 refers to the manager device. The *ptsname()* function shall return a pointer to a string containing  
 60395 the pathname of the corresponding subsidiary device.

60396 The *ptsname()* function need not be thread-safe.

60397 The *ptsname\_r()* function shall store the name of the subsidiary pseudo-terminal device  
 60398 corresponding to *fildes* in the character array referenced by *name*. The array is *namesize*  
 60399 characters long and should have space for the name and the terminating null character. The  
 60400 maximum length of the terminal name shall be {TTY\_NAME\_MAX}.

60401 **RETURN VALUE**

60402 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the  
 60403 pseudo-terminal subsidiary device. Upon failure, *ptsname()* shall return a null pointer and may  
 60404 set *errno*. This could occur if *fildes* is an invalid file descriptor or if the subsidiary device name  
 60405 does not exist in the file system.

60406 The application shall not modify the string returned. The returned pointer might be invalidated  
 60407 or the string content might be overwritten by a subsequent call to *ptsname()*. The returned  
 60408 pointer and the string content might also be invalidated if the calling thread is terminated.

60409 If successful, the *ptsname\_r()* function shall return zero. Otherwise, an error number shall be  
 60410 returned to indicate the error.

60411 **ERRORS**

60412 The *ptsname\_r()* function shall fail if:

60413 [EBADF] The *fildes* argument is not a valid file descriptor.

60414 [EINVAL] The *name* argument is a null pointer.

60415 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
 60416 including the terminating null character.

60417 The *ptsname()* function may fail if:

60418 [EBADF] The *fildes* argument is not a valid file descriptor.

60419 The *ptsname()* and *ptsname\_r()* functions may fail if:

60420 [EINVAL] or [ENOTTY]

60421 The file associated with the *fildes* argument is not a manager pseudo-terminal  
 60422 device.

60423 **EXAMPLES**

60424 None.

60425 **APPLICATION USAGE**

60426 None.

60427 **RATIONALE**

60428 The *ptsname\_r()* function is required to make it possible for a multi-threaded program to safely  
60429 determine the name of a subsidiary device. Although the name of the device is constrained by  
60430 {TTY\_NAME\_MAX}, this value might not be a compile-time constant, so an application can rely  
60431 on repeated calls with successively larger buffers until the result is no longer [ERANGE] as an  
60432 alternative for properly sizing the buffer.

60433 Historically, some versions of *ptsname()* did not set *errno* even when returning a null pointer.  
60434 However, *ptsname\_r()* is required to either populate the buffer with a valid name or return an  
60435 error value.

60436 See also the RATIONALE section for *posix\_openpt()*.60437 **FUTURE DIRECTIONS**

60438 None.

60439 **SEE ALSO**60440 *grantpt()*, *open()*, *posix\_openpt()*, *ttyname()*, *unlockpt()*

60441 XBD &lt;stdlib.h&gt;

60442 **CHANGE HISTORY**

60443 First released in Issue 4, Version 2.

60444 **Issue 5**

60445 Moved from X/OPEN UNIX extension to BASE.

60446 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

60447 **Issue 7**

60448 Austin Group Interpretation 1003.1-2001 #156 is applied.

60449 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0468 [75] and XSH/TC1-2008/0469  
60450 [96] are applied.

60451 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0298 [503], XSH/TC2-2008/0299 [656],  
60452 and XSH/TC2-2008/0300 [503] are applied.

60453 **Issue 8**60454 Austin Group Defect 508 is applied, adding the *ptsname\_r()* function.

60455 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal  
60456 devices.

60457 **NAME**

60458       putc — put a byte on a stream

60459 **SYNOPSIS**

60460       #include &lt;stdio.h&gt;

60461       int putc(int *c*, FILE \**stream*);60462 **DESCRIPTION**

60463 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
60464 conflict between the requirements described here and the ISO C standard is unintentional. This  
60465 volume of POSIX.1-2024 defers to the ISO C standard.

60466       The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it  
60467 may evaluate *stream* more than once, so the argument should never be an expression with side-  
60468 effects.

60469 **RETURN VALUE**60470       Refer to *fputc()*.60471 **ERRORS**60472       Refer to *fputc()*.60473 **EXAMPLES**

60474       None.

60475 **APPLICATION USAGE**

60476       Since it may be implemented as a macro, *putc()* may treat a *stream* argument with side-effects  
60477 incorrectly. In particular, *putc(c,\*f++)* does not necessarily work correctly. Therefore, use of this  
60478 function is not recommended in such situations; *fputc()* should be used instead.

60479 **RATIONALE**

60480       None.

60481 **FUTURE DIRECTIONS**

60482       None.

60483 **SEE ALSO**60484       Section 2.5 (on page 521), *fputc()*

60485       XBD &lt;stdio.h&gt;

60486 **CHANGE HISTORY**

60487       First released in Issue 1. Derived from Issue 1 of the SVID.

60488 **Issue 7**

60489       POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0470 [14] is applied.

60490 **NAME**

60491       putc\_unlocked — stdio with explicit client locking

60492 **SYNOPSIS**

```
60493 CX       #include <stdio.h>
60494       int putc_unlocked(int c, FILE *stream);
```

60495 **DESCRIPTION**

60496       Refer to [getc\\_unlocked\(\)](#).



60497 **NAME**

60498            putchar — put a byte on a stdout stream

60499 **SYNOPSIS**

60500            #include &lt;stdio.h&gt;

60501            int putchar(int c);

60502 **DESCRIPTION**

60503 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
60504        conflict between the requirements described here and the ISO C standard is unintentional. This  
60505        volume of POSIX.1-2024 defers to the ISO C standard.

60506        The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.60507 **RETURN VALUE**60508        Refer to *fputc()*.60509 **ERRORS**60510        Refer to *fputc()*.60511 **EXAMPLES**

60512        None.

60513 **APPLICATION USAGE**

60514        None.

60515 **RATIONALE**

60516        None.

60517 **FUTURE DIRECTIONS**

60518        None.

60519 **SEE ALSO**60520        [Section 2.5](#) (on page 521), *putc()*60521        XBD [<stdio.h>](#)60522 **CHANGE HISTORY**

60523        First released in Issue 1. Derived from Issue 1 of the SVID.

60524 **Issue 7**

60525        POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0471 [14] is applied.

60526 **NAME**

60527 putchar\_unlocked — stdio with explicit client locking

60528 **SYNOPSIS**

60529 CX `#include <stdio.h>`

60530 `int putchar_unlocked(int c);`

60531 **DESCRIPTION**

60532 Refer to [getc\\_unlocked\(\)](#).

60533 **NAME**

60534 putenv — change or add a value to an environment

60535 **SYNOPSIS**

```
60536 XSI      #include <stdlib.h>
60537         int putenv(char *string);
```

60538 **DESCRIPTION**60539 The *putenv()* function shall use the *string* argument to set, or optionally unset, an environment  
60540 variable value:

- 60541 • If the *string* argument points to a string of the form "*name=value*", where *name* is a valid  
60542 name, the *putenv()* function shall make the value of the environment variable with that  
60543 name equal to *value* by altering an existing variable or creating a new one. In either case,  
60544 the string pointed to by *string* shall become part of the environment, so altering the string  
60545 shall change the environment.
- 60546 • If the *string* argument points to a string containing a valid name, the *putenv()* function  
60547 shall either remove the environment variable with that name (if it exists) from the  
60548 environment or fail with *errno* set to [EINVAL].
- 60549 • Otherwise, the behavior is unspecified.

60550 The *putenv()* function need not be thread-safe.60551 **RETURN VALUE**60552 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value  
60553 and set *errno* to indicate the error.60554 **ERRORS**60555 The *putenv()* function may fail if:

- 60556 [EINVAL] The *string* argument points to a string that is not of the form "*name=value*",  
60557 where *name* is a valid name.
- 60558 [ENOMEM] Insufficient memory was available.

60559 **EXAMPLES**60560 **Changing the Value of an Environment Variable**60561 The following example changes the value of the *HOME* environment variable to the value  
60562 */usr/home*.

```
60563 #include <stdlib.h>
60564 ...
60565 static char *var = "HOME=/usr/home";
60566 int ret;
60567
60567 ret = putenv(var);
```

60568 **APPLICATION USAGE**60569 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in  
60570 conjunction with *getenv()*.60571 See *exec()* for restrictions on changing the environment in multi-threaded applications.60572 This routine may use *malloc()* to enlarge the environment.60573 A potential error is to call *putenv()* with an automatic variable as the argument, then return from

60574 the calling function while *string* is still part of the environment.

60575 Although the space used by *string* is no longer used once a new string which defines *name* is  
60576 passed to *putenv()*, if any thread in the application has used *getenv()* to retrieve a pointer to this  
60577 variable, it should not be freed by calling *free()*. If the changed environment variable is one  
60578 known by the system (such as the locale environment variables) the application should never  
60579 free the buffer used by earlier calls to *putenv()* for the same variable.

60580 The *setenv()* function is preferred over this function. One reason is that *putenv()* is optional and  
60581 therefore less portable. Another is that using *putenv()* can slow down environment searches, as  
60582 explained in the RATIONALE section for *getenv()*.

60583 **RATIONALE**

60584 Refer to the RATIONALE section in *setenv()*.

60585 **FUTURE DIRECTIONS**

60586 None.

60587 **SEE ALSO**

60588 *exec*, *free()*, *getenv()*, *malloc()*, *setenv()*

60589 XBD <stdlib.h>

60590 **CHANGE HISTORY**

60591 First released in Issue 1. Derived from Issue 1 of the SVID.

60592 **Issue 5**

60593 The type of the argument to this function is changed from **const char \*** to **char \***. This was  
60594 indicated as a FUTURE DIRECTION in previous issues.

60595 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

60596 **Issue 6**

60597 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the  
60598 DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to  
60599 *exec*.

60600 **Issue 7**

60601 Austin Group Interpretation 1003.1-2001 #156 is applied.

60602 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0472 [167], XSH/TC1-2008/0473 [167],  
60603 XSH/TC1-2008/0474 [273,438], and XSH/TC1-2008/0475 [273] are applied.

60604 **Issue 8**

60605 Austin Group Defect 1598 is applied, specifying the allowed behaviors when the *string* argument  
60606 points to a string containing a valid name.

60607 **NAME**

60608 puts — put a string on standard output

60609 **SYNOPSIS**

```
60610 #include <stdio.h>
60611 int puts(const char *s);
```

60612 **DESCRIPTION**

60613 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 60614 conflict between the requirements described here and the ISO C standard is unintentional. This  
 60615 volume of POSIX.1-2024 defers to the ISO C standard.

60616 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the  
 60617 standard output stream *stdout*. The terminating null byte shall not be written.

60618 CX The last data modification and last file status change timestamps of the file shall be marked for  
 60619 update between the successful execution of *puts()* and the next successful completion of a call to  
 60620 *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

60621 **RETURN VALUE**

60622 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall  
 60623 CX return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

60624 **ERRORS**60625 Refer to *fputc()*.60626 **EXAMPLES**60627 **Printing to Standard Output**

60628 The following example gets the current time, converts it to a string using *localtime()* and  
 60629 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to  
 60630 an event for which it is waiting.

```
60631 #include <time.h>
60632 #include <stdio.h>
60633 ...
60634 time_t now;
60635 int minutes_to_event;
60636 ...
60637 time(&now);
60638 printf("The time is ");
60639 puts(asctime(localtime(&now)));
60640 printf("There are %d minutes to the event.\n",
60641     minutes_to_event);
60642 ...
```

60643 **APPLICATION USAGE**60644 The *puts()* function appends a <newline>, while *fputs()* does not.

60645 This volume of POSIX.1-2024 requires that successful completion simply return a non-negative  
 60646 integer. There are at least three known different implementation conventions for this  
 60647 requirement:

- 60648 • Return a constant value.

60649           • Return the last character written.

60650           • Return the number of bytes written. Note that this implementation convention cannot be  
60651           adhered to for strings longer than {INT\_MAX} bytes as the value would not be  
60652           representable in the return type of the function. For backwards compatibility,  
60653           implementations can return the number of bytes for strings of up to {INT\_MAX} bytes, and  
60654           return {INT\_MAX} for all longer strings.

60655   **RATIONALE**  
60656           None.

60657   **FUTURE DIRECTIONS**  
60658           None.

60659   **SEE ALSO**  
60660           [Section 2.5](#) (on page 521), [fopen\(\)](#), [fputs\(\)](#), [putc\(\)](#)  
60661           XBD [<stdio.h>](#)

60662   **CHANGE HISTORY**  
60663           First released in Issue 1. Derived from Issue 1 of the SVID.

60664   **Issue 6**  
60665           Extensions beyond the ISO C standard are marked.

60666   **Issue 7**  
60667           Changes are made related to support for finegrained timestamps.

60668           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0476 [174,412] and  
60669           XSH/TC1-2008/0477 [14] are applied.

60670 **NAME**

60671 pututxline — put an entry into the user accounting database

60672 **SYNOPSIS**

```
60673 XSI #include <utmpx.h>  
60674 struct utmpx *pututxline(const struct utmpx *utmpx);
```

60675 **DESCRIPTION**60676 Refer to *endutxent()*.

60677 **NAME**

60678 putwc — put a wide character on a stream

60679 **SYNOPSIS**

60680 #include &lt;stdio.h&gt;

60681 #include &lt;wchar.h&gt;

60682 wint\_t putwc(wchar\_t *wc*, FILE \**stream*);60683 **DESCRIPTION**60684 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
60685 conflict between the requirements described here and the ISO C standard is unintentional. This  
60686 volume of POSIX.1-2024 defers to the ISO C standard.60687 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro  
60688 it may evaluate *stream* more than once, so the argument should never be an expression with  
60689 side-effects.60690 **RETURN VALUE**60691 Refer to *fputwc()*.60692 **ERRORS**60693 Refer to *fputwc()*.60694 **EXAMPLES**

60695 None.

60696 **APPLICATION USAGE**60697 Since it may be implemented as a macro, *putwc()* may treat a *stream* argument with side-effects  
60698 incorrectly. In particular, *putwc(wc,\*f++)* need not work correctly. Therefore, use of this function  
60699 is not recommended; *fputwc()* should be used instead.60700 **RATIONALE**

60701 None.

60702 **FUTURE DIRECTIONS**

60703 None.

60704 **SEE ALSO**60705 [Section 2.5](#) (on page 521), *fputwc()*

60706 XBD &lt;stdio.h&gt;, &lt;wchar.h&gt;

60707 **CHANGE HISTORY**

60708 First released as a World-wide Portability Interface in Issue 4.

60709 **Issue 5**60710 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
60711 is changed from **wint\_t** to **wchar\_t**.

60712 The Optional Header (OH) marking is removed from &lt;stdio.h&gt;.

60713 **Issue 7**

60714 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0478 [14] is applied.



60715 **NAME**

60716 putwchar — put a wide character on a stdout stream

60717 **SYNOPSIS**

60718 #include &lt;wchar.h&gt;

60719 wint\_t putwchar(wchar\_t wc);

60720 **DESCRIPTION**

60721 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
60722 conflict between the requirements described here and the ISO C standard is unintentional. This  
60723 volume of POSIX.1-2024 defers to the ISO C standard.

60724 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.60725 **RETURN VALUE**60726 Refer to *fputwc()*.60727 **ERRORS**60728 Refer to *fputwc()*.60729 **EXAMPLES**

60730 None.

60731 **APPLICATION USAGE**

60732 None.

60733 **RATIONALE**

60734 None.

60735 **FUTURE DIRECTIONS**

60736 None.

60737 **SEE ALSO**60738 [Section 2.5](#) (on page 521), *fputwc()*, *putwc()*60739 XBD [<wchar.h>](#)60740 **CHANGE HISTORY**

60741 First released in Issue 4.

60742 **Issue 5**

60743 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
60744 is changed from **wint\_t** to **wchar\_t**.

60745 **Issue 7**

60746 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0479 [14] is applied.

60747 **NAME**

60748       pwrite — write on a file

60749 **SYNOPSIS**

60750       #include <unistd.h>

60751       ssize\_t pwrite(int *fd*, const void \**buf*, size\_t *nbyte*,  
60752                   off\_t *offset*);

60753 **DESCRIPTION**

60754       Refer to *write()*.

60755 **NAME**

60756 qsort, qsort\_r — sort a table of data

60757 **SYNOPSIS**

60758 #include &lt;stdlib.h&gt;

60759 void qsort(void \*base, size\_t nel, size\_t width,  
60760 int (\*compar)(const void \*, const void \*));60761 CX void qsort\_r(void \*base, size\_t nel, size\_t width,  
60762 int (\*compar)(const void \*, const void \*, void \*), void \*arg);60763 **DESCRIPTION**60764 CX For *qsort()*: The functionality described on this reference page is aligned with the ISO C  
60765 standard. Any conflict between the requirements described here and the ISO C standard is  
60766 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.60767 The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by  
60768 *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has  
60769 the value zero, the comparison function pointed to by *compar* shall not be called and no  
60770 rearrangement shall take place.60771 The application shall ensure that the comparison function pointed to by *compar* does not alter the  
60772 contents of the array. The implementation may reorder elements of the array between calls to the  
60773 comparison function, but shall not alter the contents of any individual element.60774 When the same objects (consisting of *width* bytes, irrespective of their current positions in the  
60775 array) are passed more than once to the comparison function, the results shall be consistent with  
60776 one another. That is, they shall define a total ordering on the array.60777 The contents of the array shall be sorted in ascending order according to a comparison function.  
60778 The *compar* argument is a pointer to the comparison function, which is called with two  
60779 arguments that point to the elements being compared. The application shall ensure that the  
60780 function returns an integer less than, equal to, or greater than 0, if the first argument is  
60781 considered respectively less than, equal to, or greater than the second. If two members compare  
60782 as equal, their order in the sorted array is unspecified.60783 CX The *qsort\_r()* function shall be identical to *qsort()* except that the comparison function *compar*  
60784 takes a third argument. The *arg* opaque pointer passed to *qsort\_r()* shall in turn be passed as the  
60785 third argument to the comparison function.60786 **RETURN VALUE**

60787 These functions shall not return a value.

60788 **ERRORS**

60789 No errors are defined.

60790 **EXAMPLES**

60791 None.

60792 **APPLICATION USAGE**60793 The comparison function need not compare every byte, so arbitrary data may be contained in  
60794 the elements in addition to the values being compared.60795 If the *compar* callback function requires any additional state outside of the items being sorted, it  
60796 can only access this state through global variables, making it potentially unsafe to use *qsort()*  
60797 with the same *compar* function from separate threads at the same time. The *qsort\_r()* function  
60798 was added with the ability to pass through arbitrary arguments to the comparator, which avoids  
60799 the need to access global variables and thus making it possible to safely share a stateful

60800 comparator across threads.

60801 **RATIONALE**

60802 The requirement that each argument (hereafter referred to as *p*) to the comparison function is a  
60803 pointer to elements of the array implies that for every call, for each argument separately, all of  
60804 the following expressions are non-zero:

60805 `((char *)p - (char *)base) % width == 0`

60806 `(char *)p >= (char *)base`

60807 `(char *)p < (char *)base + nel * width`

60808 **FUTURE DIRECTIONS**

60809 None.

60810 **SEE ALSO**

60811 [\*alphasort\(\)\*](#)

60812 XBD [`<stdlib.h>`](#)

60813 **CHANGE HISTORY**

60814 First released in Issue 1. Derived from Issue 1 of the SVID.

60815 **Issue 6**

60816 The normative text is updated to avoid use of the term “must” for application requirements.

60817 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to  
60818 the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The  
60819 RATIONALE is also updated. These changes are for alignment with the ISO C standard.

60820 **Issue 8**

60821 Austin Group Defect 900 is applied, adding the *qsort\_r()* function.

60822 **NAME**

60823 quick\_exit — terminate a process

60824 **SYNOPSIS**

60825 #include &lt;stdlib.h&gt;

60826 \_Noreturn void quick\_exit(int status);

60827 **DESCRIPTION**

60828 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
60829 conflict between the requirements described here and the ISO C standard is unintentional. This  
60830 volume of POSIX.1-2024 defers to the ISO C standard.

60831 The *quick\_exit()* function shall cause normal process termination to occur. It shall not call  
60832 functions registered with *atexit()* nor any registered signal handlers. If a process calls the  
60833 *quick\_exit()* function more than once, or calls the *exit()* function in addition to the *quick\_exit()*  
60834 function, the behavior is undefined. If a signal is raised while the *quick\_exit()* function is  
60835 executing, the behavior is undefined.

60836 The *quick\_exit()* function shall first call all functions registered by *at\_quick\_exit()*, in the reverse  
60837 order of their registration, except that a function is called after any previously registered  
60838 functions that had already been called at the time it was registered. If, during the call to any such  
60839 CX function, a call to the *longjmp()* or *siglongjmp()* function is made that would terminate the call  
60840 to the registered function, the behavior is undefined.

60841 If a function registered by a call to *at\_quick\_exit()* fails to return, the remaining registered  
60842 functions shall not be called and the rest of the *quick\_exit()* processing shall not be completed.

60843 Finally, the *quick\_exit()* function shall terminate the process as if by a call to *\_Exit(status)*.

60844 **RETURN VALUE**60845 The *quick\_exit()* function does not return.60846 **ERRORS**

60847 No errors are defined.

60848 **EXAMPLES**

60849 None.

60850 **APPLICATION USAGE**

60851 None.

60852 **RATIONALE**

60853 None.

60854 **FUTURE DIRECTIONS**

60855 None.

60856 **SEE ALSO**60857 [\*\\_Exit\(\)\*](#), [\*at\\_quick\\_exit\(\)\*](#), [\*atexit\(\)\*](#), [\*exit\(\)\*](#)

60858 XBD &lt;stdlib.h&gt;

60859 **CHANGE HISTORY**

60860 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

60861 **NAME**

60862 raise — send a signal to the executing process

60863 **SYNOPSIS**

60864 #include &lt;signal.h&gt;

60865 int raise(int sig);

60866 **DESCRIPTION**60867 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
60868 conflict between the requirements described here and the ISO C standard is unintentional. This  
60869 volume of POSIX.1-2024 defers to the ISO C standard.60870 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal  
60871 handler is called, the *raise()* function shall not return until after the signal handler does.60872 CX The effect of the *raise()* function shall be equivalent to calling:

60873 pthread\_kill(pthread\_self(), sig);

60874 **RETURN VALUE**60875 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned  
60876 and *errno* shall be set to indicate the error.60877 **ERRORS**60878 The *raise()* function shall fail if:60879 CX [EINVAL] The value of the *sig* argument is an invalid signal number.60880 **EXAMPLES**

60881 None.

60882 **APPLICATION USAGE**

60883 None.

60884 **RATIONALE**

60885 The term “thread” is an extension to the ISO C standard.

60886 **FUTURE DIRECTIONS**

60887 None.

60888 **SEE ALSO**60889 [kill\(\)](#), [sigaction\(\)](#)60890 XBD [<signal.h>](#), [<sys/types.h>](#)60891 **CHANGE HISTORY**

60892 First released in Issue 4. Derived from the ANSI C standard.

60893 **Issue 5**

60894 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

60895 **Issue 6**

60896 Extensions beyond the ISO C standard are marked.

60897 The following new requirements on POSIX implementations derive from alignment with the  
60898 Single UNIX Specification:

- 60899
- In the RETURN VALUE section, the requirement to set *errno* on error is added.

60900

- The [EINVAL] error condition is added.

60901 **Issue 7**

60902

Functionality relating to the Threads option is moved to the Base.

60903 **NAME**

60904 rand, srand — pseudo-random number generator

60905 **SYNOPSIS**

```
60906 #include <stdlib.h>
60907 int rand(void);
60908 void srand(unsigned seed);
```

60909 **DESCRIPTION**

60910 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 60911 conflict between the requirements described here and the ISO C standard is unintentional. This  
 60912 volume of POSIX.1-2024 defers to the ISO C standard.

60913 The *rand()* function shall compute a sequence of pseudo-random integers in the range  
 60914 XSI [0,{RAND\_MAX}] with a period of at least  $2^{32}$ .

60915 The *rand()* function need not be thread-safe; however, *rand()* shall avoid data races with all  
 60916 functions other than non-thread-safe pseudo-random sequence generation functions.

60917 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random  
 60918 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same  
 60919 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before  
 60920 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first  
 60921 called with a seed value of 1.

60922 The *srand()* function need not be thread-safe; however, *srand()* shall avoid data races with all  
 60923 functions other than non-thread-safe pseudo-random sequence generation functions.

60924 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 60925 *rand()* or *srand()*.

60926 **RETURN VALUE**60927 The *rand()* function shall return the next pseudo-random number in the sequence.60928 The *srand()* function shall not return a value.60929 **ERRORS**

60930 No errors are defined.

60931 **EXAMPLES**60932 **Generating a Pseudo-Random Number Sequence**

60933 The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
60934 #include <stdio.h>
60935 #include <stdlib.h>
60936 ...
60937     long count, i;
60938     char *keyst;
60939     int elementlen, len;
60940     char c;
60941     ...
60942 /* Initial random number generator. */
60943     srand(1);
60944
60945     /* Create keys using only lowercase characters */
60946     len = 0;
60947     for (i=0; i<count; i++) {
```



```

60947         while (len < elementlen) {
60948             c = (char) (rand() % 128);
60949             if (islower(c))
60950                 keystr[len++] = c;
60951         }
60952         keystr[len] = '\0';
60953         printf("%s Element%0*ld\n", keystr, elementlen, i);
60954         len = 0;
60955     }

```

### 60956 **Generating the Same Sequence on Different Machines**

60957 The following code defines a pair of functions that could be incorporated into applications  
60958 wishing to ensure that the same sequence of numbers is generated across different machines.

```

60959 static unsigned long next = 1;
60960 int myrand(void) /* RAND_MAX assumed to be 32767. */
60961 {
60962     next = next * 1103515245 + 12345;
60963     return((unsigned) (next/65536) % 32768);
60964 }
60965 void mysrand(unsigned seed)
60966 {
60967     next = seed;
60968 }

```

### 60969 **APPLICATION USAGE**

60970 These functions should be avoided whenever non-trivial requirements (including safety) have to  
60971 be fulfilled, unless seeded using *getentropy()*.

60972 The *drand48()* and *random()* functions provide much more elaborate pseudo-random number  
60973 generators.

### 60974 **RATIONALE**

60975 The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams  
60976 shared by all threads. Those two functions need not change, but there has to be mutual-  
60977 exclusion that prevents interference between two threads concurrently accessing the random  
60978 number generator.

60979 With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded  
60980 program:

- 60981 1. A single per-process sequence of pseudo-random numbers that is shared by all threads  
60982 that call *rand()*
- 60983 2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

60984 This is provided by the modified thread-safe function based on whether the seed value is global  
60985 to the entire process or local to each thread.

60986 This does not address the known deficiencies of the *rand()* function implementations, which  
60987 have been approached by maintaining more state. In effect, this specifies new thread-safe forms  
60988 of a deficient function.

60989 **FUTURE DIRECTIONS**

60990 None.

60991 **SEE ALSO**60992 *drand48()*, *getentropy()*, *initstate()*

60993 XBD &lt;stdlib.h&gt;

60994 **CHANGE HISTORY**

60995 First released in Issue 1. Derived from Issue 1 of the SVID.

60996 **Issue 5**60997 The *rand\_r()* function is included for alignment with the POSIX Threads Extension.60998 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.60999 **Issue 6**

61000 Extensions beyond the ISO C standard are marked.

61001 The *rand\_r()* function is marked as part of the Thread-Safe Functions option.61002 **Issue 7**

61003 Austin Group Interpretation 1003.1-2001 #156 is applied.

61004 The *rand\_r()* function is marked obsolescent.

61005 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0301 [743] is applied.

61006 **Issue 8**61007 Austin Group Defect 1134 is applied, adding *getentropy()*.

61008 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018 standard.

61010 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

61011 **NAME**

61012 random — generate pseudo-random number

61013 **SYNOPSIS**61014 XSI `#include <stdlib.h>`61015 `long random(void);`61016 **DESCRIPTION**61017 Refer to *initstate()*.

61018 **NAME**

61019           pread, read — read from a file

61020 **SYNOPSIS**

61021       #include &lt;unistd.h&gt;

61022       ssize\_t pread(int *fildes*, void \**buf*, size\_t *nbyte*, off\_t *offset*);61023       ssize\_t read(int *fildes*, void \**buf*, size\_t *nbyte*);61024 **DESCRIPTION**

61025       The *read()* function shall attempt to read *nbyte* bytes from the file associated with the open file  
 61026       descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on  
 61027       the same pipe, FIFO, or terminal device is unspecified.

61028       Before any action described below is taken, and if *nbyte* is zero, the *read()* function may detect  
 61029       and return errors as described below. In the absence of errors, or if error detection is not  
 61030       performed, the *read()* function shall return zero and have no other results.

61031       On files that support seeking (for example, a regular file), the *read()* shall start at a position in  
 61032       the file given by the file offset associated with *fildes*. The file offset shall be incremented by the  
 61033       number of bytes actually read.

61034       Files that do not support seeking—for example, terminals—always read from the current  
 61035       position. The value of a file offset associated with such a file is undefined.

61036       No data transfer shall occur past the current end-of-file. If the starting position is at or after the  
 61037       end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent  
 61038       *read()* requests is implementation-defined.

61039       If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.

61040       When attempting to read from an empty pipe or FIFO:

- 61041           • If no process has the pipe open for writing, *read()* shall return 0 to indicate end-of-file.
- 61042           • If some process has the pipe open for writing and O\_NONBLOCK is set, *read()* shall return  
 61043            -1 and set *errno* to [EAGAIN].
- 61044           • If some process has the pipe open for writing and O\_NONBLOCK is clear, *read()* shall  
 61045            block the calling thread until some data is written or the pipe is closed by all processes that  
 61046            had the pipe open for writing.

61047       When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and  
 61048       has no data currently available:

- 61049           • If O\_NONBLOCK is set, *read()* shall return -1 and set *errno* to [EAGAIN].
- 61050           • If O\_NONBLOCK is clear, *read()* shall block the calling thread until some data becomes  
 61051            available.
- 61052           • The use of the O\_NONBLOCK flag has no effect if there is some data available.

61053       The *read()* function reads data previously written to a file. If any portion of a regular file prior to  
 61054       the end-of-file has not been written, *read()* shall return bytes with value 0. For example, *lseek()*  
 61055       allows the file offset to be set beyond the end of existing data in the file. If data is later written at  
 61056       this point, subsequent reads in the gap between the previous end of data and the newly written  
 61057       data shall return bytes with value 0 until data is written into the gap.

61058       Upon successful completion, where *nbyte* is greater than 0, *read()* shall mark for update the last  
 61059       data access timestamp of the file, and shall return the number of bytes read. This number shall  
 61060       never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes

61061		left in the file is less than <i>nbyte</i> , if the <i>read()</i> request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than <i>nbyte</i> bytes immediately available for reading. For example, a <i>read()</i> from a file associated with a terminal may return one typed line of data.
61062		
61063		
61064		If a <i>read()</i> is interrupted by a signal before it reads any data, it shall return $-1$ with <i>errno</i> set to [EINTR].
61065		
61066		If a <i>read()</i> is interrupted by a signal after it has successfully read some data, it shall return the number of bytes read.
61067		
61068		For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with <i>fildev</i> .
61069		
61070		If <i>fildev</i> refers to a socket, <i>read()</i> shall be equivalent to <i>recv()</i> with no flags set.
61071	SIO	If the O_DSYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If the O_SYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
61072		
61073		
61074		
61075	SHM	If <i>fildev</i> refers to a shared memory object, the result of the <i>read()</i> function is unspecified.
61076	TYM	If <i>fildev</i> refers to a typed memory object, the result of the <i>read()</i> function is unspecified.
61077		The <i>pread()</i> function shall be equivalent to <i>read()</i> , except that it shall read from a given position in the file without changing the file offset. The first three arguments to <i>pread()</i> are the same as <i>read()</i> with the addition of a fourth argument <i>offset</i> for the desired position inside the file. An attempt to perform a <i>pread()</i> on a file that is incapable of seeking shall result in an error.
61078		
61079		
61080		
61081		<b>RETURN VALUE</b>
61082		Upon successful completion, these functions shall return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions shall return $-1$ and set <i>errno</i> to indicate the error.
61083		
61084		
61085		<b>ERRORS</b>
61086		These functions shall fail if:
61087	[EAGAIN]	The file is neither a pipe, nor a FIFO, nor a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the read operation.
61088		
61089		
61090	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor open for reading.
61091	[EINTR]	The read operation was terminated due to the receipt of a signal, and no data was transferred.
61092		
61093	[EIO]	The process is a member of a background process group attempting to read from its controlling terminal, and either the calling thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process group of the process is orphaned. This error may also be generated for implementation-defined reasons.
61094		
61095		
61096		
61097		
61098	XSI	[EISDIR] The <i>fildev</i> argument refers to a directory and the implementation does not allow the directory to be read using <i>read()</i> or <i>pread()</i> . The <i>readdir()</i> function should be used instead.
61099		
61100		
61101	[EOVERFLOW]	The file is a regular file, <i>nbyte</i> is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fildev</i> .
61102		
61103		

- 61104 The *pread()* function shall fail if:
- 61105 [EINVAL] The file is a regular file or block special file, and the *offset* argument is  
61106 negative. The file offset shall remain unchanged.
- 61107 [ESPIPE] The file is incapable of seeking.
- 61108 The *read()* function shall fail if:
- 61109 [EAGAIN] The file is a pipe or FIFO, the O\_NONBLOCK flag is set for the file descriptor,  
61110 and the thread would be delayed in the read operation.
- 61111 [EAGAIN] or [EWOULDBLOCK]  
61112 The file is a socket, the O\_NONBLOCK flag is set for the file descriptor, and  
61113 the thread would be delayed in the read operation.
- 61114 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by  
61115 its peer.
- 61116 [ENOTCONN] A read was attempted on a socket that is not connected.
- 61117 [ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.
- 61118 These functions may fail if:
- 61119 [EIO] A physical I/O error has occurred.
- 61120 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 61121 [ENOMEM] Insufficient memory was available to fulfill the request.
- 61122 [ENXIO] A request was made of a nonexistent device, or the request was outside the  
61123 capabilities of the device.

#### 61124 EXAMPLES

##### 61125 Reading Data into a Buffer

61126 The following example reads data from the file associated with the file descriptor *fd* into the  
61127 buffer pointed to by *buf*.

```
61128 #include <sys/types.h>
61129 #include <unistd.h>
61130 ...
61131 char buf[20];
61132 size_t nbytes;
61133 ssize_t bytes_read;
61134 int fd;
61135 ...
61136 nbytes = sizeof(buf);
61137 bytes_read = read(fd, buf, nbytes);
61138 ...
```

#### 61139 APPLICATION USAGE

61140 None.

#### 61141 RATIONALE

61142 This volume of POSIX.1-2024 does not specify the value of the file offset after an error is  
61143 returned; there are too many cases. For programming errors, such as [EBADF], the concept is  
61144 meaningless since no file is involved. For errors that are detected immediately, such as  
61145 [EAGAIN], clearly the offset should not change. After an interrupt or hardware error, however,  
61146 an updated value would be very useful and is the behavior of many implementations.

61147 Note that a `read()` of zero bytes does not modify the last data access timestamp. A `read()` that  
61148 requests more than zero bytes, but returns zero, is required to modify the last data access  
61149 timestamp.

61150 Implementations are allowed, but not required, to perform error checking for `read()` requests of  
61151 zero bytes.

## 61152 **Input and Output**

61153 The use of I/O with large byte counts has always presented problems. Ideas such as `lread()` and  
61154 `lwrite()` (using and returning **longs**) were considered at one time. The current solution is to use  
61155 abstract types on the ISO C standard function to `read()` and `write()`. The abstract types can be  
61156 declared so that existing functions work, but can also be declared so that larger types can be  
61157 represented in future implementations. It is presumed that whatever constraints limit the  
61158 maximum range of **size\_t** also limit portable I/O requests to the same range. This volume of  
61159 POSIX.1-2024 also limits the range further by requiring that the byte count be limited so that a  
61160 signed return value remains meaningful. Since the return type is also a (signed) abstract type,  
61161 the byte count can be defined by the implementation to be larger than an **int** can hold.

61162 The standard developers considered adding atomicity requirements to a pipe or FIFO, but  
61163 recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of  
61164 reads of {PIPE\_BUF} or any other size that would be an aid to applications portability.

61165 This volume of POSIX.1-2024 requires that no action be taken for `read()` or `write()` when *nbyte* is  
61166 zero. This is not intended to take precedence over detection of errors (such as invalid buffer  
61167 pointers or file descriptors). This is consistent with the rest of this volume of POSIX.1-2024, but  
61168 the phrasing here could be misread to require detection of the zero case before any other errors.  
61169 A value of zero is to be considered a correct value, for which the semantics are a no-op.

61170 I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the  
61171 bytes from a single operation that started out together end up together, without interleaving  
61172 from other I/O operations. It is a known attribute of terminals that this is not honored, and  
61173 terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified.  
61174 The behavior for other device types is also left unspecified, but the wording is intended to imply  
61175 that future standards might choose to specify atomicity (or not).

61176 There were recommendations to add format parameters to `read()` and `write()` in order to handle  
61177 networked transfers among heterogeneous file system and base hardware types. Such a facility  
61178 may be required for support by the OSI presentation of layer services. However, it was  
61179 determined that this should correspond with similar C-language facilities, and that is beyond  
61180 the scope of this volume of POSIX.1-2024. The concept was suggested to the developers of the  
61181 ISO C standard for their consideration as a possible area for future work.

61182 In 4.3 BSD, a `read()` or `write()` that is interrupted by a signal before transferring any data does  
61183 not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth  
61184 Edition, there is an additional function, `select()`, whose purpose is to pause until specified  
61185 activity (data to read, space to write, and so on) is detected on specified file descriptors. It is  
61186 common in applications written for those systems for `select()` to be used before `read()` in  
61187 situations (such as keyboard input) where interruption of I/O due to a signal is desired.

61188 The issue of which files or file types are interruptible is considered an implementation design  
61189 issue. This is often affected primarily by hardware and reliability issues.

61190 There are no references to actions taken following an “unrecoverable error”. It is considered  
61191 beyond the scope of this volume of POSIX.1-2024 to describe what happens in the case of  
61192 hardware errors.

61193 Earlier versions of this standard allowed two very different behaviors with regard to the  
 61194 handling of interrupts. In order to minimize the resulting confusion, it was decided that  
 61195 POSIX.1-2024 should support only one of these behaviors. Historical practice on AT&T-derived  
 61196 systems was to have *read()* and *write()* return  $-1$  and set *errno* to [EINTR] when interrupted after  
 61197 some, but not all, of the data requested had been transferred. However, the US Department of  
 61198 Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in which *read()* and  
 61199 *write()* return the number of bytes actually transferred before the interrupt. If  $-1$  is returned  
 61200 when any data is transferred, it is difficult to recover from the error on a seekable device and  
 61201 impossible on a non-seekable device. Most new implementations support this behavior. The  
 61202 behavior required by POSIX.1-2024 is to return the number of bytes transferred.

61203 POSIX.1-2024 does not specify when an implementation that buffers *read()*s actually moves the  
 61204 data into the user-supplied buffer, so an implementation may choose to do this at the latest  
 61205 possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a  
 61206 partial byte count, but rather to return  $-1$  and set *errno* to [EINTR].

61207 Consideration was also given to combining the two previous options, and setting *errno* to  
 61208 [EINTR] while returning a short count. However, not only is there no existing practice that  
 61209 implements this, it is also contradictory to the idea that when *errno* is set, the function  
 61210 responsible shall return  $-1$ .

61211 This volume of POSIX.1-2024 intentionally does not specify any *pread()* errors related to pipes,  
 61212 FIFOs, and sockets other than [ESPIPE].

#### 61213 FUTURE DIRECTIONS

61214 None.

#### 61215 SEE ALSO

61216 *fcntl()*, *lseek()*, *open()*, *pipe()*, *readv()*

61217 XBD Chapter 11 (on page 199), [<sys/uio.h>](#), [<unistd.h>](#)

#### 61218 CHANGE HISTORY

61219 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 61220 Issue 5

61221 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 61222 Threads Extension.

61223 Large File Summit extensions are added.

61224 The *pread()* function is added.

#### 61225 Issue 6

61226 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
 61227 marked as part of the XSI STREAMS Option Group.

61228 The following new requirements on POSIX implementations derive from alignment with the  
 61229 Single UNIX Specification:

- 61230 • The DESCRIPTION now states that if *read()* is interrupted by a signal after it has  
 61231 successfully read some data, it returns the number of bytes read. In Issue 3, it was optional  
 61232 whether *read()* returned the number of bytes read, or whether it returned  $-1$  with *errno* set  
 61233 to [EINTR]. This is a FIPS requirement.
- 61234 • In the DESCRIPTION, text is added to indicate that for regular files, no data transfer  
 61235 occurs past the offset maximum established in the open file description associated with  
 61236 *files*. This change is to support large files.



- 61237           • The [Eoverflow] mandatory error condition is added.
- 61238           • The [ENXIO] optional error condition is added.
- 61239           Text referring to sockets is added to the DESCRIPTION.
- 61240           The following changes were made to align with the IEEE P1003.1a draft standard:
- 61241           • The effect of reading zero bytes is clarified.
- 61242           The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *read()* results are unspecified for typed memory objects.
- 61243
- 61244           New RATIONALE is added to explain the atomicity requirements for input and output operations.
- 61245
- 61246           The following error conditions are added for operations on sockets: [EAGAIN], [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].
- 61247
- 61248           The [EIO] error is made optional.
- 61249           The following error conditions are added for operations on sockets: [ENOBUFS] and [ENOMEM].
- 61250
- 61251           The *readv()* function is split out into a separate reference page.
- 61252           IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/108 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.
- 61253
- 61254
- 61255           IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/109 is applied, making an editorial correction in the RATIONALE section.
- 61256
- 61257   **Issue 7**
- 61258           The *pread()* function is moved from the XSI option to the Base.
- 61259           Functionality relating to the XSI STREAMS option is marked obsolescent.
- 61260           Changes are made related to support for finegrained timestamps.
- 61261           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0480 [218], XSH/TC1-2008/0481 [79], XSH/TC1-2008/0482 [218], XSH/TC1-2008/0483 [218], XSH/TC1-2008/0484 [218], and XSH/TC1-2008/0485 [218,428] are applied.
- 61262
- 61263
- 61264           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0302 [710] and XSH/TC2-2008/0303 [676,710] are applied.
- 61265
- 61266   **Issue 8**
- 61267           Austin Group Defect 1330 is applied, removing obsolescent interfaces.

61268 **NAME**

61269 readdir, readdir\_r — read a directory

61270 **SYNOPSIS**

61271 #include &lt;dirent.h&gt;

61272 struct dirent \*readdir(DIR \*dirp);

61273 OB int readdir\_r(DIR \*restrict dirp, struct dirent \*restrict entry,  
61274 struct dirent \*\*restrict result);61275 **DESCRIPTION**

61276 The type **DIR**, which is defined in the <dirent.h> header, represents a *directory stream*, which is  
 61277 an ordered sequence of all the directory entries in a particular directory. Directory entries  
 61278 represent files; files may be removed from a directory or added to a directory asynchronously to  
 61279 the operation of *readdir()*.

61280 The *readdir()* function shall return a pointer to a structure representing the directory entry at the  
 61281 current position in the directory stream specified by the argument *dirp*, and position the  
 61282 directory stream at the next entry. It shall return a null pointer upon reaching the end of the  
 61283 directory stream. The structure **dirent** defined in the <dirent.h> header describes a directory  
 61284 entry. The value of the structure's *d\_ino* member shall be set to the file serial number of the file  
 61285 named by the *d\_name* member. If the *d\_name* member names a symbolic link, the value of the  
 61286 *d\_ino* member shall be set to the file serial number of the symbolic link itself. The *d\_name*  
 61287 member shall be a filename string, and (if not dot or dot-dot) shall contain the same byte  
 61288 sequence as the last pathname component of the string used to create the directory entry, plus  
 61289 the terminating <NUL> byte.

61290 The *readdir()* function shall not return directory entries containing empty names. If entries for  
 61291 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-  
 61292 dot; otherwise, they shall not be returned.

61293 The application shall not modify the structure to which the return value of *readdir()* points, nor  
 61294 any storage areas pointed to by pointers within the structure. The returned pointer, and pointers  
 61295 within the structure, might be invalidated or the structure or the storage areas might be  
 61296 overwritten by a subsequent call to *readdir()* on the same directory stream. They shall not be  
 61297 affected by a call to *readdir()* on a different directory stream. The returned pointer, and pointers  
 61298 within the structure, might also be invalidated if the calling thread is terminated.

61299 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
 61300 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

61301 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*  
 61302 shall mark for update the last data access timestamp of the directory each time the directory is  
 61303 actually read.

61304 After a call to *fork()*, either the parent or child (but not both) may continue processing the  
 61305 XSI directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes  
 61306 use these functions, the result is undefined.

61307 The *readdir()* function need not be thread-safe if concurrent calls are made for the same directory  
 61308 stream.

61309 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If  
 61310 *errno* is set to non-zero on return, an error occurred.

61311 OB The *readdir\_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the  
 61312 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer

61313 to this structure at the location referenced by *result*, and position the directory stream at the next  
61314 entry.

61315 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d\_name*  
61316 members containing at least {NAME\_MAX}+1 elements.

61317 Upon successful return, the pointer returned at *\*result* shall have the same value as the argument  
61318 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

61319 The *readdir\_r()* function shall not return directory entries containing empty names.

61320 If a file is removed from or added to the directory after the most recent call to *opendir()* or  
61321 *rewinddir()*, whether a subsequent call to *readdir\_r()* returns an entry for that file is unspecified.

61322 The *readdir\_r()* function may buffer several directory entries per actual read operation;  
61323 *readdir\_r()* shall mark for update the last data access timestamp of the directory each time the  
61324 directory is actually read.

#### 61325 RETURN VALUE

61326 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.  
61327 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate  
61328 the error. When the end of the directory is encountered, a null pointer shall be returned and  
61329 *errno* is not changed.

61330 OB If successful, the *readdir\_r()* function shall return zero; otherwise, an error number shall be  
61331 returned to indicate the error.

#### 61332 ERRORS

61333 OB The *readdir()* and *readdir\_r()* functions shall fail if:

61334 [EOVERFLOW] One of the values in the structure to be returned cannot be represented  
61335 correctly.

61336 [ENOMEM] Insufficient memory is available.

61337 OB The *readdir()* and *readdir\_r()* functions may fail if:

61338 [EBADF] The *dirp* argument does not refer to an open directory stream.

61339 [ENOENT] The current position of the directory stream is invalid.

#### 61340 EXAMPLES

61341 The following sample program searches the current directory for each of the arguments supplied  
61342 on the command line.

```
61343 #include <dirent.h>
61344 #include <errno.h>
61345 #include <stdio.h>
61346 #include <string.h>

61347 static void lookup(const char *arg)
61348 {
61349     DIR *dirp;
61350     struct dirent *dp;

61351     if ((dirp = opendir(".")) == NULL) {
61352         perror("couldn't open '.');
61353         return;
61354     }
61355     do {
```

```

61356         errno = 0;
61357         if ((dp = readdir(dirp)) != NULL) {
61358             if (strcmp(dp->d_name, arg) != 0)
61359                 continue;
61360
61361             (void) printf("found %s\n", arg);
61362             (void) closedir(dirp);
61363             return;
61364         }
61365     } while (dp != NULL);
61366
61367     if (errno != 0)
61368         perror("error reading directory");
61369     else
61370         (void) printf("failed to find %s\n", arg);
61371     (void) closedir(dirp);
61372     return;
61373 }
61374
61375 int main(int argc, char *argv[])
61376 {
61377     int i;
61378     for (i = 1; i < argc; i++)
61379         lookup(argv[i]);
61380     return (0);
61381 }

```

#### 61379 APPLICATION USAGE

61380 The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to  
61381 examine the contents of the directory.

61382 The *readdir\_r()* function returns values in a user-supplied buffer, but does not allow the size of  
61383 the buffer to be specified by the caller. If `[NAME_MAX]` is indeterminate, there is no way for an  
61384 application to know how large the buffer needs to be and *readdir\_r()* cannot safely be used.

#### 61385 RATIONALE

61386 The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be  
61387 inferred.

61388 Historical implementations of *readdir()* obtain multiple directory entries on a single read  
61389 operation, which permits subsequent *readdir()* operations to operate from the buffered  
61390 information. Any wording that required each successful *readdir()* operation to mark the  
61391 directory last data access timestamp for update would disallow such historical performance-  
61392 oriented implementations.

61393 When returning a directory entry for the root of a mounted file system, some historical  
61394 implementations of *readdir()* returned the file serial number of the underlying mount point,  
61395 rather than of the root of the mounted file system. This behavior is considered to be a bug, since  
61396 the underlying file serial number has no significance to applications.

61397 Since *readdir()* returns NULL when it detects an error and when the end of the directory is  
61398 encountered, an application that needs to tell the difference must set *errno* to zero before the call  
61399 and check it if NULL is returned. Since the function must not change *errno* in the second case  
61400 and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL  
61401 indicates end-of-directory; otherwise, an error.

61402 Routines to deal with this problem more directly were proposed:

```
61403 int derror (dirp)
```

```
61404 DIR *dirp;
```

```
61405 void clearderr (dirp)
```

```
61406 DIR *dirp;
```

61407 The first would indicate whether an error had occurred, and the second would clear the error  
61408 indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()*  
61409 not change *errno* when end-of-directory is encountered.

61410 An error or signal indicating that a directory has changed while open was considered but  
61411 rejected.

61412 Historically, *readdir()* returned a pointer to an internal static buffer that was overwritten by each  
61413 call. The *readdir\_r()* function was added as a thread-safe alternative that returns values in a user-  
61414 supplied buffer. However, it does not allow the size of the buffer to be specified by the caller,  
61415 and so is only usable if {NAME\_MAX} is a compile-time constant or *fpathconf()* with  
61416 *\_SC\_NAME\_MAX* returns a value other than -1. If {NAME\_MAX} is indeterminate (indicated  
61417 by *fpathconf()* returning -1), there is no way to reliably allocate a buffer large enough to hold a  
61418 filename being returned by *readdir\_r()*. Therefore, *readdir\_r()* has been marked obsolescent and  
61419 *readdir()* is now required to be thread safe as long as there are no concurrent calls to it on a  
61420 single directory stream.

61421 Conforming file systems are required to store filenames unaltered from how they were created  
61422 (via *open()*, *link()*, *mkdir()*, *mkfifo()*, *rename()*, etc.). By definition, a filename string does not  
61423 include a <slash>, even if a trailing <slash> was present in the pathname presented to *mkdir()*  
61424 when creating a sub-directory.

61425 However, there are non-conforming file systems where filenames are converted to a canonical  
61426 representation before a directory entry is created, such that it is possible to create a file using one  
61427 string, then perform *opendir()* and a *readdir()* loop and not encounter the same string, because  
61428 *readdir()* returns the canonical form of the string instead. Such non-conforming file systems also  
61429 have the issue that multiple filenames can resolve to the same directory entry, with potentially  
61430 confusing results. This standard cannot mandate the behavior of non-conforming file systems,  
61431 and strictly conforming applications need not worry about dealing with such file systems, but it  
61432 is a concern for developers of portable applications. Therefore, this standard recommends that  
61433 file system implementations that perform canonicalization of filenames should reject attempts to  
61434 create a directory entry with a non-canonical filename using the [EILSEQ] error. However, if a  
61435 directory entry already exists, it is reasonable for a file system to permit accessing that file via a  
61436 non-canonical filename.

#### 61437 FUTURE DIRECTIONS

61438 The *readdir\_r()* function may be removed in a future version.

#### 61439 SEE ALSO

61440 *closedir()*, *dirfd()*, *exec*, *fdopendir()*, *fstatat()*, *posix\_getdents()*, *rewinddir()*, *symlink()*

61441 XBD <[dirent.h](#)>, <[sys/types.h](#)>

#### 61442 CHANGE HISTORY

61443 First released in Issue 2.

61444 **Issue 5**

61445 Large File Summit extensions are added.

61446 The *readdir\_r()* function is included for alignment with the POSIX Threads Extension.

61447 A note indicating that the *readdir()* function need not be reentrant is added to the  
61448 DESCRIPTION.

61449 **Issue 6**

61450 The *readdir\_r()* function is marked as part of the Thread-Safe Functions option.

61451 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir\_r()*.

61452 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful  
61453 return for the *readdir\_r()* function.

61454 The following new requirements on POSIX implementations derive from alignment with the  
61455 Single UNIX Specification:

61456 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
61457 required for conforming implementations of previous POSIX specifications, it was not  
61458 required for UNIX applications.

61459 • A statement is added to the DESCRIPTION indicating the disposition of certain fields in  
61460 **struct dirent** when an entry refers to a symbolic link.

61461 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
61462 files.

61463 • The [ENOENT] optional error condition is added.

61464 The APPLICATION USAGE section is updated to include a note on the thread-safe function and  
61465 its avoidance of possibly using a static data area.

61466 The **restrict** keyword is added to the *readdir\_r()* prototype for alignment with the  
61467 ISO/IEC 9899:1999 standard.

61468 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES  
61469 section with a new example.

61470 **Issue 7**

61471 Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.

61472 Austin Group Interpretation 1003.1-2001 #156 is applied.

61473 The *readdir\_r()* function is moved from the Thread-Safe Functions option to the Base.

61474 Changes are made related to support for finegrained timestamps.

61475 The value of the *d\_ino* member is no longer unspecified for symbolic links.

61476 SD5-XSH-ERN-193 is applied.

61477 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0486 [75] is applied.

61478 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0304 [656] is applied.

61479 **Issue 8**

61480 Austin Group Defect 293 is applied, adding a requirement that *d\_name* contains the same byte  
61481 sequence as the last pathname component of the string used to create the directory entry.

61482 Austin Group Defects 696 and 1664 are applied, making *readdir\_r()* obsolescent, requiring  
61483 *readdir()* to be thread-safe except when concurrent calls are made for the same directory stream,

61484

and adding the [ENOMEM] error.

61485

Austin Group Defect 697 is applied, adding *posix\_getdents()* to the SEE ALSO section.

61486 **NAME**

61487 readlink, readlinkat — read the contents of a symbolic link

61488 **SYNOPSIS**

61489 #include &lt;unistd.h&gt;

61490 ssize\_t readlink(const char \*restrict path, char \*restrict buf,  
61491 size\_t bufsize);

61492 OH #include &lt;fcntl.h&gt;

61493 ssize\_t readlinkat(int fd, const char \*restrict path,  
61494 char \*restrict buf, size\_t bufsize);61495 **DESCRIPTION**61496 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the  
61497 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,  
61498 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to  
61499 contain the link content, the first *bufsize* bytes shall be placed in *buf*.61500 If the value of *bufsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.61501 Upon successful completion, *readlink()* shall mark for update the last data access timestamp of  
61502 the symbolic link.61503 The *readlinkat()* function shall be equivalent to the *readlink()* function except in the case where  
61504 *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the  
61505 directory associated with the file descriptor *fd* instead of the current working directory. If the  
61506 access mode of the open file description associated with the file descriptor is not O\_SEARCH,  
61507 the function shall check whether directory searches are permitted using the current permissions  
61508 of the directory underlying the file descriptor. If the access mode is O\_SEARCH, the function  
61509 shall not perform the check.61510 If *readlinkat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
61511 directory shall be used and the behavior shall be identical to a call to *readlink()*.61512 **RETURN VALUE**61513 Upon successful completion, these functions shall return the count of bytes placed in the buffer.  
61514 Otherwise, these functions shall return a value of -1, leave the buffer unchanged, and set *errno* to  
61515 indicate the error.61516 **ERRORS**

61517 These functions shall fail if:

61518 [EACCES] Search permission is denied for a component of the path prefix of *path*.61519 [EINVAL] The *path* argument names a file that is not a symbolic link.

61520 [EIO] An I/O error occurred while reading from the file system.

61521 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
61522 argument.

61523 [ENAMETOOLONG]

61524 The length of a component of a pathname is longer than {NAME\_MAX}.

61525 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.61526 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
61527 directory nor a symbolic link to a directory, or the *path* argument contains at  
61528 least one non-*<slash>* character and ends with one or more trailing *<slash>*



61529 characters and the last pathname component names an existing file that is  
61530 neither a directory nor a symbolic link to a directory.

61531 The *readlinkat()* function shall fail if:

61532 [EACCES] The access mode of the open file description associated with *fd* is not  
61533 O\_SEARCH and the permissions of the directory underlying *fd* do not permit  
61534 directory searches.

61535 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
61536 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.

61537 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated  
61538 with a non-directory file.

61539 These functions may fail if:

61540 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
61541 resolution of the *path* argument.

61542 [ENAMETOOLONG]  
61543 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
61544 symbolic link produced an intermediate result with a length that exceeds  
61545 {PATH\_MAX}.

## 61546 EXAMPLES

### 61547 Reading the Name of a Symbolic Link

61548 The following example shows how to read the name of a symbolic link named */modules/pass1*.

```
61549 #include <unistd.h>
61550 char buf[1024];
61551 ssize_t len;
61552 ...
61553 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
61554     buf[len] = '\0';
```

### 61555 APPLICATION USAGE

61556 Conforming applications should not assume that the returned contents of the symbolic link are  
61557 null-terminated.

### 61558 RATIONALE

61559 The type associated with *bufsiz* is a **size\_t** in order to be consistent with both the ISO C standard  
61560 and the definition of *read()*. The behavior specified for *readlink()* when *bufsiz* is zero represents  
61561 historical practice. For this case, the standard developers considered a change whereby *readlink()*  
61562 would return the number of non-null bytes contained in the symbolic link with the buffer *buf*  
61563 remaining unchanged; however, since the **stat** structure member *st\_size* value can be used to  
61564 determine the size of buffer necessary to contain the contents of the symbolic link as returned by  
61565 *readlink()*, this proposal was rejected, and the historical practice retained.

61566 The purpose of the *readlinkat()* function is to read the content of symbolic links in directories  
61567 other than the current working directory without exposure to race conditions. Any part of the  
61568 path of a file could be changed in parallel to a call to *readlink()*, resulting in unspecified behavior.  
61569 By opening a file descriptor for the target directory and using the *readlinkat()* function it can be  
61570 guaranteed that the symbolic link read is located relative to the desired directory.

61571 **FUTURE DIRECTIONS**

61572 None.

61573 **SEE ALSO**61574 *fstatat()*, *symlink()*61575 XBD [<fcntl.h>](#), [<unistd.h>](#)61576 **CHANGE HISTORY**

61577 First released in Issue 4, Version 2.

61578 **Issue 5**

61579 Moved from X/OPEN UNIX extension to BASE.

61580 **Issue 6**61581 The return type is changed to **ssize\_t**, to align with the IEEE P1003.1a draft standard.61582 The following new requirements on POSIX implementations derive from alignment with the  
61583 Single UNIX Specification:

- 61584 • This function is made mandatory.
- 61585 • In this function it is possible for the return value to exceed the range of the type **ssize\_t**  
61586 (since **size\_t** has a larger range of positive values than **ssize\_t**). A sentence restricting the  
61587 size of the **size\_t** object is added to the description to resolve this conflict.

61588 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 61589 • The FUTURE DIRECTIONS section is changed to None.

61590 The following changes were made to align with the IEEE P1003.1a draft standard:

- 61591 • The [ELOOP] optional error condition is added.

61592 The **restrict** keyword is added to the *readlink()* prototype for alignment with the  
61593 ISO/IEC 9899: 1999 standard.61594 **Issue 7**

61595 Austin Group Interpretation 1003.1-2001 #143 is applied.

61596 SD5-XSH-ERN-189 is applied, updating the ERRORS section.

61597 The *readlinkat()* function is added from The Open Group Technical Standard, 2006, Extended  
61598 API Set Part 2.

61599 The [EACCES] error is removed from the “may fail” error conditions.

61600 Changes are made to allow a directory to be opened for searching.

61601 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
61602 pathname exists but is not a directory or a symbolic link to a directory.61603 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0487 [120], XSH/TC1-2008/0488 [461],  
61604 XSH/TC1-2008/0489 [143], XSH/TC1-2008/0490 [324], XSH/TC1-2008/0491 [278],  
61605 XSH/TC1-2008/0492 [278], XSH/TC1-2008/0493 [455], and XSH/TC1-2008/0494 [151,231] are  
61606 applied.61607 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0305 [591], XSH/TC2-2008/0306 [817],  
61608 XSH/TC2-2008/0307 [817], and XSH/TC2-2008/0308 [591] are applied.

61609 **NAME**

61610 readv — read a vector

61611 **SYNOPSIS**

```
61612 XSI #include <sys/uio.h>
61613      ssize_t readv(int fd, const struct iovec *iov, int iovcnt);
```

61614 **DESCRIPTION**

61615 The *readv()* function shall be equivalent to *read()*, except as described below. The *readv()*  
 61616 function shall place the input data into the *iovcnt* buffers specified by the members of the *iov*  
 61617 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than  
 61618 or equal to {IOV\_MAX}.

61619 Each *iovec* entry specifies the base address and length of an area in memory where data should  
 61620 be placed. The *readv()* function shall always fill an area completely before proceeding to the  
 61621 next.

61622 Upon successful completion, *readv()* shall mark for update the last data access timestamp of the  
 61623 file.

61624 **RETURN VALUE**61625 Refer to *read()*.61626 **ERRORS**61627 Refer to *read()*.61628 In addition, the *readv()* function shall fail if:61629 [EINVAL] The sum of the *iov\_len* values in the *iov* array overflowed an *ssize\_t*.61630 The *readv()* function may fail if:61631 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV\_MAX}.61632 **EXAMPLES**61633 **Reading Data into an Array**

61634 The following example reads data from the file associated with the file descriptor *fd* into the  
 61635 buffers specified by members of the *iov* array.

```
61636 #include <sys/types.h>
61637 #include <sys/uio.h>
61638 #include <unistd.h>
61639 ...
61640 ssize_t bytes_read;
61641 int fd;
61642 char buf0[20];
61643 char buf1[30];
61644 char buf2[40];
61645 int iovcnt;
61646 struct iovec iov[3];

61647 iov[0].iov_base = buf0;
61648 iov[0].iov_len = sizeof(buf0);
61649 iov[1].iov_base = buf1;
61650 iov[1].iov_len = sizeof(buf1);
61651 iov[2].iov_base = buf2;
```

```
61652     iov[2].iov_len = sizeof(buf2);
61653     ...
61654     iovcnt = sizeof(iov) / sizeof(struct iovec);
61655     bytes_read = readv(fd, iov, iovcnt);
61656     ...
```

**61657 APPLICATION USAGE**

61658 None.

**61659 RATIONALE**

61660 Refer to *read()*.

**61661 FUTURE DIRECTIONS**

61662 None.

**61663 SEE ALSO**

61664 *read()*, *writev()*

61665 XBD <sys/uio.h>

**61666 CHANGE HISTORY**

61667 First released in Issue 4, Version 2.

**61668 Issue 6**

61669 Split out from the *read()* reference page.

**61670 Issue 7**

61671 Changes are made related to support for finegrained timestamps.

61672 **NAME**

61673 realloc, reallocarray — memory reallocators

61674 **SYNOPSIS**

61675 #include &lt;stdlib.h&gt;

61676 void \*realloc(void \*ptr, size\_t size);

61677 CX void \*reallocarray(void \*ptr, size\_t nelem, size\_t elsize);

61678 **DESCRIPTION**

61679 CX For `realloc()`: The functionality described on this reference page is aligned with the ISO C  
 61680 standard. Any conflict between the requirements described here and the ISO C standard is  
 61681 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

61682 The `realloc()` function shall deallocate the old object pointed to by `ptr` and return a pointer to a  
 61683 new object that has the size specified by `size`. The contents of the new object shall be the same as  
 61684 that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in  
 61685 the new object beyond the size of the old object have indeterminate values.

61686 CX The `reallocarray()` function shall be equivalent to the call `realloc(ptr, nelem * elsize)`  
 61687 except that overflow in the multiplication shall be an error.

61688 CX If `ptr` is a null pointer, `realloc()` or `reallocarray()` shall be equivalent to `malloc()` for the specified  
 61689 size. Otherwise, if `ptr` does not match a pointer earlier returned by `aligned_alloc()`, `calloc()`,  
 61690 ADV `malloc()`, `posix_memalign()`, `realloc()`,  
 61691 CX `reallocarray()`, or a function in POSIX.1-2024 that allocates memory as if by `malloc()`, or if the  
 61692 CX space has been deallocated by a call to `free()`, `reallocarray()`, or `realloc()`, the behavior is  
 61693 undefined.

61694 If `size` is non-zero and memory for the new object is not allocated, the old object shall not be  
 61695 deallocated.

61696 CX The order and contiguity of storage allocated by successive calls to `realloc()` or `reallocarray()` is  
 61697 unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it  
 61698 may be assigned to a pointer to any type of object with a fundamental alignment requirement  
 61699 and then used to access such an object in the space allocated (until the space is explicitly freed or  
 61700 reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object.  
 61701 The pointer returned shall point to the start (lowest byte address) of the allocated space. If the  
 61702 space cannot be allocated, a null pointer shall be returned.

61703 CX For purposes of determining the existence of a data race, `realloc()` and `reallocarray()` shall each  
 61704 behave as though it accessed only memory locations accessible through its argument and not  
 61705 other static duration storage. The function may, however, visibly modify the storage that it  
 61706 ADV allocates. Calls to `aligned_alloc()`, `calloc()`, `free()`, `malloc()`, `posix_memalign()`,  
 61707 CX `reallocarray()`, and `realloc()` that allocate or deallocate a particular region of memory shall occur  
 61708 in a single total order (see [Section 4.15.1](#), on page 100), and each such deallocation call shall  
 61709 synchronize with the next allocation (if any) in this order.

61710 **RETURN VALUE**

61711 CX Upon successful completion, `realloc()` and `reallocarray()` shall return a pointer to the new object  
 61712 (which can have the same value as a pointer to the old object), or a null pointer if the new object  
 61713 has not been allocated.

61714 OB If `size` is 0,61715 OB CX or either `nelem` or `elsize` is 0,

61716 OB either:

- 61717 OB • A null pointer shall be returned
- 61718 OB CX and, if *ptr* is not a null pointer, *errno* shall be set to [EINVAL].
- 61719 OB • A pointer to the allocated space shall be returned, and the memory object pointed to by *ptr*
- 61720 shall be freed. The application shall ensure that the pointer is not used to access an object.
- 61721 CX If there is not enough available memory, *realloc()* and *reallocarray()* shall return a null pointer
- 61722 CX and set *errno* to [ENOMEM].
- 61723 **ERRORS**
- 61724 CX The *realloc()* and *reallocarray()* functions shall fail if:
- 61725 CX [ENOMEM] Insufficient memory is available.
- 61726 CX The *reallocarray()* function shall fail if:
- 61727 [ENOMEM] The calculation *nelem \* elsize* would overflow.
- 61728 CX The *realloc()* and *reallocarray()* functions may fail if:
- 61729 CX [EINVAL] The requested allocation size is 0 and the implementation does not support 0
- 61730 sized allocations.
- 61731 **EXAMPLES**
- 61732 None.
- 61733 **APPLICATION USAGE**
- 61734 The ISO C standard makes it implementation-defined whether a call to *realloc(p, 0)* frees the
- 61735 space pointed to by *p* if it returns a null pointer because memory for the new object was not
- 61736 allocated. POSIX.1 instead requires that implementations set *errno* if a null pointer is returned
- 61737 and the space has not been freed, and POSIX applications should only free the space if *errno* was
- 61738 changed.
- 61739 **RATIONALE**
- 61740 See the RATIONALE for *malloc()*.
- 61741 **FUTURE DIRECTIONS**
- 61742 The ISO C standard states that invoking *realloc()* with a *size* argument equal to zero is an
- 61743 obsolescent feature. This feature may be removed in a future version of this standard.
- 61744 **SEE ALSO**
- 61745 *aligned\_alloc()*, *calloc()*, *free()*, *malloc()*
- 61746 XBD <stdlib.h>
- 61747 **CHANGE HISTORY**
- 61748 First released in Issue 1. Derived from Issue 1 of the SVID.
- 61749 **Issue 6**
- 61750 Extensions beyond the ISO C standard are marked.
- 61751 The following new requirements on POSIX implementations derive from alignment with the
- 61752 Single UNIX Specification:
- 61753 • In the RETURN VALUE section, if there is not enough available memory, the setting of
- 61754 *errno* to [ENOMEM] is added.
- 61755 • The [ENOMEM] error condition is added.

61756 **Issue 7**

61757 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0495 [400], XSH/TC1-2008/0496 [400],  
61758 XSH/TC1-2008/0497 [400], and XSH/TC1-2008/0498 [400] are applied.

61759 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0309 [526] and XSH/TC2-2008/0310  
61760 [526,688] are applied.

61761 **Issue 8**

61762 Austin Group Defect 374 is applied, adding the [EINVAL] error.

61763 Austin Group Defect 1218 is applied, adding *reallocarray()*.

61764 Austin Group Defect 1302 is applied, aligning the *realloc()* function with the ISO/IEC 9899:2018  
61765 standard.

61766 Austin Group Defect 1387 is applied, changing the RATIONALE section.

61767 **NAME**

61768            realpath — resolve a pathname

61769 **SYNOPSIS**

61770            #include &lt;stdlib.h&gt;

61771            char \*realpath(const char \*restrict *file\_name*,  
61772                            char \*restrict *resolved\_name*);61773 **DESCRIPTION**

61774            The *realpath()* function shall derive, from the pathname pointed to by *file\_name*, an absolute  
61775            pathname that resolves to the same directory entry, whose resolution does not involve '.',  
61776            '..', or symbolic links. If *resolved\_name* is a null pointer, the generated pathname shall be  
61777            stored as a null-terminated string in a buffer allocated as if by a call to *malloc()*. Otherwise, if  
61778            {PATH\_MAX} is defined as a constant in the <limits.h> header, then the generated pathname  
61779            shall be stored as a null-terminated string, up to a maximum of {PATH\_MAX} bytes, in the  
61780            buffer pointed to by *resolved\_name*.

61781            If *resolved\_name* is not a null pointer and {PATH\_MAX} is not defined as a constant in the  
61782            <limits.h> header, the behavior is undefined.

61783 **RETURN VALUE**

61784            Upon successful completion, *realpath()* shall return a pointer to the buffer containing the  
61785            resolved name. Otherwise, *realpath()* shall return a null pointer and set *errno* to indicate the  
61786            error.

61787            If the *resolved\_name* argument is a null pointer, the pointer returned by *realpath()* can be passed  
61788            to *free()*.

61789            If the *resolved\_name* argument is not a null pointer and the *realpath()* function fails, the contents  
61790            of the buffer pointed to by *resolved\_name* are undefined.

61791 **ERRORS**61792            The *realpath()* function shall fail if:61793            [EACCES]            Search permission was denied for a component of the path prefix of *file\_name*.61794            [EINVAL]            The *file\_name* argument is a null pointer.

61795            [EIO]                An error occurred while reading from the file system.

61796            [ELOOP]            A loop exists in symbolic links encountered during resolution of the *file\_name*  
61797            argument.

61798            [ENAMETOOLONG]

61799                            The length of a component of a pathname is longer than {NAME\_MAX}.

61800            [ENOENT]            A component of *file\_name* does not name an existing file or *file\_name* points to  
61801            an empty string.61802            [ENOTDIR]           A component of the path prefix names an existing file that is neither a  
61803            directory nor a symbolic link to a directory, or the *file\_name* argument contains  
61804            at least one non-<slash> character and ends with one or more trailing <slash>  
61805            characters and the last pathname component names an existing file that is  
61806            neither a directory nor a symbolic link to a directory.



- 61807 The *realpath()* function may fail if:
- 61808 [EACCES] The *file\_name* argument does not begin with a <slash> and none of the
  - 61809 symbolic links (if any) processed during pathname resolution of *file\_name* had
  - 61810 contents that began with a <slash>, and either search permission was denied
  - 61811 for the current directory or read or search permission was denied for a
  - 61812 directory above the current directory in the file hierarchy.
  - 61813 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during
  - 61814 resolution of the *file\_name* argument.
  - 61815 [ENAMETOOLONG]
  - 61816 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a
  - 61817 symbolic link produced an intermediate result with a length that exceeds
  - 61818 {PATH\_MAX}.
  - 61819 [ENOMEM] Insufficient storage space is available.

## 61820 EXAMPLES

### 61821 Generating an Absolute Pathname

61822 The following example generates an absolute pathname for the file identified by the *symlinkpath*

61823 argument. The generated pathname is stored in the buffer pointed to by *actualpath*.

```
61824 #include <stdlib.h>
61825 ...
61826 char *symlinkpath = "/tmp/symlink/file";
61827 char *actualpath;
61828
61829 actualpath = realpath(symlinkpath, NULL);
61830 if (actualpath != NULL)
61831 {
61832     ... use actualpath ...
61833     free(actualpath);
61834 }
61835 else
61836 {
61837     ... handle error ...
61838 }
```

## 61838 APPLICATION USAGE

61839 For functions that allocate memory as if by *malloc()*, the application should release such memory

61840 when it is no longer required by a call to *free()*. For *realpath()*, this is the return value.

## 61841 RATIONALE

61842 Since *realpath()* has no *length* argument, if {PATH\_MAX} is not defined as a constant in

61843 <limits.h>, applications have no way of determining how large a buffer they need to allocate for

61844 it to be safe to pass to *realpath()*. A {PATH\_MAX} value obtained from a prior *pathconf()* call is

61845 out-of-date by the time *realpath()* is called. Hence the only reliable way to use *realpath()* when

61846 {PATH\_MAX} is not defined in <limits.h> is to pass a null pointer for *resolved\_name* so that

61847 *realpath()* will allocate a buffer of the necessary size.

61848 **FUTURE DIRECTIONS**

61849 None.

61850 **SEE ALSO**61851 *fpathconf()*, *free()*, *getcwd()*, *sysconf()*

61852 XBD &lt;limits.h&gt;, &lt;stdlib.h&gt;

61853 **CHANGE HISTORY**

61854 First released in Issue 4, Version 2.

61855 **Issue 5**

61856 Moved from X/OPEN UNIX extension to BASE.

61857 **Issue 6**61858 The **restrict** keyword is added to the *realpath()* prototype for alignment with the  
61859 ISO/IEC 9899:1999 standard.61860 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
61861 [ELOOP] error condition is added.61862 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the  
61863 DESCRIPTION for the case when *resolved\_name* is a null pointer, changing the [EINVAL] error  
61864 text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.61865 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/110 is applied, updating the ERRORS  
61866 section to refer to the *file\_name* argument, rather than a nonexistent *path* argument.61867 **Issue 7**

61868 Austin Group Interpretation 1003.1-2001 #143 is applied.

61869 This function is updated for passing a null pointer to *realpath()* for the *resolved\_name* argument.61870 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
61871 *malloc()*.61872 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0499 [353], XSH/TC1-2008/0500 [324],  
61873 and XSH/TC1-2008/0501 [353] are applied.61874 **Issue 8**61875 Austin Group Defect 1663 is applied, removing XSI shading from *realpath()*.

61876 **NAME**

61877       recv — receive a message from a connected socket

61878 **SYNOPSIS**

61879       #include &lt;sys/socket.h&gt;

61880       ssize\_t recv(int *socket*, void \**buffer*, size\_t *length*, int *flags*);61881 **DESCRIPTION**

61882       The *recv()* function shall receive a message from a connection-mode or connectionless-mode  
 61883       socket. It is normally used with connected sockets because it does not permit the application to  
 61884       retrieve the source address of received data.

61885       The *recv()* function takes the following arguments:

61886       *socket*       Specifies the socket file descriptor.

61887       *buffer*       Points to a buffer where the message should be stored.

61888       *length*       Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

61889       *flags*       Specifies the type of message reception. Values of this argument are formed by  
 61890       logically OR'ing zero or more of the following values:

61891               MSG\_PEEK       Peeks at an incoming message. The data is treated as unread and  
 61892               the next *recv()* or similar function shall still return this data.

61893               MSG\_OOB       Requests out-of-band data. The significance and semantics of  
 61894               out-of-band data are protocol-specific.

61895               MSG\_WAITALL   On SOCK\_STREAM sockets this requests that the function block  
 61896               until the full amount of data can be returned. The function may  
 61897               return the smaller amount of data if the socket is a message-  
 61898               based socket, if a signal is caught, if the connection is terminated,  
 61899               if MSG\_PEEK was specified, or if an error is pending for the  
 61900               socket.

61901       The *recv()* function shall return the length of the message written to the buffer pointed to by the  
 61902       *buffer* argument. For message-based sockets, such as SOCK\_DGRAM and SOCK\_SEQPACKET,  
 61903       the entire message shall be read in a single operation. If a message is too long to fit in the  
 61904       supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess bytes shall be  
 61905       discarded. For stream-based sockets, such as SOCK\_STREAM, message boundaries shall be  
 61906       ignored. In this case, data shall be returned to the user as soon as it becomes available, and no  
 61907       data shall be discarded.

61908       If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 61909       message.

61910       If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 61911       descriptor, *recv()* shall block until a message arrives or a timeout occurs (see SO\_RCVTIMEO in  
 61912       Section 2.10.16, on page 554). If no messages are available at the socket and O\_NONBLOCK is  
 61913       set on the socket's file descriptor, *recv()* shall fail and set *errno* to [EAGAIN] or  
 61914       [EWOULDBLOCK].

61915 **RETURN VALUE**

61916       Upon successful completion, *recv()* shall return the length of the message in bytes. If no  
 61917       messages are available to be received and the peer has performed an orderly shutdown, *recv()*  
 61918       shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

61919 **ERRORS**61920 The *recv()* function shall fail if:

61921 [EAGAIN] or [EWOULDBLOCK]

61922 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
61923 to be received; or MSG\_OOB is set and no out-of-band data is available and  
61924 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
61925 not support blocking to await out-of-band data. See also SO\_RCVTIMEO in  
61926 [Section 2.10.16](#) (on page 554).61927 [EBADF] The *socket* argument is not a valid file descriptor.

61928 [ECONNRESET] A connection was forcibly closed by a peer.

61929 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any  
61930 data was available.

61931 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.

61932 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

61933 [ENOTSOCK] The *socket* argument does not refer to a socket.

61934 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.

61935 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
61936 transmission timeout on active connection.61937 The *recv()* function may fail if:

61938 [EIO] An I/O error occurred while reading from or writing to the file system.

61939 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

61940 [ENOMEM] Insufficient memory was available to fulfill the request.

61941 **EXAMPLES**

61942 None.

61943 **APPLICATION USAGE**61944 The *recv()* function is equivalent to *recvfrom()* with null pointer *address* and *address\_len*  
61945 arguments, and to *read()* if the *socket* argument refers to a socket and the *flags* argument is 0.61946 The *select()* and *poll()* functions can be used to determine when data is available to be received.61947 **RATIONALE**

61948 None.

61949 **FUTURE DIRECTIONS**

61950 None.

61951 **SEE ALSO**61952 [poll\(\)](#), [pselect\(\)](#), [read\(\)](#), [recvmsg\(\)](#), [recvfrom\(\)](#), [send\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#), [shutdown\(\)](#),  
61953 [socket\(\)](#), [write\(\)](#)61954 XBD [<sys/socket.h>](#)61955 **CHANGE HISTORY**

61956 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

61957 **Issue 7**

61958 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0502 [462] is applied.

61959 **Issue 8**

61960 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to  
61961 [Section 2.10.16](#) (on page 554).

61962 **NAME**

61963 recvfrom — receive a message from a socket

61964 **SYNOPSIS**

```
61965 #include <sys/socket.h>
61966 ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
61967                 int flags, struct sockaddr *restrict address,
61968                 socklen_t *restrict address_len);
```

61969 **DESCRIPTION**

61970 The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode  
 61971 socket. It is normally used with connectionless-mode sockets because it permits the application  
 61972 to retrieve the source address of received data.

61973 The *recvfrom()* function takes the following arguments:

61974	<i>socket</i>	Specifies the socket file descriptor.
61975	<i>buffer</i>	Points to the buffer where the message should be stored.
61976	<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
61977	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by 61978 logically OR'ing zero or more of the following values:
61979	MSG_PEEK	Peeks at an incoming message. The data is treated as unread 61980 and the next <i>recvfrom()</i> or similar function shall still return 61981 this data.
61982	MSG_OOB	Requests out-of-band data. The significance and semantics 61983 of out-of-band data are protocol-specific.
61984	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function 61985 block until the full amount of data can be returned. The 61986 function may return the smaller amount of data if the socket 61987 is a message-based socket, if a signal is caught, if the 61988 connection is terminated, if MSG_PEEK was specified, or if 61989 an error is pending for the socket.
61990	<i>address</i>	A null pointer, or points to a <b>sockaddr</b> structure in which the sending address 61991 is to be stored. The length and format of the address depend on the address 61992 family of the socket.
61993	<i>address_len</i>	Either a null pointer, if <i>address</i> is a null pointer, or a pointer to a <b>socklen_t</b> 61994 object which on input specifies the length of the supplied <b>sockaddr</b> structure, 61995 and on output specifies the length of the sending address.

61996 The *recvfrom()* function shall return the length of the message written to the buffer pointed to by  
 61997 RS the *buffer* argument. For message-based sockets, such as **SOCK\_RAW**, **SOCK\_DGRAM**, and  
 61998 **SOCK\_SEQPACKET**, the entire message shall be read in a single operation. If a message is too  
 61999 long to fit in the supplied buffer, and MSG\_PEEK is not set in the *flags* argument, the excess  
 62000 bytes shall be discarded. For stream-based sockets, such as **SOCK\_STREAM**, message  
 62001 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes  
 62002 available, and no data shall be discarded.

62003 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
 62004 message.

62005 Not all protocols provide the source address for messages. If the *address* argument is not a null

62006 pointer and the protocol provides the source address of messages, the source address of the  
 62007 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,  
 62008 and the length of this address shall be stored in the object pointed to by the *address\_len*  
 62009 argument.

62010 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,  
 62011 the stored address shall be truncated.

62012 If the *address* argument is not a null pointer and the protocol does not provide the source address  
 62013 of messages, the value stored in the object pointed to by *address* is unspecified.

62014 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
 62015 descriptor, *recvfrom()* shall block until a message arrives or a timeout occurs (see  
 62016 SO\_RCVTIMEO in Section 2.10.16, on page 554). If no messages are available at the socket and  
 62017 O\_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno* to  
 62018 [EAGAIN] or [EWOULDBLOCK].

#### 62019 RETURN VALUE

62020 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no  
 62021 messages are available to be received and the peer has performed an orderly shutdown,  
 62022 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the  
 62023 error.

#### 62024 ERRORS

62025 The *recvfrom()* function shall fail if:

62026 [EAGAIN] or [EWOULDBLOCK]

62027 The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
 62028 to be received; or MSG\_OOB is set and no out-of-band data is available and  
 62029 either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
 62030 not support blocking to await out-of-band data. See also SO\_RCVTIMEO in  
 62031 Section 2.10.16 (on page 554).

62032 [EBADF] The *socket* argument is not a valid file descriptor.

62033 [ECONNRESET] A connection was forcibly closed by a peer.

62034 [EINTR] A signal interrupted *recvfrom()* before any data was available.

62035 [EINVAL] The MSG\_OOB flag is set and no out-of-band data is available.

62036 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

62037 [ENOTSOCK] The *socket* argument does not refer to a socket.

62038 [EOPNOTSUPP] The specified flags are not supported for this socket type.

62039 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
 62040 transmission timeout on active connection.

62041 The *recvfrom()* function may fail if:

62042 [EIO] An I/O error occurred while reading from or writing to the file system.

62043 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

62044 [ENOMEM] Insufficient memory was available to fulfill the request.

62045 **EXAMPLES**

62046 None.

62047 **APPLICATION USAGE**62048 The *select()* and *poll()* functions can be used to determine when data is available to be received.

62049 For AF\_UNIX sockets, it is recommended that *address* points to a buffer of length greater than  
62050 `sizeof(struct sockaddr_un)` which has been initialized with null bytes. That way, even if  
62051 the implementation supports the use of all bytes of *sun\_path* without a terminating null byte, the  
62052 larger buffer guarantees that the *sun\_path* member can then be passed to other interfaces that  
62053 expect a null-terminated string. If no truncation occurred based on the input value of *address\_len*,  
62054 it is unspecified whether the returned *address\_len* will be `sizeof(struct sockaddr_un)`, or  
62055 merely a value at least as large as `offsetof(struct sockaddr_un, sun_path)` plus the  
62056 number of non-null bytes stored in *sun\_path*.

62057 **RATIONALE**

62058 None.

62059 **FUTURE DIRECTIONS**

62060 None.

62061 **SEE ALSO**

62062 *poll()*, *pselect()*, *read()*, *recv()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *setsockopt()*, *shutdown()*,  
62063 *socket()*, *write()*

62064 XBD <[sys/socket.h](#)>62065 **CHANGE HISTORY**

62066 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

62067 **Issue 7**

62068 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0503 [464] is applied.

62069 **Issue 8**

62070 Austin Group Defect 561 is applied, adding a paragraph about *sun\_path* to APPLICATION  
62071 USAGE.

62072 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to  
62073 [Section 2.10.16](#) (on page 554).

62074 Austin Group Defect 1565 is applied, changing the description of *address\_len*.



62075 **NAME**

62076       recvmsg — receive a message from a socket

62077 **SYNOPSIS**

62078       #include &lt;sys/socket.h&gt;

62079       ssize\_t recvmsg(int socket, struct msghdr \*message, int flags);

62080 **DESCRIPTION**62081       The *recvmsg()* function shall receive a message from a connection-mode or connectionless-mode  
62082 socket. It is normally used with connectionless-mode sockets because it permits the application  
62083 to retrieve the source address of received data.62084       The *recvmsg()* function takes the following arguments:62085       *socket*               Specifies the socket file descriptor.62086       *message*             Points to a **msghdr** structure, containing both the buffer to store the source  
62087 address and the buffers for the incoming message. The length and format of  
62088 the address depend on the address family of the socket. The *msg\_flags* member  
62089 is ignored on input, but may contain meaningful values on output.62090       *flags*               Specifies the type of message reception. Values of this argument are formed by  
62091 logically OR'ing zero or more of the following values:62092               MSG\_OOB       Requests out-of-band data. The significance and semantics  
62093 of out-of-band data are protocol-specific.

62094               MSG\_PEEK      Peeks at the incoming message.

62095               MSG\_WAITALL On SOCK\_STREAM sockets this requests that the function  
62096 block until the full amount of data can be returned. The  
62097 function may return the smaller amount of data if the socket  
62098 is a message-based socket, if a signal is caught, if the  
62099 connection is terminated, if MSG\_PEEK was specified, or if  
62100 an error is pending for the socket.

62101               MSG\_CMSG\_CLOEXEC

62102               On sockets that permit a *cmsg\_type* of SCM\_RIGHTS in the  
62103 *msg\_control* ancillary data as a means of copying file  
62104 descriptors into the process, the file descriptors shall be  
62105 created with the FD\_CLOEXEC flag atomically set.

62106               MSG\_CMSG\_CLOFORK

62107               On sockets that permit a *cmsg\_type* of SCM\_RIGHTS in the  
62108 *msg\_control* ancillary data as a means of copying file  
62109 descriptors into the process, the file descriptors shall be  
62110 created with the FD\_CLOFORK flag atomically set.62111       The *recvmsg()* function shall receive messages from unconnected or connected sockets and shall  
62112 return the length of the message.62113       The *recvmsg()* function shall return the total length of the message. For message-based sockets,  
62114 such as SOCK\_DGRAM and SOCK\_SEQPACKET, the entire message shall be read in a single  
62115 operation. If a message is too long to fit in the supplied buffers, and MSG\_PEEK is not set in the  
62116 *flags* argument, the excess bytes shall be discarded, and MSG\_TRUNC shall be set in the  
62117 *msg\_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK\_STREAM,  
62118 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it  
62119 becomes available, and no data shall be discarded.

62120 If the MSG\_WAITALL flag is not set, data shall be returned only up to the end of the first  
62121 message.

62122 If no messages are available at the socket and O\_NONBLOCK is not set on the socket's file  
62123 descriptor, *recvmsg()* shall block until a message arrives or a timeout occurs (see SO\_RCVTIMEO  
62124 in Section 2.10.16, on page 554). If no messages are available at the socket and O\_NONBLOCK  
62125 is set on the socket's file descriptor, the *recvmsg()* function shall fail and set *errno* to [EAGAIN]  
62126 or [EWOULDBLOCK].

62127 In the **msghdr** structure, the *msg\_name* member may be a null pointer if the source address is not  
62128 required. Otherwise, if the socket is unconnected, the *msg\_name* member points to a **sockaddr**  
62129 structure in which the source address is to be stored, and the *msg\_namelen* member on input  
62130 specifies the length of the supplied **sockaddr** structure and on output specifies the length of the  
62131 source address. If the actual length of the address is greater than the length of the supplied  
62132 **sockaddr** structure, the stored address shall be truncated. If the socket is connected, the  
62133 *msg\_name* and *msg\_namelen* members shall be ignored. The *msg\_iov* and *msg\_iovlen* fields are  
62134 used to specify where the received data shall be stored. The *msg\_iov* member points to an array  
62135 of **iovec** structures; the *msg\_iovlen* member shall be set to the dimension of this array. In each  
62136 **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field gives its size in  
62137 bytes. Each storage area indicated by *msg\_iov* is filled with received data in turn until all of the  
62138 received data is stored or all of the areas have been filled.

62139 Upon successful completion, the *msg\_flags* member of the message header shall be the bitwise-  
62140 inclusive OR of all of the following flags that indicate conditions detected for the received  
62141 message:

62142 MSG\_EOR        End-of-record was received (if supported by the protocol).

62143 MSG\_OOB        Out-of-band data was received.

62144 MSG\_TRUNC      Normal data was truncated.

62145 MSG\_CTRUNC    Control data was truncated.

#### 62146 RETURN VALUE

62147 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no  
62148 messages are available to be received and the peer has performed an orderly shutdown,  
62149 *recvmsg()* shall return 0. Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

#### 62150 ERRORS

62151 The *recvmsg()* function shall fail if:

62152 [EAGAIN] or [EWOULDBLOCK]

62153        The socket's file descriptor is marked O\_NONBLOCK and no data is waiting  
62154        to be received; or MSG\_OOB is set and no out-of-band data is available and  
62155        either the socket's file descriptor is marked O\_NONBLOCK or the socket does  
62156        not support blocking to await out-of-band data. See also SO\_RCVTIMEO in  
62157        Section 2.10.16 (on page 554).

62158 [EBADF]        The *socket* argument is not a valid open file descriptor.

62159 [ECONNRESET]   A connection was forcibly closed by a peer.

62160 [EINTR]        This function was interrupted by a signal before any data was available.

62161 [EINVAL]        The sum of the *iov\_len* values overflows a **ssize\_t**, or the MSG\_OOB flag is set  
62162        and no out-of-band data is available.

- 62163 [EMSGSIZE] The *msg\_iovlen* member of the **msg\_hdr** structure pointed to by *message* is less  
62164 than or equal to 0, or is greater than {IOV\_MAX}.
- 62165 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.
- 62166 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 62167 [EOPNOTSUPP] The specified flags are not supported for this socket type.
- 62168 [ETIMEDOUT] The connection timed out during connection establishment, or due to a  
62169 transmission timeout on active connection.

62170 The *recvmsg()* function may fail if:

- 62171 [EIO] An I/O error occurred while reading from or writing to the file system.
- 62172 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 62173 [ENOMEM] Insufficient memory was available to fulfill the request.

#### 62174 EXAMPLES

62175 None.

#### 62176 APPLICATION USAGE

62177 The *select()* and *poll()* functions can be used to determine when data is available to be received.

#### 62178 RATIONALE

62179 The use of the MSG\_CMSG\_CLOEXEC and MSG\_CMSG\_CLOFORK flags to *recvmsg()* when  
62180 using SCM\_RIGHTS to receive file descriptors via ancillary data is necessary to avoid a data race  
62181 in multi-threaded applications. Without MSG\_CMSG\_CLOFORK, a file descriptor is leaked into  
62182 a child process created by one thread in the window between another thread calling *recvmsg()*  
62183 and using *fcntl()* to set the FD\_CLOFORK flag. Without MSG\_CMSG\_CLOEXEC, a file  
62184 descriptor intentionally inherited by child processes is similarly leaked into an executed  
62185 program if FD\_CLOEXEC is not set atomically.

#### 62186 FUTURE DIRECTIONS

62187 None.

#### 62188 SEE ALSO

62189 *poll()*, *pselect()*, *recv()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *setsockopt()*, *shutdown()*, *socket()*

62190 XBD <sys/socket.h>

#### 62191 CHANGE HISTORY

62192 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

#### 62193 Issue 7

62194 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0504 [464] is applied.

#### 62195 Issue 8

62196 Austin Group Defects 411 and 1318 are applied, adding MSG\_CMSG\_CLOEXEC and  
62197 MSG\_CMSG\_CLOFORK.

62198 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to  
62199 [Section 2.10.16](#) (on page 554).

62200 Austin Group Defect 1565 is applied, changing the description of *msg\_namelen*.

62201 **NAME**  
 62202 regcomp, regerror, regex, regfree — regular expression matching

62203 **SYNOPSIS**  
 62204 #include <regex.h>  
 62205 int regcomp(regex\_t \*restrict preg, const char \*restrict pattern,  
 62206 int cflags);  
 62207 size\_t regerror(int errcode, const regex\_t \*restrict preg,  
 62208 char \*restrict errbuf, size\_t errbuf\_size);  
 62209 int regex(const regex\_t \*restrict preg, const char \*restrict string,  
 62210 size\_t nmatch, regmatch\_t pmatch[restrict], int eflags);  
 62211 void regfree(regex\_t \*preg);

62212 **DESCRIPTION**  
 62213 These functions interpret *basic* and *extended* regular expressions as described in XBD [Chapter 9](#)  
 62214 (on page 179).

62215 The **regex\_t** structure is defined in <regex.h> and contains at least the following member:

Member Type	Member Name	Description
size_t	re_nsub	Number of parenthesized subexpressions.

62219 The **regmatch\_t** structure is defined in <regex.h> and contains at least the following members:

Member Type	Member Name	Description
regoff_t	rm_so	Byte offset from start of <i>string</i> to start of substring.
regoff_t	rm_eo	Byte offset from start of <i>string</i> of the first character after the end of substring.

62225 The *regcomp()* function shall compile the regular expression contained in the string pointed to by  
 62226 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*  
 62227 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in  
 62228 the <regex.h> header:

- 62229 REG\_EXTENDED Use Extended Regular Expressions.
- 62230 REG\_ICASE Perform matching in a case-insensitive manner (see XBD [Section 9.2](#), on  
 62231 page 180).
- 62232 REG\_MINIMAL Change the matching behavior for duplication symbols to the leftmost  
 62233 shortest possible match, and invert the behavior of the repetition modifier  
 62234 '?' (<question-mark>) to match the longest possible match instead of the  
 62235 shortest. Only applicable to REG\_EXTENDED regular expressions.
- 62236 REG\_NOSUB Report only success/fail in *regex()*.
- 62237 REG\_NEWLINE Change the handling of <newline> characters, as described in the text.

62238 The default regular expression type for *pattern* is a Basic Regular Expression. The application can  
 62239 specify Extended Regular Expressions using the REG\_EXTENDED *cflags* flag.

62240 If the REG\_NOSUB flag was not set in *cflags*, then *regcomp()* shall set *re\_nsub* to the number of  
 62241 parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "()" in  
 62242 extended regular expressions) found in *pattern*.

62243 The *regex()* function compares the null-terminated string specified by *string* with the compiled

62244 regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regexec()*  
 62245 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The  
 62246 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are  
 62247 defined in the `<regex.h>` header:

62248 REG\_NOTBOL        The first character of the string pointed to by *string* is not the beginning of  
 62249 the line. Therefore, the `<circumflex>` character ('^'), when taken as a  
 62250 special character, shall not match the beginning of *string*.

62251 REG\_NOTEOL        The last character of the string pointed to by *string* is not the end of the  
 62252 line. Therefore, the `<dollar-sign>` ('\$'), when taken as a special character,  
 62253 shall not match the end of *string*.

62254 If *nmatch* is 0 or REG\_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexec()* shall  
 62255 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument  
 62256 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that  
 62257 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions  
 62258 of *pattern*: *pmatch[i].rm\_so* shall be the byte offset of the beginning and *pmatch[i].rm\_eo* shall be  
 62259 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th  
 62260 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that  
 62261 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*  
 62262 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself  
 62263 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first  
 62264 *nmatch* substrings.

62265 When matching a basic or extended regular expression, any given parenthesized subexpression  
 62266 of *pattern* might participate in the match of several different substrings of *string*, or it might not  
 62267 match any substring even though the pattern as a whole did match. The following rules shall be  
 62268 used to determine which substrings to report in *pmatch* when matching regular expressions:

62269        1. If subexpression *i* in a regular expression is not contained within another subexpression,  
 62270        and it participated in the match several times, then the byte offsets in *pmatch[i]* shall  
 62271        delimit the last such match.

62272        2. If subexpression *i* is not contained within another subexpression, and it did not  
 62273        participate in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A  
 62274        subexpression does not participate in the match when:

62275                '\*' or "\{\}" appears immediately after the subexpression in a basic regular  
 62276                expression, or '\*', '?', or "{ }" appears immediately after the subexpression in  
 62277                an extended regular expression, and the subexpression did not match (matched 0  
 62278                times)

62279        or:

62280                '| ' is used in an extended regular expression to select this subexpression or  
 62281                another, and the other subexpression matched.

62282        3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained  
 62283        within any other subexpression that is contained within *j*, and a match of subexpression *j*  
 62284        is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in  
 62285        *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in  
 62286        *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start  
 62287        of *string*.

- 62288 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are  $-1$ ,  
 62289 then the pointers in *pmatch[i]* shall also be  $-1$ .
- 62290 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be  
 62291 the byte offset of the character or null terminator immediately following the zero-length  
 62292 string.

62293 If, when *regexexec()* is called, the locale is different from when the regular expression was  
 62294 compiled, the result is undefined.

62295 If REG\_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an  
 62296 ordinary character. If REG\_NEWLINE is set, then <newline> shall be treated as an ordinary  
 62297 character except as follows:

- 62298 1. A <newline> in *string* shall not be matched by a <period> outside a bracket expression or  
 62299 by any form of a non-matching list (see XBD Chapter 9, on page 179).
- 62300 2. A <circumflex> ('^') in *pattern*, when used to specify expression anchoring (see XBD  
 62301 Section 9.3.8, on page 186), shall match the zero-length string immediately after a  
 62302 <newline> in *string*, regardless of the setting of REG\_NOTBOL.
- 62303 3. A <dollar-sign> ('\$') in *pattern*, when used to specify expression anchoring, shall match  
 62304 the zero-length string immediately before a <newline> in *string*, regardless of the setting  
 62305 of REG\_NOTEOL.

62306 The *regfree()* function shall free any memory allocated by *regcomp()* associated with *preg*. The  
 62307 *regfree()* function shall not modify *errno* if *preg* was previously returned by *regcomp()* and not yet  
 62308 freed.

62309 The following constants are defined as the minimum set of error return values, although other  
 62310 errors listed as implementation extensions in <regex.h> are possible:

62311	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than
62312		two numbers, first larger than second.
62313	REG_BADPAT	Invalid regular expression.
62314	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
62315	REG_EBRACE	"\{\}" imbalance.
62316	REG_EBRACK	"[]" imbalance.
62317	REG_ECOLLATE	Invalid collating element referenced.
62318	REG_ECTYPE	Invalid character class type referenced.
62319	REG_EESCAPE	Trailing <backslash> character in pattern.
62320	REG_EPAREN	"\(\)" or "()" imbalance.
62321	REG_ERANGE	Invalid endpoint in range expression.
62322	REG_ESPACE	Out of memory.
62323	REG_ESUBREG	Number in "\digit" invalid or in error.
62324	REG_NOMATCH	<i>regexexec()</i> failed to match.

62325 If more than one error occurs in processing a function call, any one of the possible constants may  
 62326 be returned, as the order of detection is unspecified.

62327 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and

62328 *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the  
 62329 *errcode* argument, which the application shall ensure is the last non-zero value returned by  
 62330 *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of  
 62331 the generated string is unspecified.

62332 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*,  
 62333 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not  
 62334 be as detailed under some implementations.

62335 If the *errbuf\_size* argument is not 0, *regerror()* shall place the generated string into the buffer of  
 62336 size *errbuf\_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit  
 62337 in the buffer, *regerror()* shall truncate the string and null-terminate the result.

62338 If *errbuf\_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer  
 62339 needed to hold the generated string.

62340 If the *preg* argument to *regexec()* or *regfree()* is not a compiled regular expression returned by  
 62341 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression  
 62342 after it is given to *regfree()*.

#### 62343 RETURN VALUE

62344 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an  
 62345 integer value indicating an error as described in <**regex.h**>, and the content of *preg* is undefined.  
 62346 If a code is returned, the interpretation shall be as given in <**regex.h**>.

62347 If *regcomp()* detects an invalid RE, it may return REG\_BADPAT, or it may return one of the error  
 62348 codes that more precisely describes the error.

62349 Upon successful completion, the *regexec()* function shall return 0. Otherwise, it shall return  
 62350 REG\_NOMATCH to indicate no match.

62351 Upon successful completion, the *regerror()* function shall return the number of bytes needed to  
 62352 hold the entire generated string, including the null termination. If the return value is greater  
 62353 than *errbuf\_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

62354 The *regfree()* function shall not return a value.

#### 62355 ERRORS

62356 No errors are defined.

#### 62357 EXAMPLES

```
62358 #include <regex.h>
62359 /*
62360  * Match string against the extended regular expression in
62361  * pattern, treating errors as no match.
62362  *
62363  * Return 1 for match, 0 for no match.
62364  */
62365 int
62366 match(const char *string, char *pattern)
62367 {
62368     int    status;
62369     regex_t re;
62370     if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
62371         return(0);    /* Report error. */

```

```

62372     }
62373     status = regexec(&re, string, (size_t) 0, NULL, 0);
62374     regfree(&re);
62375     if (status != 0) {
62376         return(0);        /* Report error. */
62377     }
62378     return(1);
62379 }

```

62380 The following demonstrates how the REG\_NOTBOL flag could be used with *regexec()* to find all  
62381 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very  
62382 little error checking is done.)

```

62383 (void) regcomp (&re, pattern, 0);
62384 /* This call to regexec() finds the first match on the line. */
62385 error = regexec (&re, &buffer[0], 1, &pm, 0);
62386 while (error == 0) { /* While matches found. */
62387     /* Substring found between pm.rm_so and pm.rm_eo. */
62388     /* This call to regexec() finds the next match. */
62389     error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
62390 }

```

#### 62391 APPLICATION USAGE

62392 An application could use:

```
62393 regerror(code, preg, (char *)NULL, (size_t)0)
```

62394 to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the  
62395 string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed,  
62396 static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger  
62397 buffer if it finds that this is too small.

62398 To match a pattern as described in XCU [Section 2.14](#) (on page 2523), use the *fnmatch()* function.

#### 62399 RATIONALE

62400 The *regexec()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are  
62401 supplied by the application, even if some elements of *pmatch* do not correspond to  
62402 subexpressions in *pattern*. The application developer should note that there is probably no  
62403 reason for using a value of *nmatch* that is larger than *preg->re\_nsub*+1.

62404 The REG\_NEWLINE flag supports a use of RE matching that is needed in some applications like  
62405 text editors. In such applications, the user supplies an RE asking the application to find a line  
62406 that matches the given expression. An anchor in such an RE anchors at the beginning or end of  
62407 any line. Such an application can pass a sequence of <newline>-separated lines to *regexec()* as a  
62408 single long string and specify REG\_NEWLINE to *regcomp()* to get the desired behavior. The  
62409 application must ensure that there are no explicit <newline> characters in *pattern* if it wants to  
62410 ensure that any match occurs entirely within a single line.

62411 The REG\_NEWLINE flag affects the behavior of *regexec()*, but it is in the *cflags* parameter to  
62412 *regcomp()* to allow flexibility of implementation. Some implementations will want to generate  
62413 the same compiled RE in *regcomp()* regardless of the setting of REG\_NEWLINE and have  
62414 *regexec()* handle anchors differently based on the setting of the flag. Other implementations will  
62415 generate different compiled REs based on the REG\_NEWLINE.

62416 The REG\_ICASE flag supports the operations taken by the *grep -i* option and the historical  
62417 implementations of *ex* and *vi*. Including this flag will make it easier for application code to be  
62418 written that does the same thing as these utilities.



62419 The substrings reported in *pmatch*[] are defined using offsets from the start of the string rather  
62420 than pointers. This allows type-safe access to both constant and non-constant strings.

62421 The type **regoff\_t** is used for the elements of *pmatch*[] to ensure that the application can  
62422 represent large arrays in memory (important for an application conforming to the Shell and  
62423 Utilities volume of POSIX.1-2024).

62424 The 1992 edition of this standard required **regoff\_t** to be at least as wide as **off\_t**, to facilitate  
62425 future extensions in which the string to be searched is taken from a file. However, these future  
62426 extensions have not appeared. The requirement rules out popular implementations with 32-bit  
62427 **regoff\_t** and 64-bit **off\_t**, so it has been removed.

62428 The standard developers rejected the inclusion of a *regsub*() function that would be used to do  
62429 substitutions for a matched RE. While such a routine would be useful to some applications, its  
62430 utility would be much more limited than the matching function described here. Both RE parsing  
62431 and substitution are possible to implement without support other than that required by the  
62432 ISO C standard, but matching is much more complex than substituting. The only difficult part of  
62433 substitution, given the information supplied by *regexexec*(), is finding the next character in a string  
62434 when there can be multi-byte characters. That is a much larger issue, and one that needs a more  
62435 general solution.

62436 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for  
62437 this feature.

62438 The interface is defined so that the matched substrings *rm\_sp* and *rm\_ep* are in a separate  
62439 **regmatch\_t** structure instead of in **regex\_t**. This allows a single compiled RE to be used  
62440 simultaneously in several contexts; in *main*() and a signal handler, perhaps, or in multiple  
62441 threads of lightweight processes. (The *preg* argument to *regexexec*() is declared with type **const**, so  
62442 the implementation is not permitted to use the structure to store intermediate results.) It also  
62443 allows an application to request an arbitrary number of substrings from an RE. The number of  
62444 subexpressions in the RE is reported in *re\_nsub* in *preg*. With this change to *regexexec*(),  
62445 consideration was given to dropping the REG\_NOSUB flag since the user can now specify this  
62446 with a zero *nmatch* argument to *regexexec*(). However, keeping REG\_NOSUB allows an  
62447 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp*() that  
62448 no subexpressions need be reported. The implementation is only required to fill in *pmatch* if  
62449 *nmatch* is not zero and if REG\_NOSUB is not specified. Note that the **size\_t** type, as defined in  
62450 the ISO C standard, is unsigned, so the description of *regexexec*() does not need to address  
62451 negative values of *nmatch*.

62452 REG\_NOTBOL was added to allow an application to do repeated searches for the same pattern  
62453 in a line. If the pattern contains a <circumflex> character that should match the beginning of a  
62454 line, then the pattern should only match when matched against the beginning of the line.  
62455 Without the REG\_NOTBOL flag, the application could rewrite the expression for subsequent  
62456 matches, but in the general case this would require parsing the expression. The need for  
62457 REG\_NOTEOL is not as clear; it was added for symmetry.

62458 The addition of the *regerror*() function addresses the historical need for conforming application  
62459 programs to have access to error information more than “Function failed to compile/match your  
62460 RE for unknown reasons”.

62461 This interface provides for two different methods of dealing with error conditions. The specific  
62462 error codes (REG\_EBRACE, for example), defined in <**regex.h**>, allow an application to recover  
62463 from an error if it is so able. Many applications, especially those that use patterns supplied by a  
62464 user, will not try to deal with specific error cases, but will just use *regerror*() to obtain a human-  
62465 readable error message to present to the user.

62466 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating  
 62467 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was  
 62468 considered unacceptable since it creates difficulties for multi-threaded applications.

62469 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more  
 62470 descriptive message than would be possible with *errcode* alone. An implementation might, for  
 62471 example, save the character offset of the offending character of the pattern in a field of *preg*, and  
 62472 then include that in the generated message string. The implementation may also ignore *preg*.

62473 A REG\_FILENAME flag was considered, but omitted. This flag caused *regexec()* to match  
 62474 patterns as described in XCU Section 2.14 (on page 2523) instead of REs. This service is now  
 62475 provided by the *fnmatch()* function.

62476 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and  
 62477 POSIX.1-2024 in how to handle a “bad” regular expression. The ISO POSIX-2:1993 standard says  
 62478 that many bad constructs “produce undefined results”, or that “the interpretation is undefined”.  
 62479 POSIX.1-2024, however, says that the interpretation of such REs is unspecified. The term  
 62480 “undefined” means that the action by the application is an error, of similar severity to passing a  
 62481 bad pointer to a function.

62482 The *regcomp()* and *regexec()* functions are required to accept any null-terminated string as the  
 62483 *pattern* argument. If the meaning of the string is “undefined”, the behavior of the function is  
 62484 “unspecified”. POSIX.1-2024 does not specify how the functions will interpret the pattern; they  
 62485 might return error codes, or they might do pattern matching in some completely unexpected  
 62486 way, but they should not do something like abort the process.

#### 62487 FUTURE DIRECTIONS

62488 None.

#### 62489 SEE ALSO

62490 *fnmatch()*, *glob()*

62491 XBD Chapter 9 (on page 179), [<regex.h>](#), [<sys/types.h>](#)

62492 XCU Section 2.14 (on page 2523)

#### 62493 CHANGE HISTORY

62494 First released in Issue 4. Derived from the ISO POSIX-2 standard.

#### 62495 Issue 5

62496 Moved from POSIX2 C-language Binding to BASE.

#### 62497 Issue 6

62498 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

62499 The following new requirements on POSIX implementations derive from alignment with the  
 62500 Single UNIX Specification:

- 62501 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 62502 required for conforming implementations of previous POSIX specifications, it was not  
 62503 required for UNIX applications.

62504 The normative text is updated to avoid use of the term “must” for application requirements.

62505 The REG\_ENOSYS constant is removed.

62506 The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexec()* prototypes for  
 62507 alignment with the ISO/IEC 9899:1999 standard.

62508 **Issue 7**

62509 Austin Group Interpretation 1003.1-2001 #134 is applied, clarifying that if more than one error  
62510 occurs in processing a function call, any one of the possible constants may be returned.

62511 SD5-XBD-ERN-60 is applied.

62512 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0505 [305] is applied.

62513 **Issue 8**

62514 Austin Group Defect 385 is applied, adding a requirement that *regfree()* does not modify *errno*  
62515 when passed a pointer to a **regex\_t** that can be freed.

62516 Austin Group Defects 793 and 1329 are applied, adding REG\_MINIMAL.

62517 Austin Group Defect 1031 is applied, changing the description of REG\_ICASE.

62518 **NAME**

62519 remainder, remainderf, remainderl — remainder function

62520 **SYNOPSIS**

62521 #include &lt;math.h&gt;

62522 double remainder(double *x*, double *y*);62523 float remainderf(float *x*, float *y*);62524 long double remainderl(long double *x*, long double *y*);62525 **DESCRIPTION**

62526 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 62527 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62528 volume of POSIX.1-2024 defers to the ISO C standard.

62529 These functions shall return the floating-point remainder  $r=x-ny$  when  $y$  is non-zero. The value  
 62530  $n$  is the integral value nearest the exact value  $x/y$ . When  $|n-x/y|=\frac{1}{2}$ , the value  $n$  is chosen to  
 62531 be even.

62532 The behavior of *remainder()* shall be independent of the rounding mode.

62533 **RETURN VALUE**

62534 Upon successful completion, these functions shall return the floating-point remainder  $r=x-ny$   
 62535 when  $y$  is non-zero.

62536 MX When subnormal results are supported, the returned value shall be exact.

62537 On systems that do not support the IEC 60559 Floating-Point option, if  $y$  is zero, it is  
 62538 implementation-defined whether a domain error occurs or zero is returned.

62539 MX If  $x$  or  $y$  is NaN, a NaN shall be returned.

62540 If  $x$  is infinite or  $y$  is 0 and the other is non-NaN, a domain error shall occur, and a NaN shall be  
 62541 returned.

62542 **ERRORS**

62543 These functions shall fail if:

62544 MX **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ , or the  $y$  argument is  $\pm 0$  and the other argument is non-  
 62545 NaN.

62546 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62547 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 62548 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 62549 shall be raised.

62550 These functions may fail if:

62551 **Domain Error** The  $y$  argument is zero.

62552 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62553 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 62554 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 62555 shall be raised.

**62556 EXAMPLES**

62557 None.

**62558 APPLICATION USAGE**

62559 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
62560 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

**62561 RATIONALE**

62562 None.

**62563 FUTURE DIRECTIONS**

62564 None.

**62565 SEE ALSO**

62566 [abs\(\)](#), [div\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [ldiv\(\)](#)

62567 XBD Section 4.23 (on page 109), [<math.h>](#)

**62568 CHANGE HISTORY**

62569 First released in Issue 4, Version 2.

**62570 Issue 5**

62571 Moved from X/OPEN UNIX extension to BASE.

**62572 Issue 6**

62573 The *remainder()* function is no longer marked as an extension.

62574 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999  
62575 standard.

62576 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
62577 revised to align with the ISO/IEC 9899:1999 standard.

62578 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
62579 marked.

**62580 Issue 7**

62581 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

62582 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0506 [320] is applied.

**62583 Issue 8**

62584 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
62585 standard.

62586 **NAME**

62587 remove — remove a file

62588 **SYNOPSIS**

62589 #include &lt;stdio.h&gt;

62590 int remove(const char \*path);

62591 **DESCRIPTION**

62592 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
62593 conflict between the requirements described here and the ISO C standard is unintentional. This  
62594 volume of POSIX.1-2024 defers to the ISO C standard.

62595 The *remove()* function shall cause the file named by the pathname pointed to by *path* to be no  
62596 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,  
62597 unless it is created anew.

62598 CX If *path* does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

62599 If *path* names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

62600 **RETURN VALUE**62601 CX Refer to *rmdir()* or *unlink()*.62602 **ERRORS**62603 CX Refer to *rmdir()* or *unlink()*.62604 **EXAMPLES**62605 **Removing Access to a File**62606 The following example shows how to remove access to a file named `/home/cnd/old_mods`.

```
62607 #include <stdio.h>
62608 int status;
62609 ...
62610 status = remove("/home/cnd/old_mods");
```

62611 **APPLICATION USAGE**

62612 None.

62613 **RATIONALE**

62614 None.

62615 **FUTURE DIRECTIONS**

62616 None.

62617 **SEE ALSO**62618 *rmdir()*, *unlink()*

62619 XBD &lt;stdio.h&gt;

62620 **CHANGE HISTORY**

62621 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C  
62622 standard.

62623 **Issue 6**

62624 Extensions beyond the ISO C standard are marked.

62625  
62626

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

62627  
62628  
62629

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to *rmdir()*.

62630 **NAME**

62631 remque — remove an element from a queue

62632 **SYNOPSIS**

62633 XSI `#include <search.h>`

62634 `void remque(void *element);`

62635 **DESCRIPTION**

62636 Refer to *insque()*.



62637 **NAME**

62638 remquo, remquof, remquol — remainder functions

62639 **SYNOPSIS**

62640 #include &lt;math.h&gt;

62641 double remquo(double *x*, double *y*, int \**quo*);62642 float remquof(float *x*, float *y*, int \**quo*);62643 long double remquol(long double *x*, long double *y*, int \**quo*);62644 **DESCRIPTION**

62645 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 62646 conflict between the requirements described here and the ISO C standard is unintentional. This  
 62647 volume of POSIX.1-2024 defers to the ISO C standard.

62648 The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the  
 62649 *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by *quo*,  
 62650 they store a value whose sign is the sign of  $x/y$  and whose magnitude is congruent modulo  $2^n$  to  
 62651 the magnitude of the integral quotient of  $x/y$ , where  $n$  is an implementation-defined integer  
 62652 greater than or equal to 3. If  $y$  is zero, the value stored in the object pointed to by *quo* is  
 62653 unspecified.

62654 An application wishing to check for error situations should set *errno* to zero and call  
 62655 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 62656 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 62657 zero, an error has occurred.

62658 **RETURN VALUE**62659 These functions shall return  $x \text{ REM } y$ .

62660 MX When subnormal results are supported, the returned value shall be exact.

62661 If NaN is supported and a NaN is returned, the value stored in the object pointed to by *quo* is  
 62662 unspecified.

62663 On systems that do not support the IEC 60559 Floating-Point option, if  $y$  is zero, it is  
 62664 implementation-defined whether a domain error occurs or zero is returned.

62665 MX If  $x$  or  $y$  is NaN, a NaN shall be returned.

62666 If  $x$  is  $\pm\text{Inf}$  or  $y$  is zero and the other argument is non-NaN, a domain error shall occur, and a  
 62667 NaN shall be returned.

62668 **ERRORS**

62669 These functions shall fail if:

62670 MX **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ , or the  $y$  argument is  $\pm 0$  and the other argument is non-  
 62671 NaN.

62672 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62673 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 62674 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 62675 shall be raised.

62676 These functions may fail if:

62677 **Domain Error** The  $y$  argument is zero.

62678 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 62679 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 62680 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception

62681 shall be raised.

62682 **EXAMPLES**

62683 None.

62684 **APPLICATION USAGE**

62685 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
62686 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

62687 **RATIONALE**

62688 These functions are intended for implementing argument reductions which can exploit a few  
62689 low-order bits of the quotient. Note that *x* may be so large in magnitude relative to *y* that an  
62690 exact representation of the quotient is not practical.

62691 **FUTURE DIRECTIONS**

62692 None.

62693 **SEE ALSO**

62694 *feclearexcept()*, *fetestexcept()*, *remainder()*

62695 XBD Section 4.23 (on page 109), <math.h>

62696 **CHANGE HISTORY**

62697 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

62698 **Issue 7**

62699 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

62700 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0507 [320] is applied.

62701 **Issue 8**

62702 Austin Group Defect 713 is applied, clarifying the behavior when a NaN is returned.

62703 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
62704 standard.

62705 **NAME**

62706 rename, renameat — rename file

62707 **SYNOPSIS**

62708 #include &lt;stdio.h&gt;

62709 int rename(const char \*old, const char \*new);

62710 OH CX #include &lt;fcntl.h&gt;

62711 CX int renameat(int oldfd, const char \*old, int newfd,  
62712 const char \*new);62713 **DESCRIPTION**62714 CX For *rename()*: The functionality described on this reference page is aligned with the ISO C  
62715 standard. Any conflict between the requirements described here and the ISO C standard is  
62716 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.62717 The *rename()* function shall change the name of a file. The *old* argument points to the pathname  
62718 CX of the file to be renamed. The *new* argument points to the new pathname of the file. If the *new*  
62719 argument does not resolve to an existing directory entry for a file of type directory and the *new*  
62720 argument contains at least one non-`</code>slash> character and ends with one or more trailing  
62721 </code>slash> characters after all symbolic links have been processed, rename() shall fail.`62722 If either the *old* or *new* argument names a symbolic link, *rename()* shall operate on the symbolic  
62723 link itself, and shall not resolve the last component of the argument. If the *old* argument and the  
62724 *new* argument resolve to either the same existing directory entry or different directory entries for  
62725 the same existing file, *rename()* shall return successfully and perform no other action.62726 If the *old* argument names a file that is not a directory and the *new* argument names a directory,  
62727 or *old* names a directory and *new* names a file that is not a directory, or *new* names a directory  
62728 that is not empty, *rename()* shall fail. Otherwise, if the directory entry named by *new* exists, it  
62729 shall be removed and *old* renamed to *new*. In this case, a directory entry named *new* shall remain  
62730 visible to other threads throughout the renaming operation and refer either to the file referred to  
62731 by *new* or *old* before the operation began.62732 If either *pathname* argument refers to a path whose final component is either dot or dot-dot,  
62733 *rename()* shall fail.62734 If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed.  
62735 If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.62736 The *old* pathname shall not name an ancestor directory of the *new* pathname. Write access  
62737 permission is required for the directory containing *old* and the directory containing *new*. If the  
62738 *old* argument points to the pathname of a directory, write access permission may be required for  
62739 the directory named by *old*, and, if it exists, the directory named by *new*.62740 If the *new* argument names an existing file and the file's link count becomes 0 when it is  
62741 removed and no process has the file open, the space occupied by the file shall be freed and the  
62742 file shall no longer be accessible. If one or more processes have the file open when the last link is  
62743 removed, the link shall be removed before *rename()* returns, but the removal of the file contents  
62744 shall be postponed until all references to the file are closed.62745 Upon successful completion, *rename()* shall mark for update the last data modification and last  
62746 file status change timestamps of the parent directory of each file.62747 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be  
62748 unaffected.

62749 The *renameat()* function shall be equivalent to the *rename()* function except in the case where  
 62750 either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located  
 62751 relative to the directory associated with the file descriptor *oldfd* instead of the current working  
 62752 directory. If *new* is a relative path, the same happens only relative to the directory associated  
 62753 with *newfd*. If the access mode of the open file description associated with the file descriptor is  
 62754 not `O_SEARCH`, the function shall check whether directory searches are permitted using the  
 62755 current permissions of the directory underlying the file descriptor. If the access mode is  
 62756 `O_SEARCH`, the function shall not perform the check.

62757 If *renameat()* is passed the special value `AT_FDCWD` in the *oldfd* or *newfd* parameter, the current  
 62758 working directory shall be used in the determination of the file for the respective *path* parameter.

#### 62759 RETURN VALUE

62760 CX Upon successful completion, the *rename()* function shall return 0. Otherwise, it shall return `-1`,  
 62761 *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by  
 62762 *new* shall be changed or created.

62763 CX Upon successful completion, the *renameat()* function shall return 0. Otherwise, it shall return `-1`  
 62764 and set *errno* to indicate the error.

#### 62765 ERRORS

62766 CX The *rename()* and *renameat()* functions shall fail if:

62767 CX `[EACCES]` A component of either path prefix denies search permission; or one of the  
 62768 directories containing *old* or *new* denies write permissions; or, write  
 62769 permission is required and is denied for a directory pointed to by the *old* or  
 62770 *new* arguments.

62771 CX `[EBUSY]` The directory named by *old* or *new* is currently in use by the system or another  
 62772 process, and the implementation considers this an error.

62773 CX `[EEXIST]` or `[ENOTEMPTY]`

62774 The *new* argument names a directory that is not empty.

62775 CX `[EILSEQ]` The last pathname component of the new pathname is not a portable filename,  
 62776 and cannot be created in the target directory.

62777 CX `[EINVAL]` The *old* pathname names an ancestor directory of the *new* pathname, or either  
 62778 pathname argument contains a final component that is dot or dot-dot.

62779 CX `[EIO]` A physical I/O error has occurred.

62780 CX `[EISDIR]` The *new* argument points to a directory and the *old* argument points to a file  
 62781 that is not a directory.

62782 CX `[ELOOP]` A loop exists in symbolic links encountered during resolution of the *old* or *new*  
 62783 argument.

62784 CX `[EMLINK]` The file named by *old* is a directory, and the link count of the parent directory  
 62785 of *new* would exceed `{LINK_MAX}`.

62786 CX `[ENAMETOOLONG]`

62787 The length of a component of a pathname is longer than `{NAME_MAX}`.

62788 CX `[ENOENT]` The *old* argument does not name an existing file, a component of the path  
 62789 prefix of *new* does not exist, or either *old* or *new* points to an empty string.

62790	CX	[ENOSPC]	The directory that would contain <i>new</i> cannot be extended.
62791	CX	[ENOTDIR]	A component of either path prefix names an existing file that is neither a directory nor a symbolic link to a directory; or the <i>old</i> argument names a directory and the <i>new</i> argument names a non-directory file; or the <i>old</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or the <i>old</i> argument names an existing non-directory file and the <i>new</i> argument names a nonexistent file, contains at least one non- <code>&lt;slash&gt;</code> character, and ends with one or more trailing <code>&lt;slash&gt;</code> characters; or the <i>new</i> argument names an existing non-directory file, contains at least one non- <code>&lt;slash&gt;</code> character, and ends with one or more trailing <code>&lt;slash&gt;</code> characters.
62802	XSI	[EPERM] or [EACCES]	The <code>S_ISVTX</code> flag is set on the directory containing the file referred to by <i>old</i> and the process does not satisfy the criteria specified in XBD Section 4.5 (on page 96) with respect to <i>old</i> ; or <i>new</i> refers to an existing file, the <code>S_ISVTX</code> flag is set on the directory containing this file, and the process does not satisfy the criteria specified in XBD Section 4.5 with respect to this file.
62808	CX	[EROFS]	The requested operation requires writing in a directory on a read-only file system.
62810	CX	[EXDEV]	The file named by <i>old</i> and the directory in which the directory entry named by <i>new</i> is to be created or replaced are on different file systems and the implementation does not support hard links between file systems.
62813	CX	In addition, the <i>renameat()</i> function shall fail if:	
62814		[EACCES]	The access mode of the open file description associated with <i>oldfd</i> or <i>newfd</i> is not <code>O_SEARCH</code> and the permissions of the directory underlying <i>oldfd</i> or <i>newfd</i> , respectively, do not permit directory searches.
62817		[EBADF]	The <i>old</i> argument does not specify an absolute path and the <i>oldfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching, or the <i>new</i> argument does not specify an absolute path and the <i>newfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
62822		[ENOTDIR]	The <i>old</i> or <i>new</i> argument is not an absolute path and <i>oldfd</i> or <i>newfd</i> , respectively, is a file descriptor associated with a non-directory file.
62824	CX	The <i>rename()</i> and <i>renameat()</i> functions may fail if:	
62825	CX	[ELOOP]	More than <code>{SYMLOOP_MAX}</code> symbolic links were encountered during resolution of the <i>old</i> or <i>new</i> argument.
62827	CX	[ENAMETOOLONG]	The length of a pathname exceeds <code>{PATH_MAX}</code> , or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds <code>{PATH_MAX}</code> .
62831	CX	[ETXTBSY]	The file named by <i>new</i> exists and is the last directory entry to a pure procedure (shared text) file that is being executed.

62833 **EXAMPLES**62834 **Renaming a File**

62835 The following example shows how to rename a file named `/home/cnd/mod1` to  
62836 `/home/cnd/mod2`.

```
62837 #include <stdio.h>
62838 int status;
62839 ...
62840 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

62841 **APPLICATION USAGE**

62842 Some implementations mark for update the last file status change timestamp of renamed files  
62843 and some do not. Applications which make use of the last file status change timestamp may  
62844 behave differently with respect to renamed files unless they are designed to allow for either  
62845 behavior.

62846 **RATIONALE**

62847 This `rename()` function is equivalent for regular files to that defined by the ISO C standard. Its  
62848 inclusion here expands that definition to include actions on directories and specifies behavior  
62849 when the *new* parameter names a file that already exists. That specification requires that the  
62850 action of the function be atomic.

62851 One of the reasons for introducing this function was to have a means of renaming directories  
62852 while permitting implementations to prohibit the use of `link()` and `unlink()` with directories,  
62853 thus constraining links to directories to those made by `mkdir()`.

62854 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
62855 rename("x", "x");
62856 does not remove the file.
```

62857 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

62858 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in `rmdir()` and [EBUSY] in  
62859 `unlink()`. For a discussion of [EXDEV], see `link()`.

62860 The purpose of the `renameat()` function is to rename files in directories other than the current  
62861 working directory without exposure to race conditions. Any part of the path of a file could be  
62862 changed in parallel to a call to `rename()`, resulting in unspecified behavior. By opening file  
62863 descriptors for the source and target directories and using the `renameat()` function it can be  
62864 guaranteed that that renamed file is located correctly and the resulting file is in the desired  
62865 directory.

62866 Implementations are encouraged to have `rename()` and `renameat()` report an [EILSEQ] error if the  
62867 file named by *new* does not already exist and the last component of that pathname contains any  
62868 bytes that have the encoded value of a <newline> character.

62869 **FUTURE DIRECTIONS**

62870 None.

62871 **SEE ALSO**

62872 [link\(\)](#), [rmdir\(\)](#), [symlink\(\)](#), [unlink\(\)](#)

62873 XBD Section 4.5 (on page 96), [<fcntl.h>](#), [<stdio.h>](#)

62874 **CHANGE HISTORY**

62875 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

62876 **Issue 5**

62877 The [EBUSY] error is added to the optional part of the ERRORS section.

62878 **Issue 6**

62879 Extensions beyond the ISO C standard are marked.

62880 The following new requirements on POSIX implementations derive from alignment with the  
62881 Single UNIX Specification:

- 62882 • The [EIO] mandatory error condition is added.
- 62883 • The [ELOOP] mandatory error condition is added.
- 62884 • A second [ENAMETOOLONG] is added as an optional error condition.
- 62885 • The [ETXTBSY] optional error condition is added.

62886 The following changes were made to align with the IEEE P1003.1a draft standard:

- 62887 • Details are added regarding the treatment of symbolic links.
- 62888 • The [ELOOP] optional error condition is added.

62889 The normative text is updated to avoid use of the term “must” for application requirements.

62890 **Issue 7**

62891 Austin Group Interpretation 1003.1-2001 #016 is applied, changing the definition of the  
62892 [ENOTDIR] error.

62893 Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final  
62894 component of a path is either dot or dot-dot, and adding the associated [EINVAL] error case.

62895 Austin Group Interpretation 1003.1-2001 #143 is applied.

62896 Austin Group Interpretation 1003.1-2001 #145 is applied, clarifying that the [ENOENT] error  
62897 condition also applies to the case in which a component of *new* does not exist.

62898 Austin Group Interpretations 1003.1-2001 #174 and #181 are applied.

62899 The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API  
62900 Set Part 2.

62901 Changes are made related to support for finegrained timestamps.

62902 Changes are made to allow a directory to be opened for searching.

62903 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0508 [324], XSH/TC1-2008/0509 [147],  
62904 XSH/TC1-2008/0510 [379], XSH/TC1-2008/0511 [278], and XSH/TC1-2008/0512 [278] are  
62905 applied.

62906 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0311 [873], XSH/TC2-2008/0312 [591],  
62907 XSH/TC2-2008/0313 [716], XSH/TC2-2008/0314 [817], XSH/TC2-2008/0315 [817], and  
62908 XSH/TC2-2008/0316 [591] are applied.

62909 **Issue 8**

62910 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
62911 filenames containing any bytes that have the encoded value of a <newline> character.

62912 Austin Group Defect 293 is applied, adding the [EILSEQ] error.

62913 Austin Group Defect 1200 is applied, correcting the argument names in the [ELOOP] errors.

62914

Austin Group Defect 1330 is applied, removing obsolescent interfaces.

62915

Austin Group Defect 1380 is applied, changing text using the term “link” in line with its updated definition.

62916



62917 **NAME**

62918           rewind — reset the file position indicator in a stream

62919 **SYNOPSIS**

62920           #include &lt;stdio.h&gt;

62921           void rewind(FILE \*stream);

62922 **DESCRIPTION**62923 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
62924 conflict between the requirements described here and the ISO C standard is unintentional. This  
62925 volume of POSIX.1-2024 defers to the ISO C standard.

62926       The call:

62927       rewind(stream)

62928       shall be equivalent to:

62929       (void) fseek(stream, 0L, SEEK\_SET)

62930       except that *rewind()* shall also ensure the error indicator is clear when the function returns.62931 CX       Since *rewind()* does not return a value, an application wishing to detect errors should clear *errno*,  
62932 then call *rewind()*, and if *errno* is non-zero, assume an error has occurred.62933 **RETURN VALUE**62934       The *rewind()* function shall not return a value.62935 **ERRORS**62936 CX       Refer to *fseek()* with the exception of [EINVAL] which does not apply.62937 **EXAMPLES**

62938       None.

62939 **APPLICATION USAGE**

62940       None.

62941 **RATIONALE**

62942       None.

62943 **FUTURE DIRECTIONS**

62944       None.

62945 **SEE ALSO**62946       Section 2.5 (on page 521), *fseek()*

62947       XBD &lt;stdio.h&gt;

62948 **CHANGE HISTORY**

62949       First released in Issue 1. Derived from Issue 1 of the SVID.

62950 **Issue 6**

62951       Extensions beyond the ISO C standard are marked.

62952 **Issue 7**

62953       POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0513 [14] is applied.

62954 **Issue 8**62955       Austin Group Defect 1414 is applied, clarifying that *rewind()* ensures the error indicator is clear  
62956 when the function returns.

62957 **NAME**

62958           rewinddir — reset the position of a directory stream to the beginning of a directory

62959 **SYNOPSIS**

62960           #include <dirent.h>

62961           void rewinddir(DIR \*dirp);

62962 **DESCRIPTION**

62963           The *rewinddir()* function shall reset the position of the directory stream to which *dirp* refers to the  
62964 beginning of the directory. It shall also cause the directory stream to refer to the current state of  
62965 the corresponding directory, as a call to *opendir()* would have done. If *dirp* does not refer to a  
62966 directory stream, the effect is undefined.

62967           After a call to the *fork()* function, either the parent or child (but not both) may continue  
62968 XSI       processing the directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and  
62969 child processes use these functions, the result is undefined.

62970 **RETURN VALUE**

62971           The *rewinddir()* function shall not return a value.

62972 **ERRORS**

62973           No errors are defined.

62974 **EXAMPLES**

62975           None.

62976 **APPLICATION USAGE**

62977           The *rewinddir()* function should be used in conjunction with *opendir()*, *readdir()*, and *closedir()* to  
62978 examine the contents of the directory. This method is recommended for portability.

62979 **RATIONALE**

62980           None.

62981 **FUTURE DIRECTIONS**

62982           None.

62983 **SEE ALSO**

62984           *closedir()*, *fdopendir()*, *readdir()*

62985           XBD <dirent.h>, <sys/types.h>

62986 **CHANGE HISTORY**

62987           First released in Issue 2.

62988 **Issue 6**

62989           In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

62990           The following new requirements on POSIX implementations derive from alignment with the  
62991 Single UNIX Specification:

- 62992           • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was  
62993 required for conforming implementations of previous POSIX specifications, it was not  
62994 required for UNIX applications.

62995 **NAME**

62996 rint, rintf, rintl — round-to-nearest integral value

62997 **SYNOPSIS**

```
62998 #include <math.h>
62999 double rint(double x);
63000 float rintf(float x);
63001 long double rintl(long double x);
```

63002 **DESCRIPTION**

63003 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 63004 conflict between the requirements described here and the ISO C standard is unintentional. This  
 63005 volume of POSIX.1-2024 defers to the ISO C standard.

63006 These functions shall return the integral value (represented as a **double**) nearest  $x$  in the  
 63007 direction of the current rounding mode. The current rounding mode is implementation-defined.

63008 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to  
 63009 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be  
 63010 equivalent to *ceil()*. If the current rounding mode rounds towards zero, then *rint()* shall be  
 63011 MX equivalent to *trunc()*. If the current rounding mode rounds towards nearest, then *rint()* differs  
 63012 from *round()* in that halfway cases are rounded to even rather than away from zero.

63013 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that  
 63014 they may raise the inexact floating-point exception if the result differs in value from the  
 63015 argument.

63016 An application wishing to check for error situations should set *errno* to zero and call  
 63017 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 63018 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 63019 zero, an error has occurred.

63020 **RETURN VALUE**

63021 Upon successful completion, these functions shall return the integer (represented as a double  
 63022 MX precision number) nearest  $x$  in the direction of the current rounding mode. The result shall have  
 63023 the same sign as  $x$ .

63024 MX If  $x$  is NaN, a NaN shall be returned.

63025 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.

63026 **ERRORS**

63027 No errors are defined.

63028 **EXAMPLES**

63029 None.

63030 **APPLICATION USAGE**

63031 The integral value returned by these functions need not be expressible as an **intmax\_t**. The  
 63032 return value should be tested before assigning it to an integer type to avoid the undefined  
 63033 results of an integer overflow.

63034 **RATIONALE**

63035 None.

63036 **FUTURE DIRECTIONS**

63037 None.

63038 **SEE ALSO**63039 *abs()*, *ceil()*, *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, *nearbyint()*

63040 XBD Section 4.23 (on page 109), &lt;math.h&gt;

63041 **CHANGE HISTORY**

63042 First released in Issue 4, Version 2.

63043 **Issue 5**

63044 Moved from X/OPEN UNIX extension to BASE.

63045 **Issue 6**

63046 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 63047 • The *rintf()* and *rintl()* functions are added.
- 63048 • The *rint()* function is no longer marked as an extension.
- 63049 • The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
63050 revised to align with the ISO/IEC 9899:1999 standard.
- 63051 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard  
63052 are marked.

63053 **Issue 7**

63054 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0514 [346], XSH/TC1-2008/0515 [346],  
63055 XSH/TC1-2008/0516 [346], XSH/TC1-2008/0517 [346], and XSH/TC1-2008/0518 [346] are  
63056 applied.

63057 **NAME**

63058           rmdir — remove a directory

63059 **SYNOPSIS**

63060           #include &lt;unistd.h&gt;

63061           int rmdir(const char \*path);

63062 **DESCRIPTION**63063           The *rmdir()* function shall remove a directory whose name is given by *path*. The directory shall  
63064           be removed only if it is an empty directory.63065           If the directory is the root directory or the current working directory of any process, it is  
63066           unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].63067           If *path* names a symbolic link, then *rmdir()* shall fail and set *errno* to [ENOTDIR].63068           If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir()* shall  
63069           fail.63070           If the directory's link count becomes 0 and no process has the directory open, the space occupied  
63071           by the directory shall be freed and the directory shall no longer be accessible. If one or more  
63072           processes have the directory open when the last link is removed, the dot and dot-dot entries, if  
63073           present, shall be removed before *rmdir()* returns and no new entries may be created in the  
63074           directory, but the directory shall not be removed until all references to the directory are closed.63075           If the directory is not an empty directory, *rmdir()* shall fail and set *errno* to [EEXIST] or  
63076           [ENOTEMPTY].63077           Upon successful completion, *rmdir()* shall mark for update the last data modification and last  
63078           file status change timestamps of the parent directory.63079 **RETURN VALUE**63080           Upon successful completion, the function *rmdir()* shall return 0. Otherwise, -1 shall be returned,  
63081           and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.63082 **ERRORS**63083           The *rmdir()* function shall fail if:63084           [EACCES]           Search permission is denied on a component of the path prefix, or write  
63085           permission is denied on the parent directory of the directory to be removed.63086           [EBUSY]           The directory to be removed is currently in use by the system or some process  
63087           and the implementation considers this to be an error.

63088           [EEXIST] or [ENOTEMPTY]

63089                           The *path* argument names a directory that is not an empty directory, or there  
63090                           are hard links to the directory other than dot or a single entry in dot-dot.63091           [EINVAL]           The *path* argument contains a last component that is dot.

63092           [EIO]           A physical I/O error has occurred.

63093           [ELOOP]           A loop exists in symbolic links encountered during resolution of the *path*  
63094           argument.

63095           [ENAMETOOLONG]

63096                           The length of a component of a pathname is longer than {NAME\_MAX}.

63097           [ENOENT]           A component of *path* does not name an existing file, or the *path* argument  
63098           names a nonexistent directory or points to an empty string.

- 63099 [ENOTDIR] A component of *path* names an existing file that is neither a directory nor a  
63100 symbolic link to a directory.
- 63101 XSI [EPERM] or [EACCES]  
63102 The S\_ISVTX flag is set on the directory containing the file referred to by the  
63103 *path* argument and the process does not satisfy the criteria specified in XBD  
63104 [Section 4.5](#) (on page 96).
- 63105 [EROFS] The directory entry to be removed resides on a read-only file system.
- 63106 The *rmdir()* function may fail if:
- 63107 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
63108 resolution of the *path* argument.
- 63109 [ENAMETOOLONG]  
63110 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
63111 symbolic link produced an intermediate result with a length that exceeds  
63112 {PATH\_MAX}.

### 63113 EXAMPLES

#### 63114 Removing a Directory

63115 The following example shows how to remove a directory named `/home/cnd/mod1`.

```
63116 #include <unistd.h>
63117 int status;
63118 ...
63119 status = rmdir("/home/cnd/mod1");
```

### 63120 APPLICATION USAGE

63121 None.

### 63122 RATIONALE

63123 The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the  
63124 condition when the directory to be removed does not exist or *new* already exists. When the 1984  
63125 /usr/group standard was published, it contained [EEXIST] instead. When these functions were  
63126 adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore,  
63127 several existing applications and implementations support/use both forms, and no agreement  
63128 could be reached on either value. All implementations are required to supply both [EEXIST] and  
63129 [ENOTEMPTY] in `<errno.h>` with distinct values, so that applications can use both values in C-  
63130 language **case** statements.

63131 The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the  
63132 parent directory to be removed is not clear, particularly in the presence of multiple links to a  
63133 directory.

63134 The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir()* when there are  
63135 multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or  
63136 [ENOTEMPTY] clarifies the behavior in this case.

63137 If the current working directory of the process is being removed, that should be an allowed  
63138 error.

63139 Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in  
63140 [Section 2.3](#) (on page 507) about returning any one of the possible errors permits that behavior to  
63141 continue. The [ELOOP] error may be returned if more than {SYMLOOP\_MAX} symbolic links

63142 are encountered during resolution of the *path* argument.

63143 **FUTURE DIRECTIONS**

63144 None.

63145 **SEE ALSO**

63146 [Section 2.3](#) (on page 507), *mkdir()*, *remove()*, *rename()*, *unlink()*

63147 [XBD Section 4.5](#) (on page 96), [<unistd.h>](#)

63148 **CHANGE HISTORY**

63149 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63150 **Issue 6**

63151 The following new requirements on POSIX implementations derive from alignment with the  
63152 Single UNIX Specification:

- 63153 • The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- 63154 • The [EIO] mandatory error condition is added.
- 63155 • The [ELOOP] mandatory error condition is added.
- 63156 • A second [ENAMETOOLONG] is added as an optional error condition.

63157 The following changes were made to align with the IEEE P1003.1a draft standard:

- 63158 • The [ELOOP] optional error condition is added.

63159 **Issue 7**

63160 Austin Group Interpretation 1003.1-2001 #143 is applied.

63161 Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for  
63162 operations when the S\_ISVTX bit is set.

63163 Changes are made related to support for finegrained timestamps.

63164 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0519 [324] is applied.

63165 **NAME**

63166 round, roundf, roundl — round to the nearest integer value in a floating-point format

63167 **SYNOPSIS**

63168 #include &lt;math.h&gt;

63169 double round(double x);

63170 float roundf(float x);

63171 long double roundl(long double x);

63172 **DESCRIPTION**63173 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
63174 conflict between the requirements described here and the ISO C standard is unintentional. This  
63175 volume of POSIX.1-2024 defers to the ISO C standard.63176 These functions shall round their argument to the nearest integer value in floating-point format,  
63177 rounding halfway cases away from zero, regardless of the current rounding direction.

63178 MX These functions may raise the inexact floating-point exception for finite non-integer arguments.

63179 **RETURN VALUE**63180 MX Upon successful completion, these functions shall return the rounded integer value. The result  
63181 shall have the same sign as  $x$ .63182 MX If  $x$  is NaN, a NaN shall be returned.63183 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.63184 **ERRORS**

63185 No errors are defined.

63186 **EXAMPLES**

63187 None.

63188 **APPLICATION USAGE**63189 The integral value returned by these functions need not be expressible as an `intmax_t`. The  
63190 return value should be tested before assigning it to an integer type to avoid the undefined  
63191 results of an integer overflow.63192 **RATIONALE**

63193 None.

63194 **FUTURE DIRECTIONS**

63195 None.

63196 **SEE ALSO**63197 [feclearexcept\(\)](#), [fetestexcept\(\)](#)63198 XBD [Section 4.23](#) (on page 109), [<math.h>](#)63199 **CHANGE HISTORY**

63200 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

63201 **Issue 7**

63202 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0520 [346] is applied.

63203 **Issue 8**63204 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
63205 standard.



63206 **NAME**

63207 scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT\_RADIX

63208 **SYNOPSIS**

```
63209 #include <math.h>
63210 double scalbn(double x, long n);
63211 float scalblnf(float x, long n);
63212 long double scalblnl(long double x, long n);
63213 double scalbn(double x, int n);
63214 float scalbnf(float x, int n);
63215 long double scalbnl(long double x, int n);
```

63216 **DESCRIPTION**

63217 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 63218 conflict between the requirements described here and the ISO C standard is unintentional. This  
 63219 volume of POSIX.1-2024 defers to the ISO C standard.

63220 These functions shall compute  $x * FLT\_RADIX^n$  efficiently, not normally by computing  
 63221  $FLT\_RADIX^n$  explicitly.

63222 An application wishing to check for error situations should set *errno* to zero and call  
 63223 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 63224 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 63225 zero, an error has occurred.

63226 **RETURN VALUE**63227 Upon successful completion, these functions shall return  $x * FLT\_RADIX^n$ .

63228 MX If the calculation does not overflow or underflow, the returned value shall be exact and shall be  
 63229 independent of the current rounding direction mode.

63230 If the result would cause overflow, a range error shall occur and these functions shall return  
 63231  $\pm HUGE\_VAL$ ,  $\pm HUGE\_VALF$ , and  $\pm HUGE\_VALL$  (according to the sign of  $x$ ) as appropriate for  
 63232 the return type of the function.

63233 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
 63234 MXX and *scalbn*(), *scalblnf*(), *scalblnl*(), *scalbn*(), *scalbnf*(), and *scalbnl*() shall return 0.0, or (if IEC  
 63235 60559 Floating-Point is not supported) an implementation-defined value no greater in  
 63236 magnitude than DBL\_MIN, FLT\_MIN, LDBL\_MIN, DBL\_MIN, FLT\_MIN, and LDBL\_MIN,  
 63237 respectively.

63238 MX If  $x$  is NaN, a NaN shall be returned.63239 If  $x$  is  $\pm 0$  or  $\pm Inf$ ,  $x$  shall be returned.63240 If  $n$  is 0,  $x$  shall be returned.

63241 MXX If the correct value would cause underflow, and is representable, a range error may occur and  
 63242 the correct value shall be returned.

63243 **ERRORS**

63244 These functions shall fail if:

63245 Range Error The result overflows.

63246 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 63247 then *errno* shall be set to [ERANGE]. If the integer expression  
 63248 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 63249 floating-point exception shall be raised.

63250 These functions may fail if:

63251 Range Error The result underflows.

63252 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
63253 then *errno* shall be set to [ERANGE]. If the integer expression  
63254 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
63255 floating-point exception shall be raised.

#### 63256 EXAMPLES

63257 None.

#### 63258 APPLICATION USAGE

63259 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
63260 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

#### 63261 RATIONALE

63262 These functions are named so as to avoid conflicting with the historical definition of the *scalb()*  
63263 function from the Single UNIX Specification. The difference is that the *scalb()* function has a  
63264 second argument of **double** instead of **int**. The *scalb()* function is not part of the ISO C standard.  
63265 The three functions whose second type is **long** are provided because the factor required to scale  
63266 from the smallest positive floating-point value to the largest finite one, on many  
63267 implementations, is too large to represent in the minimum-width **int** format.

#### 63268 FUTURE DIRECTIONS

63269 None.

#### 63270 SEE ALSO

63271 [\*feclearexcept\(\)\*](#), [\*fetestexcept\(\)\*](#)

63272 XBD Section 4.23 (on page 109), [<math.h>](#)

#### 63273 CHANGE HISTORY

63274 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

#### 63275 Issue 7

63276 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0521 [68] and XSH/TC1-2008/0522  
63277 [68] are applied.

#### 63278 Issue 8

63279 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
63280 standard.

63281 **NAME**

63282       scandir — scan a directory

63283 **SYNOPSIS**

63284       #include &lt;dirent.h&gt;

```
63285       int scandir(const char *dir, struct dirent ***namelist,  
63286                   int (*sel)(const struct dirent *),  
63287                   int (*compar)(const struct dirent **, const struct dirent **));
```

63288 **DESCRIPTION**63289       Refer to *alphasort()*.

63290 **NAME**

63291           scanf — convert formatted input

63292 **SYNOPSIS**

63293           #include <stdio.h>

63294           int scanf(const char \*restrict *format*, ...);

63295 **DESCRIPTION**

63296           Refer to *fscanf()*.

**63297 NAME**

63298 sched\_get\_priority\_max, sched\_get\_priority\_min — get priority limits (**REALTIME**)

**63299 SYNOPSIS**

```
63300 PS|TPS #include <sched.h>
63301 int sched_get_priority_max(int policy);
63302 int sched_get_priority_min(int policy);
```

**63303 DESCRIPTION**

63304 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the appropriate  
63305 maximum or minimum, respectively, for the scheduling policy specified by *policy*.

63306 The value of *policy* shall be one of the scheduling policy values defined in **<sched.h>**.

**63307 RETURN VALUE**

63308 If successful, the *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall return the  
63309 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a  
63310 value of  $-1$  and set *errno* to indicate the error.

**63311 ERRORS**

63312 The *sched\_get\_priority\_max()* and *sched\_get\_priority\_min()* functions shall fail if:

63313 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling  
63314 policy.

**63315 EXAMPLES**

63316 None.

**63317 APPLICATION USAGE**

63318 None.

**63319 RATIONALE**

63320 None.

**63321 FUTURE DIRECTIONS**

63322 None.

**63323 SEE ALSO**

63324 [sched\\_getparam\(\)](#), [sched\\_setparam\(\)](#), [sched\\_getscheduler\(\)](#), [sched\\_rr\\_get\\_interval\(\)](#),  
63325 [sched\\_setscheduler\(\)](#)

63326 XBD **<sched.h>**

**63327 CHANGE HISTORY**

63328 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**63329 Issue 6**

63330 These functions are marked as part of the Process Scheduling option.

63331 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
63332 implementation does not support the Process Scheduling option.

63333 The [ESRCH] error condition has been removed since these functions do not take a *pid*  
63334 argument.

63335 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin  
63336 code in the SYNOPSIS to PS|TPS.

63337 **NAME**63338 sched\_getparam — get scheduling parameters (**REALTIME**)63339 **SYNOPSIS**

```
63340 PS #include <sched.h>
63341 int sched_getparam(pid_t pid, struct sched_param *param);
```

63342 **DESCRIPTION**

63343 The *sched\_getparam()* function shall return the scheduling parameters of a process specified by  
63344 *pid* in the **sched\_param** structure pointed to by *param*.

63345 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
63346 parameters for the process whose process ID is equal to *pid* shall be returned.

63347 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of  
63348 the *sched\_getparam()* function is unspecified if the value of *pid* is negative.

63349 **RETURN VALUE**

63350 Upon successful completion, the *sched\_getparam()* function shall return zero. If the call to  
63351 *sched\_getparam()* is unsuccessful, the function shall return a value of  $-1$  and set *errno* to indicate  
63352 the error.

63353 **ERRORS**

63354 The *sched\_getparam()* function shall fail if:

63355 [EPERM] The requesting process does not have permission to obtain the scheduling  
63356 parameters of the specified process.

63357 [ESRCH] No process can be found corresponding to that specified by *pid*.

63358 **EXAMPLES**

63359 None.

63360 **APPLICATION USAGE**

63361 None.

63362 **RATIONALE**

63363 None.

63364 **FUTURE DIRECTIONS**

63365 None.

63366 **SEE ALSO**

63367 [sched\\_getscheduler\(\)](#), [sched\\_setparam\(\)](#), [sched\\_setscheduler\(\)](#)

63368 XBD [<sched.h>](#)

63369 **CHANGE HISTORY**

63370 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63371 **Issue 6**

63372 The *sched\_getparam()* function is marked as part of the Process Scheduling option.

63373 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
63374 implementation does not support the Process Scheduling option.

63375 **NAME**63376 sched\_getscheduler — get scheduling policy (**REALTIME**)63377 **SYNOPSIS**

```
63378 PS #include <sched.h>
63379 int sched_getscheduler(pid_t pid);
```

63380 **DESCRIPTION**

63381 The *sched\_getscheduler()* function shall return the scheduling policy of the process specified by  
 63382 *pid*. If the value of *pid* is negative, the behavior of the *sched\_getscheduler()* function is  
 63383 unspecified.

63384 The values that can be returned by *sched\_getscheduler()* are defined in the **<sched.h>** header.

63385 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 63386 policy shall be returned for the process whose process ID is equal to *pid*.

63387 If *pid* is zero, the scheduling policy shall be returned for the calling process.

63388 **RETURN VALUE**

63389 Upon successful completion, the *sched\_getscheduler()* function shall return the scheduling policy  
 63390 of the specified process. If unsuccessful, the function shall return  $-1$  and set *errno* to indicate the  
 63391 error.

63392 **ERRORS**

63393 The *sched\_getscheduler()* function shall fail if:

63394 [EPERM] The requesting process does not have permission to determine the scheduling  
 63395 policy of the specified process.

63396 [ESRCH] No process can be found corresponding to that specified by *pid*.

63397 **EXAMPLES**

63398 None.

63399 **APPLICATION USAGE**

63400 None.

63401 **RATIONALE**

63402 None.

63403 **FUTURE DIRECTIONS**

63404 None.

63405 **SEE ALSO**

63406 [sched\\_getparam\(\)](#), [sched\\_setparam\(\)](#), [sched\\_setscheduler\(\)](#)

63407 XBD **<sched.h>**

63408 **CHANGE HISTORY**

63409 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63410 **Issue 6**

63411 The *sched\_getscheduler()* function is marked as part of the Process Scheduling option.

63412 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 63413 implementation does not support the Process Scheduling option.

63414 **NAME**63415 sched\_rr\_get\_interval — get execution time limits (**REALTIME**)63416 **SYNOPSIS**

```
63417 PS|TPS #include <sched.h>
63418 int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

63419 **DESCRIPTION**

63420 The *sched\_rr\_get\_interval()* function shall update the **timespec** structure referenced by the  
63421 *interval* argument to contain the current execution time limit (that is, time quantum) for the  
63422 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process  
63423 shall be returned.

63424 **RETURN VALUE**

63425 If successful, the *sched\_rr\_get\_interval()* function shall return zero. Otherwise, it shall return a  
63426 value of -1 and set *errno* to indicate the error.

63427 **ERRORS**

63428 The *sched\_rr\_get\_interval()* function shall fail if:

63429 [ESRCH] No process can be found corresponding to that specified by *pid*.

63430 **EXAMPLES**

63431 None.

63432 **APPLICATION USAGE**

63433 None.

63434 **RATIONALE**

63435 None.

63436 **FUTURE DIRECTIONS**

63437 None.

63438 **SEE ALSO**

63439 [sched\\_getparam\(\)](#), [sched\\_get\\_priority\\_max\(\)](#), [sched\\_getscheduler\(\)](#), [sched\\_setparam\(\)](#),  
63440 [sched\\_setscheduler\(\)](#)

63441 XBD [<sched.h>](#)

63442 **CHANGE HISTORY**

63443 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63444 **Issue 6**

63445 The *sched\_rr\_get\_interval()* function is marked as part of the Process Scheduling option.

63446 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
63447 implementation does not support the Process Scheduling option.

63448 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in  
63449 the SYNOPSIS to PS|TPS.



63450 **NAME**63451 sched\_setparam — set scheduling parameters (**REALTIME**)63452 **SYNOPSIS**

```
63453 PS #include <sched.h>
63454 int sched_setparam(pid_t pid, const struct sched_param *param);
```

63455 **DESCRIPTION**

63456 The *sched\_setparam()* function shall set the scheduling parameters of the process specified by *pid*  
 63457 to the values specified by the **sched\_param** structure pointed to by *param*. The value of the  
 63458 *sched\_priority* member in the **sched\_param** structure shall be any integer within the inclusive  
 63459 priority range for the current scheduling policy of the process specified by *pid*. Higher  
 63460 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the  
 63461 behavior of the *sched\_setparam()* function is unspecified.

63462 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 63463 parameters shall be set for the process whose process ID is equal to *pid*.

63464 If *pid* is zero, the scheduling parameters shall be set for the calling process.

63465 The conditions under which one process has permission to change the scheduling parameters of  
 63466 another process are implementation-defined.

63467 Implementations may require the requesting process to have appropriate privileges to set its  
 63468 own scheduling parameters or those of another process.

63469 See [Scheduling Policies](#) (on page 531) for a description on how this function affects the  
 63470 scheduling of the threads within the target process.

63471 SS If the current scheduling policy for the target process is not SCHED\_FIFO, SCHED\_RR, or  
 63472 SCHED\_SPORADIC, the result is implementation-defined; this case includes the  
 63473 SCHED\_OTHER policy.

63474 SS The specified *sched\_ss\_repl\_period* shall be greater than or equal to the specified  
 63475 *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.

63476 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,{SS\_REPL\_MAX}] for the  
 63477 function to succeed; if not, the function shall fail. It is unspecified whether the  
 63478 *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values are stored as provided by this function or are  
 63479 rounded to align with the resolution of the clock being used.

63480 This function is not atomic with respect to other threads in the process. Threads may continue to  
 63481 execute while this function call is in the process of changing the scheduling policy for the  
 63482 underlying kernel-scheduled entities used by the process contention scope threads.

63483 **RETURN VALUE**

63484 If successful, the *sched\_setparam()* function shall return zero.

63485 If the call to *sched\_setparam()* is unsuccessful, the priority shall remain unchanged, and the  
 63486 function shall return a value of  $-1$  and set *errno* to indicate the error.

63487 **ERRORS**

63488 The *sched\_setparam()* function shall fail if:

63489 [EINVAL] One or more of the requested scheduling parameters is outside the range  
 63490 defined for the scheduling policy of the specified *pid*.

63491	[EPERM]	The requesting process does not have permission to set the scheduling parameters for the specified process, or does not have appropriate privileges to invoke <i>sched_setparam()</i> .
63492		
63493		
63494	[ESRCH]	No process can be found corresponding to that specified by <i>pid</i> .
63495	<b>EXAMPLES</b>	
63496		None.
63497	<b>APPLICATION USAGE</b>	
63498		None.
63499	<b>RATIONALE</b>	
63500		None.
63501	<b>FUTURE DIRECTIONS</b>	
63502		None.
63503	<b>SEE ALSO</b>	
63504		<a href="#">Scheduling Policies</a> (on page 531), <i>sched_getparam()</i> , <i>sched_getscheduler()</i> , <i>sched_setscheduler()</i>
63505		XBD <a href="#">&lt;sched.h&gt;</a>
63506	<b>CHANGE HISTORY</b>	
63507		First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
63508	<b>Issue 6</b>	
63509		The <i>sched_setparam()</i> function is marked as part of the Process Scheduling option.
63510		The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Scheduling option.
63511		
63512		The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
63513		
63514		• In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is added.
63515		
63516		• Sections describing two-level scheduling and atomicity of the function are added.
63517		The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.
63518		IEEE PASC Interpretation 1003.1 #100 is applied.
63519	<b>Issue 7</b>	
63520		Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.
63521		Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements for the <i>sched_ss_repl_period</i> and <i>sched_ss_init_budget</i> values.
63522		

63523 **NAME**63524 sched\_setscheduler — set scheduling policy and parameters (**REALTIME**)63525 **SYNOPSIS**

```
63526 PS #include <sched.h>
63527 int sched_setscheduler(pid_t pid, int policy,
63528     const struct sched_param *param);
```

63529 **DESCRIPTION**

63530 The *sched\_setscheduler()* function shall set the scheduling policy and scheduling parameters of  
 63531 the process specified by *pid* to *policy* and the parameters specified in the **sched\_param** structure  
 63532 pointed to by *param*, respectively. The value of the *sched\_priority* member in the **sched\_param**  
 63533 structure shall be any integer within the inclusive priority range for the scheduling policy  
 63534 specified by *policy*. If the value of *pid* is negative, the behavior of the *sched\_setscheduler()*  
 63535 function is unspecified.

63536 The possible values for the *policy* parameter are defined in the **<sched.h>** header.

63537 If a process specified by *pid* exists, and if the calling process has permission, the scheduling  
 63538 policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

63539 If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling  
 63540 process.

63541 The conditions under which one process has appropriate privileges to change the scheduling  
 63542 parameters of another process are implementation-defined.

63543 Implementations may require that the requesting process have permission to set its own  
 63544 scheduling parameters or those of another process. Additionally, implementation-defined  
 63545 restrictions may apply as to the appropriate privileges required to set the scheduling policy of  
 63546 the process, or the scheduling policy of another process, to a particular value.

63547 The *sched\_setscheduler()* function shall be considered successful if it succeeds in setting the  
 63548 scheduling policy and scheduling parameters of the process specified by *pid* to the values  
 63549 specified by *policy* and the structure pointed to by *param*, respectively.

63550 See [Scheduling Policies](#) (on page 531) for a description on how this function affects the  
 63551 scheduling of the threads within the target process.

63552 SS If the current scheduling policy for the target process is not SCHED\_FIFO, SCHED\_RR, or  
 63553 SCHED\_SPORADIC, the result is implementation-defined; this case includes the  
 63554 SCHED\_OTHER policy.

63555 SS The specified *sched\_ss\_repl\_period* shall be greater than or equal to the specified  
 63556 *sched\_ss\_init\_budget* for the function to succeed; if it is not, then the function shall fail.

63557 The value of *sched\_ss\_max\_repl* shall be within the inclusive range [1,SS\_REPL\_MAX] for the  
 63558 function to succeed; if not, the function shall fail. It is unspecified whether the  
 63559 *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values are stored as provided by this function or are  
 63560 rounded to align with the resolution of the clock being used.

63561 This function is not atomic with respect to other threads in the process. Threads may continue to  
 63562 execute while this function call is in the process of changing the scheduling policy and  
 63563 associated scheduling parameters for the underlying kernel-scheduled entities used by the  
 63564 process contention scope threads.

63565 **RETURN VALUE**

63566 Upon successful completion, the function shall return the former scheduling policy of the  
 63567 specified process. If the *sched\_setscheduler()* function fails to complete successfully, the policy  
 63568 and scheduling parameters shall remain unchanged, and the function shall return a value of -1  
 63569 and set *errno* to indicate the error.

63570 **ERRORS**

63571 The *sched\_setscheduler()* function shall fail if:

63572 [EINVAL] The value of the *policy* parameter is invalid, or one or more of the parameters  
 63573 contained in *param* is outside the valid range for the specified scheduling  
 63574 policy.

63575 [EPERM] The requesting process does not have permission to set either or both of the  
 63576 scheduling parameters or the scheduling policy of the specified process.

63577 [ESRCH] No process can be found corresponding to that specified by *pid*.

63578 **EXAMPLES**

63579 None.

63580 **APPLICATION USAGE**

63581 None.

63582 **RATIONALE**

63583 None.

63584 **FUTURE DIRECTIONS**

63585 None.

63586 **SEE ALSO**

63587 [Scheduling Policies](#) (on page 531), *sched\_getparam()*, *sched\_getscheduler()*, *sched\_setparam()*

63588 XBD [<sched.h>](#)

63589 **CHANGE HISTORY**

63590 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

63591 **Issue 6**

63592 The *sched\_setscheduler()* function is marked as part of the Process Scheduling option.

63593 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 63594 implementation does not support the Process Scheduling option.

63595 The following new requirements on POSIX implementations derive from alignment with the  
 63596 Single UNIX Specification:

- 63597 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is  
 63598 added.

- 63599 • Sections describing two-level scheduling and atomicity of the function are added.

63600 The SCHED\_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

63601 **Issue 7**

63602 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

63603 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements  
 63604 for the *sched\_ss\_repl\_period* and *sched\_ss\_init\_budget* values.

**63605 NAME**

63606 sched\_yield — yield the processor

**63607 SYNOPSIS**

63608 #include <sched.h>

63609 int sched\_yield(void);

**63610 DESCRIPTION**

63611 The *sched\_yield()* function shall force the running thread to relinquish the processor until it again  
63612 becomes the head of its thread list. It takes no arguments.

**63613 RETURN VALUE**

63614 The *sched\_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a  
63615 value of -1 and set *errno* to indicate the error.

**63616 ERRORS**

63617 No errors are defined.

**63618 EXAMPLES**

63619 None.

**63620 APPLICATION USAGE**

63621 The conceptual model for scheduling semantics in POSIX.1-2024 defines a set of thread lists. This  
63622 set of thread lists is always present regardless of the scheduling options supported by the  
63623 system. On a system where the Process Scheduling option is not supported, portable  
63624 applications should not make any assumptions regarding whether threads from other processes  
63625 will be on the same thread list.

**63626 RATIONALE**

63627 None.

**63628 FUTURE DIRECTIONS**

63629 None.

**63630 SEE ALSO**

63631 XBD <[sched.h](#)>

**63632 CHANGE HISTORY**

63633 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
63634 POSIX Threads Extension.

**63635 Issue 6**

63636 The *sched\_yield()* function is now marked as part of the Process Scheduling and Threads options.

**63637 Issue 7**

63638 SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

63639 The *sched\_yield()* function is moved to the Base.

63640 **NAME**

63641 secure\_getenv — get value of an environment variable

63642 **SYNOPSIS**

63643 #include <stdlib.h>

63644 CX `char *secure_getenv(const char *name);`

63645 **DESCRIPTION**

63646 Refer to *getenv()*.

63647 **NAME**

63648 seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

63649 **SYNOPSIS**

```
63650 XSI #include <stdlib.h>  
63651 unsigned short *seed48(unsigned short seed16v[3]);
```

63652 **DESCRIPTION**63653 Refer to *drand48()*.

63654 **NAME**

63655 seekdir — set the position of a directory stream

63656 **SYNOPSIS**

```
63657 XSI #include <dirent.h>  
63658 void seekdir(DIR *dirp, long loc);
```

63659 **DESCRIPTION**

63660 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory  
63661 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been  
63662 returned from an earlier call to *telldir()* using the same directory stream. The new position  
63663 reverts to the one associated with the directory stream when *telldir()* was performed.

63664 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*  
63665 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to  
63666 *readdir()* are unspecified.

63667 **RETURN VALUE**63668 The *seekdir()* function shall not return a value.63669 **ERRORS**

63670 No errors are defined.

63671 **EXAMPLES**

63672 None.

63673 **APPLICATION USAGE**

63674 None.

63675 **RATIONALE**

63676 The original standard developers perceived that there were restrictions on the use of the  
63677 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these  
63678 functions need not be supported on all POSIX-conforming systems. They are required on  
63679 implementations supporting the XSI option.

63680 One of the perceived problems of implementation is that returning to a given point in a directory  
63681 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-  
63682 trees, hashing functions, or other similar mechanisms to order their directories are considered.  
63683 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a  
63684 given directory entry will be seen at all, or more than once.

63685 On systems not supporting these functions, their capability can sometimes be accomplished by  
63686 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to  
63687 relocate the position from which the filename was saved.

63688 **FUTURE DIRECTIONS**

63689 None.

63690 **SEE ALSO**63691 *fdopendir()*, *readdir()*, *telldir()*63692 XBD [<dirent.h>](#), [<sys/types.h>](#)63693 **CHANGE HISTORY**

63694 First released in Issue 2.



63695 **Issue 6**

63696 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

63697 **Issue 7**

63698 SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should  
63699 have been returned from an earlier call to *telldir()* using the same directory stream.

63700 **NAME**

63701           select — synchronous I/O multiplexing

63702 **SYNOPSIS**

63703           #include <sys/select.h>

```
63704           int select(int nfds, fd_set *restrict readfds,  
63705                      fd_set *restrict writefds, fd_set *restrict errorfds,  
63706                      struct timeval *restrict timeout);
```

63707 **DESCRIPTION**

63708           Refer to [pselect\(\)](#).

63709 **NAME**

63710 sem\_clockwait, sem\_timedwait — lock a semaphore

63711 **SYNOPSIS**

```
63712 #include <semaphore.h>
63713 int sem_clockwait(sem_t *restrict sem, clockid_t clock_id,
63714                 const struct timespec *restrict abstime);
63715 int sem_timedwait(sem_t *restrict sem,
63716                 const struct timespec *restrict abstime);
```

63717 **DESCRIPTION**

63718 The *sem\_clockwait()* and *sem\_timedwait()* functions shall lock the semaphore referenced by *sem* as  
 63719 in the *sem\_wait()* function. However, if the semaphore cannot be locked without waiting for  
 63720 another process or thread to unlock the semaphore by performing a *sem\_post()* function, this  
 63721 wait shall be terminated when the specified timeout expires.

63722 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the  
 63723 clock on which timeouts are based (that is, when the value of that clock equals or exceeds  
 63724 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the  
 63725 call.

63726 For *sem\_timedwait()*, the timeout shall be based on the CLOCK\_REALTIME clock. For  
 63727 *sem\_clockwait()*, the timeout shall be based on the clock specified by the *clock\_id* argument. The  
 63728 resolution of the timeout shall be the resolution of the clock on which it is based.  
 63729 Implementations shall support passing CLOCK\_REALTIME and CLOCK\_MONOTONIC to  
 63730 *sem\_clockwait()* as the *clock\_id* argument.

63731 Under no circumstance shall the function fail with a timeout if the semaphore can be locked  
 63732 immediately. The validity of the *abstime* need not be checked if the semaphore can be locked  
 63733 immediately.

63734 **RETURN VALUE**

63735 The *sem\_clockwait()* and *sem\_timedwait()* functions shall return zero if the calling process  
 63736 successfully performed the semaphore lock operation on the semaphore designated by *sem*. If  
 63737 the call was unsuccessful, the state of the semaphore shall be unchanged, and the functions shall  
 63738 return a value of -1 and set *errno* to indicate the error.

63739 **ERRORS**

63740 The *sem\_clockwait()* and *sem\_timedwait()* functions shall fail if:

63741 [EINVAL] The process or thread would have blocked, and either the *abstime* parameter  
 63742 specified a nanoseconds field value less than zero or greater than or equal to  
 63743 1000 million, or the *sem\_clockwait()* function was passed an invalid or  
 63744 unsupported *clock\_id* value.

63745 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

63746 The *sem\_clockwait()* and *sem\_timedwait()* functions may fail if:

63747 [EDEADLK] A deadlock condition was detected.

63748 [EINTR] A signal interrupted the function.

63749 [EINVAL] The *sem* argument does not refer to a valid semaphore.

63750 **EXAMPLES**

63751 The program shown below operates on an unnamed semaphore. The program expects two  
 63752 command-line arguments. The first argument specifies a seconds value that is used to set an  
 63753 alarm timer to generate a SIGALRM signal. This handler performs a *sem\_post()* to increment the  
 63754 semaphore that is being waited on in *main()* using *sem\_clockwait()*. The second command-line  
 63755 argument specifies the length of the timeout, in seconds, for *sem\_clockwait()*.

```

63756 #include <unistd.h>
63757 #include <stdio.h>
63758 #include <stdlib.h>
63759 #include <semaphore.h>
63760 #include <time.h>
63761 #include <assert.h>
63762 #include <errno.h>
63763 #include <signal.h>

63764 sem_t sem;

63765 static void
63766 handler(int sig)
63767 {
63768     int sav_errno = errno;
63769     static const char info_msg[] = "sem_post() from handler\n";
63770     write(STDOUT_FILENO, info_msg, sizeof info_msg - 1);
63771     if (sem_post(&sem) == -1) {
63772         static const char err_msg[] = "sem_post() failed\n";
63773         write(STDERR_FILENO, err_msg, sizeof err_msg - 1);
63774         _exit(EXIT_FAILURE);
63775     }
63776     errno = sav_errno;
63777 }

63778 int
63779 main(int argc, char *argv[])
63780 {
63781     struct sigaction sa;
63782     struct timespec ts;
63783     int s;

63784     if (argc != 3) {
63785         fprintf(stderr, "Usage: %s <alarm-secs> <wait-secs>\n",
63786             argv[0]);
63787         exit(EXIT_FAILURE);
63788     }

63789     if (sem_init(&sem, 0, 0) == -1) {
63790         perror("sem_init");
63791         exit(EXIT_FAILURE);
63792     }

63793     /* Establish SIGALRM handler; set alarm timer using argv[1] */

63794     sa.sa_handler = handler;
63795     sigemptyset(&sa.sa_mask);
63796     sa.sa_flags = 0;
63797     if (sigaction(SIGALRM, &sa, NULL) == -1) {

```

```

63798         perror("sigaction");
63799         exit(EXIT_FAILURE);
63800     }
63801     alarm(atoi(argv[1]));
63802     /* Calculate relative interval as current time plus
63803        number of seconds given argv[2] */
63804     if (clock_gettime(CLOCK_MONOTONIC, &ts) == -1) {
63805         perror("clock_gettime");
63806         exit(EXIT_FAILURE);
63807     }
63808     ts.tv_sec += atoi(argv[2]);
63809     printf("main() about to call sem_clockwait()\n");
63810     while ((s = sem_clockwait(&sem, CLOCK_MONOTONIC, &ts)) == -1 &&
63811         errno == EINTR)
63812         continue; /* Restart if interrupted by handler */
63813     /* Check what happened */
63814     if (s == -1) {
63815         if (errno == ETIMEDOUT)
63816             printf("sem_clockwait() timed out\n");
63817         else
63818             perror("sem_clockwait");
63819     } else
63820         printf("sem_clockwait() succeeded\n");
63821     exit((s == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
63822 }

```

### 63823 APPLICATION USAGE

63824 Applications using these functions may be subject to priority inversion, as discussed in XBD  
63825 [Section 3.275](#) (on page 72).

### 63826 RATIONALE

63827 None.

### 63828 FUTURE DIRECTIONS

63829 None.

### 63830 SEE ALSO

63831 [sem\\_post\(\)](#), [sem\\_trywait\(\)](#), [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [time\(\)](#)

63832 XBD [Section 3.275](#) (on page 72), [<semaphore.h>](#), [<time.h>](#)

### 63833 CHANGE HISTORY

63834 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

63835 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/120 is applied, updating the ERRORS  
63836 section so that the [EINVAL] error becomes optional.

### 63837 Issue 7

63838 The [sem\\_timedwait\(\)](#) function is moved from the Semaphores option to the Base.

63839 Functionality relating to the Timers option is moved to the Base.

63840 An example is added.

- 63841 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0529 [138] is applied.
- 63842 **Issue 8**
- 63843 Austin Group Defect 592 is applied, removing text relating to **<time.h>** from the SYNOPSIS and
- 63844 DESCRIPTION sections.
- 63845 Austin Group Defect 1216 is applied, adding *sem\_clockwait()*.

**63846 NAME**

63847 sem\_close — close a named semaphore

**63848 SYNOPSIS**

```
63849 #include <semaphore.h>
63850 int sem_close(sem_t *sem);
```

**63851 DESCRIPTION**

63852 The *sem\_close()* function shall indicate that the calling process is finished using the named  
63853 semaphore indicated by *sem*. The effects of calling *sem\_close()* for an unnamed semaphore (one  
63854 created by *sem\_init()*) are undefined. The *sem\_close()* function shall deallocate (that is, make  
63855 available for reuse by a subsequent *sem\_open()* by this process) any system resources allocated  
63856 by the system for use by this process for this semaphore. If the semaphore indicated by *sem* is  
63857 implemented using a file descriptor, the file descriptor shall be closed. The effect of subsequent  
63858 use of the semaphore indicated by *sem* by this process is undefined. If any threads in the calling  
63859 process are currently blocked on the semaphore, the behavior is undefined. If the semaphore  
63860 has not been removed with a successful call to *sem\_unlink()*, then *sem\_close()* has no effect on the  
63861 state of the semaphore. If the *sem\_unlink()* function has been successfully invoked for *name* after  
63862 the most recent call to *sem\_open()* with O\_CREAT for this semaphore, then when all processes  
63863 that have opened the semaphore close all semaphore handles to it, the semaphore is no longer  
63864 accessible.

**63865 RETURN VALUE**

63866 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
63867 returned and *errno* set to indicate the error.

**63868 ERRORS**

63869 The *sem\_close()* function may fail if:

63870 [EINVAL] The *sem* argument is not a valid semaphore descriptor.

**63871 EXAMPLES**

63872 None.

**63873 APPLICATION USAGE**

63874 None.

**63875 RATIONALE**

63876 None.

**63877 FUTURE DIRECTIONS**

63878 None.

**63879 SEE ALSO**

63880 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem\\_init\(\)](#), [sem\\_open\(\)](#), [sem\\_unlink\(\)](#)

63881 XBD [<semaphore.h>](#)

**63882 CHANGE HISTORY**

63883 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**63884 Issue 6**

63885 The *sem\_close()* function is marked as part of the Semaphores option.

63886 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
63887 implementation does not support the Semaphores option.

63888 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/113 is applied, updating the ERRORS  
63889 section so that the [EINVAL] error becomes optional.

63890 **Issue 7**

63891 The *sem\_close()* function is moved from the Semaphores option to the Base.

63892 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0523 [37] is applied.

63893 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0317 [870] is applied.

63894 **Issue 8**

63895 Austin Group Defect 368 is applied, adding a requirement that if *sem* is implemented using a file  
63896 descriptor, *sem\_close()* closes the file descriptor.

63897 Austin Group Defect 1324 is applied, clarifying the circumstances under which an unlinked  
63898 semaphore is no longer accessible.



**63899 NAME**

63900 sem\_destroy — destroy an unnamed semaphore

**63901 SYNOPSIS**

```
63902 #include <semaphore.h>
63903 int sem_destroy(sem_t *sem);
```

**63904 DESCRIPTION**

63905 The *sem\_destroy()* function shall destroy the unnamed semaphore indicated by *sem*. If an  
63906 unnamed semaphore is implemented using a file descriptor, the file descriptor shall be closed.  
63907 Only a semaphore that was created using *sem\_init()* can be destroyed using *sem\_destroy()*; the  
63908 effect of calling *sem\_destroy()* with a named semaphore is undefined. The effect of subsequent  
63909 use of the semaphore *sem* is undefined until *sem* is reinitialized by another call to *sem\_init()*.

63910 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The  
63911 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

**63912 RETURN VALUE**

63913 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be  
63914 returned and *errno* set to indicate the error.

**63915 ERRORS**

63916 The *sem\_destroy()* function may fail if:

- |       |          |   |
|-------|----------|---|
| 63917 | [EINVAL] | The <i>sem</i> argument is not a valid semaphore.       |
| 63918 | [EBUSY]  | There are currently processes blocked on the semaphore. |

**63919 EXAMPLES**

63920 None.

**63921 APPLICATION USAGE**

63922 None.

**63923 RATIONALE**

63924 None.

**63925 FUTURE DIRECTIONS**

63926 None.

**63927 SEE ALSO**

63928 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem\\_init\(\)](#), [sem\\_open\(\)](#)

63929 XBD [<semaphore.h>](#)

**63930 CHANGE HISTORY**

63931 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**63932 Issue 6**

63933 The *sem\_destroy()* function is marked as part of the Semaphores option.

63934 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
63935 implementation does not support the Semaphores option.

63936 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/114 is applied, updating the ERRORS  
63937 section so that the [EINVAL] error becomes optional.

63938 **Issue 7**

63939 The *sem\_destroy()* function is moved from the Semaphores option to the Base.

63940 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0524 [37] is applied.

63941 **Issue 8**

63942 Austin Group Defect 368 is applied, adding a requirement that if an unnamed semaphore is  
63943 implemented using a file descriptor, *sem\_destroy()* closes the file descriptor.

**63944 NAME**

63945 sem\_getvalue — get the value of a semaphore

**63946 SYNOPSIS**

63947 #include <semaphore.h>

63948 int sem\_getvalue(sem\_t \*restrict sem, int \*restrict sval);

**63949 DESCRIPTION**

63950 The *sem\_getvalue()* function shall update the location referenced by the *sval* argument to have  
63951 the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The  
63952 updated value represents an actual semaphore value that occurred at some unspecified time  
63953 during the call, but it need not be the actual value of the semaphore when it is returned to the  
63954 calling process.

63955 If *sem* is locked, then the object to which *sval* points shall either be set to zero or to a negative  
63956 number whose absolute value represents the number of processes waiting for the semaphore at  
63957 some unspecified time during the call.

**63958 RETURN VALUE**

63959 Upon successful completion, the *sem\_getvalue()* function shall return a value of zero. Otherwise,  
63960 it shall return a value of -1 and set *errno* to indicate the error.

**63961 ERRORS**

63962 The *sem\_getvalue()* function may fail if:

63963 [EINVAL] The *sem* argument does not refer to a valid semaphore.

**63964 EXAMPLES**

63965 None.

**63966 APPLICATION USAGE**

63967 None.

**63968 RATIONALE**

63969 None.

**63970 FUTURE DIRECTIONS**

63971 None.

**63972 SEE ALSO**

63973 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem\\_clockwait\(\)](#), [sem\\_post\(\)](#), [sem\\_trywait\(\)](#)

63974 XBD [<semaphore.h>](#)

**63975 CHANGE HISTORY**

63976 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**63977 Issue 6**

63978 The *sem\_getvalue()* function is marked as part of the Semaphores option.

63979 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
63980 implementation does not support the Semaphores option.

63981 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std  
63982 1003.1d-1999.

63983 The **restrict** keyword is added to the *sem\_getvalue()* prototype for alignment with the  
63984 ISO/IEC 9899:1999 standard.

63985 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied.

63986 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/115 is applied, updating the ERRORS

63987 section so that the [EINVAL] error becomes optional.

63988 **Issue 7**

63989 The *sem\_getvalue()* function is moved from the Semaphores option to the Base.

63990 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0525 [37] is applied.

**63991 NAME**

63992 sem\_init — initialize an unnamed semaphore

**63993 SYNOPSIS**

63994 #include <semaphore.h>

63995 int sem\_init(sem\_t \*sem, int pshared, unsigned value);

**63996 DESCRIPTION**

63997 The *sem\_init()* function shall initialize the unnamed semaphore referred to by *sem*. The value of  
63998 the initialized semaphore shall be *value*. Following a successful call to *sem\_init()*, the semaphore  
63999 can be used in subsequent calls to *sem\_clockwait()*, *sem\_destroy()*, *sem\_post()*, *sem\_timedwait()*,  
64000 *sem\_trywait()*, and *sem\_wait()*. This semaphore shall remain usable until the semaphore is  
64001 destroyed. An unnamed semaphore may be implemented using a file descriptor.

64002 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;  
64003 in this case, any process that can access the semaphore *sem* can use *sem* for performing  
64004 *sem\_clockwait()*, *sem\_destroy()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, and *sem\_wait()*  
64005 operations.

64006 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any  
64007 thread in this process can use *sem* for performing *sem\_clockwait()*, *sem\_destroy()*, *sem\_post()*,  
64008 *sem\_timedwait()*, *sem\_trywait()*, and *sem\_wait()* operations.

64009 See [Section 2.9.9](#) (on page 548) for further requirements.

64010 Attempting to initialize an already initialized semaphore results in undefined behavior.

**64011 RETURN VALUE**

64012 Upon successful completion, the *sem\_init()* function shall initialize the semaphore in *sem* and  
64013 return 0. Otherwise, it shall return -1 and set *errno* to indicate the error.

**64014 ERRORS**

64015 The *sem\_init()* function shall fail if:

64016 [EINVAL] The *value* argument exceeds {SEM\_VALUE\_MAX}.

64017 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the  
64018 limit on semaphores ({SEM\_NSEMS\_MAX}) has been reached.

64019 [EPERM] The process lacks appropriate privileges to initialize the semaphore.

64020 The *sem\_init()* function may fail if:

64021 [EMFILE] All file descriptors available to the process are currently open.

64022 [ENFILE] The maximum allowable number of files is currently open in the system.

**64023 EXAMPLES**

64024 None.

**64025 APPLICATION USAGE**

64026 None.

**64027 RATIONALE**

64028 None.

**64029 FUTURE DIRECTIONS**

64030 None.

64031 **SEE ALSO**64032 [sem\\_clockwait\(\)](#), [sem\\_destroy\(\)](#), [sem\\_post\(\)](#), [sem\\_trywait\(\)](#)64033 XBD <[semaphore.h](#)>64034 **CHANGE HISTORY**

64035 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

64036 **Issue 6**64037 The `sem_init()` function is marked as part of the Semaphores option.64038 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
64039 implementation does not support the Semaphores option.64040 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with IEEE Std  
64041 1003.1d-1999.64042 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/116 is applied, updating the  
64043 DESCRIPTION to add the `sem_timedwait()` function for alignment with IEEE Std 1003.1d-1999.64044 **Issue 7**

64045 SD5-XSH-ERN-176 is applied.

64046 The `sem_init()` function is moved from the Semaphores option to the Base.

64047 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0526 [37] is applied.

64048 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0318 [972] is applied.

64049 **Issue 8**64050 Austin Group Defect 368 is applied, adding a statement that an unnamed semaphore may be  
64051 implemented using a file descriptor and adding the [EMFILE] and [ENFILE] errors.64052 Austin Group Defect 1216 is applied, adding `sem_clockwait()`.

64053 **NAME**

64054 sem\_open — initialize and open a named semaphore

64055 **SYNOPSIS**

64056 #include &lt;semaphore.h&gt;

64057 sem\_t \*sem\_open(const char \*name, int oflag, ...);

64058 **DESCRIPTION**

64059 The *sem\_open()* function shall establish a connection between a named semaphore and a process.  
 64060 A named semaphore may be implemented using a file descriptor. Following a call to *sem\_open()*  
 64061 with semaphore name *name*, the process may reference the semaphore associated with *name*  
 64062 using the semaphore handle returned from the call. This semaphore can be used in subsequent  
 64063 calls to *sem\_clockwait()*, *sem\_close()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, and *sem\_wait()*.  
 64064 The semaphore remains usable by this process until the semaphore is closed by a successful call  
 64065 to *sem\_close()*, *\_exit()*, or one of the *exec* functions.

64066 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to  
 64067 *sem\_open()*. The following flag bits may be set in *oflag*:

64068 **O\_CREAT** This flag is used to create a semaphore if it does not already exist. If **O\_CREAT** is  
 64069 set and the semaphore already exists, then **O\_CREAT** has no effect, except as noted  
 64070 under **O\_EXCL**. Otherwise, *sem\_open()* creates a named semaphore. The **O\_CREAT**  
 64071 flag requires a third and a fourth argument: *mode*, which is of type **mode\_t**, and  
 64072 *value*, which is of type **unsigned**. The semaphore is created with an initial value of  
 64073 *value*. Valid initial values for semaphores are less than or equal to  
 64074 {SEM\_VALUE\_MAX}.

64075 The user ID of the semaphore shall be set to the effective user ID of the process.  
 64076 The group ID of the semaphore shall be set to the effective group ID of the process;  
 64077 however, if the *name* argument is visible in the file system, the group ID may be set  
 64078 to the group ID of the containing directory. The permission bits of the semaphore  
 64079 are set to the value of the *mode* argument except those set in the file mode creation  
 64080 mask of the process. When bits in *mode* other than the file permission bits are  
 64081 specified, the effect is unspecified.

64082 After the semaphore named *name* has been created by *sem\_open()* with the  
 64083 **O\_CREAT** flag, other processes can connect to the semaphore by calling  
 64084 *sem\_open()* with the same value of *name*.

64085 **O\_EXCL** If **O\_EXCL** and **O\_CREAT** are set, *sem\_open()* fails if the semaphore *name* exists.  
 64086 The check for the existence of the semaphore and the creation of the semaphore if it  
 64087 does not exist are atomic with respect to other processes executing *sem\_open()* with  
 64088 **O\_EXCL** and **O\_CREAT** set. If **O\_EXCL** is set and **O\_CREAT** is not set, the effect is  
 64089 undefined.

64090 If flags other than **O\_CREAT** and **O\_EXCL** are specified in the *oflag* parameter, the  
 64091 effect is unspecified.

64092 The *name* argument points to a string naming a semaphore object. It is unspecified whether the  
 64093 name appears in the file system and is visible to functions that take pathnames as arguments.  
 64094 The *name* argument conforms to the construction rules for a pathname, except that the  
 64095 interpretation of <slash> characters other than the leading <slash> character in *name* is  
 64096 implementation-defined, and that the length limits for the *name* argument are implementation-  
 64097 defined and need not be the same as the pathname limits {PATH\_MAX} and {NAME\_MAX}. If  
 64098 *name* begins with the <slash> character, then processes calling *sem\_open()* with the same value of  
 64099 *name* shall refer to the same semaphore object, as long as that name has not been removed. If

- 64100 *name* does not begin with the <slash> character, the effect is implementation-defined.
- 64101 If a process makes multiple successful calls to *sem\_open()* with the same value for *name*, there  
64102 have been no intervening calls to *sem\_unlink()* for *name*, and at least one open handle for this  
64103 semaphore has not been closed with a *sem\_close()* call, it is implementation-defined whether the  
64104 same handle or a unique handle is returned for each such successful call.
- 64105 References to copies of the semaphore produce undefined results.
- 64106 **RETURN VALUE**
- 64107 Upon successful completion, the *sem\_open()* function shall return the address of the semaphore.  
64108 Otherwise, it shall return a value of SEM\_FAILED and set *errno* to indicate the error. The symbol  
64109 SEM\_FAILED is defined in the <semaphore.h> header. No successful return from *sem\_open()*  
64110 shall return the value SEM\_FAILED.
- 64111 **ERRORS**
- 64112 If any of the following conditions occur, the *sem\_open()* function shall return SEM\_FAILED and  
64113 set *errno* to the corresponding value:
- 64114 [EACCES] The named semaphore exists and the permissions specified by *oflag* are  
64115 denied, or the named semaphore does not exist and permission to create the  
64116 named semaphore is denied.
- 64117 [EEXIST] O\_CREAT and O\_EXCL are set and the named semaphore already exists.
- 64118 [EINTR] The *sem\_open()* operation was interrupted by a signal.
- 64119 [EINVAL] The *sem\_open()* operation is not supported for the given name, or O\_CREAT  
64120 was specified in *oflag* and *value* was greater than {SEM\_VALUE\_MAX}.
- 64121 [ENOENT] O\_CREAT is not set and the named semaphore does not exist.
- 64122 [ENOMEM] There is insufficient memory for the creation of the new named semaphore.
- 64123 [ENOSPC] There is insufficient space on a storage device for the creation of the new  
64124 named semaphore.
- 64125 If any of the following conditions occur, the *sem\_open()* function may return SEM\_FAILED and  
64126 set *errno* to the corresponding value:
- 64127 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by this  
64128 process.
- 64129 [ENAMETOOLONG]
- 64130 The length of the *name* argument exceeds {\_POSIX\_PATH\_MAX} on systems  
64131 XSI that do not support the XSI option or exceeds {\_XOPEN\_PATH\_MAX} on XSI  
64132 systems, or has a pathname component that is longer than  
64133 XSI {\_POSIX\_NAME\_MAX} on systems that do not support the XSI option or  
64134 longer than {\_XOPEN\_NAME\_MAX} on XSI systems.
- 64135 [ENFILE] Too many semaphore descriptors or file descriptors are currently open in the  
64136 system.



**64137 EXAMPLES**

64138 None.

**64139 APPLICATION USAGE**

64140 None.

**64141 RATIONALE**

64142 Early drafts required an error return value of `-1` with the type `sem_t *` for the `sem_open()`  
64143 function, which is not guaranteed to be portable across implementations. The revised text  
64144 provides the symbolic error code `SEM_FAILED` to eliminate the type conflict.

**64145 FUTURE DIRECTIONS**

64146 A future version might require the `sem_open()` and `sem_unlink()` functions to have semantics  
64147 similar to normal file system operations.

**64148 SEE ALSO**

64149 [\*semctl\(\)\*](#), [\*semget\(\)\*](#), [\*semop\(\)\*](#), [\*sem\\_clockwait\(\)\*](#), [\*sem\\_close\(\)\*](#), [\*sem\\_post\(\)\*](#), [\*sem\\_trywait\(\)\*](#), [\*sem\\_unlink\(\)\*](#)

64150 XBD <[\*\*semaphore.h\*\*](#)>

**64151 CHANGE HISTORY**

64152 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**64153 Issue 6**

64154 The `sem_open()` function is marked as part of the Semaphores option.

64155 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an  
64156 implementation does not support the Semaphores option.

64157 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with IEEE Std  
64158 1003.1d-1999.

64159 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/117 is applied, updating the  
64160 DESCRIPTION to add the `sem_timedwait()` function for alignment with IEEE Std 1003.1d-1999.

64161 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/118 is applied, updating the  
64162 DESCRIPTION to describe the conditions to return the same semaphore address on a call to  
64163 `sem_open()`. The words “and at least one previous successful `sem_open()` call for this semaphore  
64164 has not been matched with a `sem_close()` call” are added.

**64165 Issue 7**

64166 Austin Group Interpretation 1003.1-2001 #066 is applied, updating the `[ENOSPC]` error case and  
64167 adding the `[ENOMEM]` error case.

64168 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the `name` argument and  
64169 adding `[ENAMETOOLONG]` as a “may fail” error.

64170 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

64171 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the  
64172 user ID and group ID of the semaphore.

64173 The `sem_open()` function is moved from the Semaphores option to the Base.

64174 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0527 [37] is applied.

**64175 Issue 8**

64176 Austin Group Defect 368 is applied, adding a statement that a named semaphore may be  
64177 implemented using a file descriptor and changing the ERRORS section.

64178 Austin Group Defect 1216 is applied, adding `sem_clockwait()`.

64179  
64180  
64181

Austin Group Defect 1324 is applied, making it implementation-defined whether the same handle or a unique handle is returned when multiple successful calls to *sem\_open()* are made with the same value for *name*.

64182 **NAME**

64183 sem\_post — unlock a semaphore

64184 **SYNOPSIS**

64185 #include &lt;semaphore.h&gt;

64186 int sem\_post(sem\_t \*sem);

64187 **DESCRIPTION**64188 The *sem\_post()* function shall unlock the semaphore referenced by *sem* by performing a  
64189 semaphore unlock operation on that semaphore.64190 If the semaphore value resulting from this operation is positive, then no threads were blocked  
64191 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.64192 If the value of the semaphore resulting from this operation is zero, then one of the threads  
64193 blocked waiting for the semaphore shall be allowed to return successfully from its call to  
64194 *sem\_wait()*. If the Process Scheduling option is supported, the thread to be unblocked shall be  
64195 chosen in a manner appropriate to the scheduling policies and parameters in effect for the  
64196 blocked threads. In the case of the schedulers SCHED\_FIFO and SCHED\_RR, the highest  
64197 priority waiting thread shall be unblocked, and if there is more than one highest priority thread  
64198 blocked waiting for the semaphore, then the highest priority thread that has been waiting the  
64199 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread  
64200 to unblock is unspecified.64201 SS If the Process Sporadic Server option is supported, and the scheduling policy is  
64202 SCHED\_SPORADIC, the semantics are as per SCHED\_FIFO above.64203 The *sem\_post()* function shall be async-signal-safe and may be invoked from a signal-catching  
64204 function.64205 **RETURN VALUE**64206 If successful, the *sem\_post()* function shall return zero; otherwise, the function shall return -1  
64207 and set *errno* to indicate the error.64208 **ERRORS**64209 The *sem\_post()* function shall fail if:

64210 [EOVERFLOW] The maximum allowable value of the semaphore would be exceeded.

64211 The *sem\_post()* function may fail if:64212 [EINVAL] The *sem* argument does not refer to a valid semaphore.64213 **EXAMPLES**64214 See *sem\_clockwait()*.64215 **APPLICATION USAGE**

64216 None.

64217 **RATIONALE**

64218 None.

64219 **FUTURE DIRECTIONS**

64220 None.

64221 **SEE ALSO**64222 *semctl()*, *semget()*, *semop()*, *sem\_clockwait()*, *sem\_trywait()*

64223 XBD Section 4.15.2 (on page 104), &lt;semaphore.h&gt;

64224 **CHANGE HISTORY**

64225 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

64226 **Issue 6**

64227 The *sem\_post()* function is marked as part of the Semaphores option.

64228 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
64229 implementation does not support the Semaphores option.

64230 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std  
64231 1003.1d-1999.

64232 SCHED\_SPORADIC is added to the list of scheduling policies for which the thread that is to be  
64233 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

64234 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/119 is applied, updating the ERRORS  
64235 section so that the [EINVAL] error becomes optional.

64236 **Issue 7**

64237 Austin Group Interpretation 1003.1-2001 #156 is applied.

64238 The *sem\_post()* function is moved from the Semaphores option to the Base.

64239 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0528 [37] is applied.

64240 **Issue 8**

64241 Austin Group Defect 315 is applied, adding the [E\_OVERFLOW] error.

64242 **NAME**

64243       sem\_timedwait — lock a semaphore

64244 **SYNOPSIS**

64245       #include &lt;semaphore.h&gt;

64246       int sem\_timedwait(sem\_t \*restrict sem,  
64247                        const struct timespec \*restrict abstime);64248 **DESCRIPTION**64249       Refer to *sem\_clockwait()*.

64250 **NAME**

64251 sem\_trywait, sem\_wait — lock a semaphore

64252 **SYNOPSIS**

```
64253 #include <semaphore.h>
64254 int sem_trywait(sem_t *sem);
64255 int sem_wait(sem_t *sem);
```

64256 **DESCRIPTION**

64257 The *sem\_trywait()* function shall lock the semaphore referenced by *sem* only if the semaphore is  
 64258 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not  
 64259 lock the semaphore.

64260 The *sem\_wait()* function shall lock the semaphore referenced by *sem* by performing a semaphore  
 64261 lock operation on that semaphore. If the semaphore value is currently zero, then the calling  
 64262 thread shall not return from the call to *sem\_wait()* until it either locks the semaphore or the call is  
 64263 interrupted by a signal.

64264 Upon successful return, the state of the semaphore shall be locked and shall remain locked until  
 64265 the *sem\_post()* function is executed and returns successfully.

64266 The *sem\_wait()* function is interruptible by the delivery of a signal.

64267 **RETURN VALUE**

64268 The *sem\_trywait()* and *sem\_wait()* functions shall return zero if the calling process successfully  
 64269 performed the semaphore lock operation on the semaphore designated by *sem*. If the call was  
 64270 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a  
 64271 value of  $-1$  and set *errno* to indicate the error.

64272 **ERRORS**

64273 The *sem\_trywait()* function shall fail if:

64274 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the  
 64275 *sem\_trywait()* operation.

64276 The *sem\_trywait()* and *sem\_wait()* functions may fail if:

64277 [EDEADLK] A deadlock condition was detected.

64278 [EINTR] A signal interrupted this function.

64279 [EINVAL] The *sem* argument does not refer to a valid semaphore.

64280 **EXAMPLES**

64281 None.

64282 **APPLICATION USAGE**

64283 Applications using these functions may be subject to priority inversion, as discussed in XBD  
 64284 [Section 3.275](#) (on page 72).

64285 **RATIONALE**

64286 None.

64287 **FUTURE DIRECTIONS**

64288 None.

64289 **SEE ALSO**

64290 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem\\_clockwait\(\)](#), [sem\\_post\(\)](#)

64291 XBD [Section 3.275](#) (on page 72), [Section 4.15.2](#) (on page 104), [<semaphore.h>](#)

64292 **CHANGE HISTORY**

64293 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

64294 **Issue 6**

64295 The *sem\_trywait()* and *sem\_wait()* functions are marked as part of the Semaphores option.

64296 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
64297 implementation does not support the Semaphores option.

64298 The *sem\_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std  
64299 1003.1d-1999.

64300 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/121 is applied, updating the ERRORS  
64301 section so that the [EINVAL] error becomes optional.

64302 **Issue 7**

64303 SD5-XSH-ERN-54 is applied, removing the *sem\_wait()* function from the "shall fail" error cases.

64304 The *sem\_trywait()* and *sem\_wait()* functions are moved from the Semaphores option to the Base.  
64305 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0530 [37] is applied.

64306 **NAME**

64307 sem\_unlink — remove a named semaphore

64308 **SYNOPSIS**

64309 #include &lt;semaphore.h&gt;

64310 int sem\_unlink(const char \*name);

64311 **DESCRIPTION**

64312 The *sem\_unlink()* function shall remove the semaphore named by the string *name*. If the  
64313 semaphore named by *name* is currently referenced by other processes, then *sem\_unlink()* shall  
64314 have no effect on the state of the semaphore. If one or more processes have the semaphore open  
64315 when *sem\_unlink()* is called, destruction of the semaphore is postponed until all references to the  
64316 semaphore have been destroyed by calls to *sem\_close()*, *\_exit()*, or *exec*. Calls to *sem\_open()* to  
64317 recreate or reconnect to the semaphore refer to a new semaphore after *sem\_unlink()* is called. The  
64318 *sem\_unlink()* call shall not block until all references have been destroyed; it shall return  
64319 immediately.

64320 **RETURN VALUE**

64321 Upon successful completion, the *sem\_unlink()* function shall return a value of 0. Otherwise, the  
64322 semaphore shall not be changed and the function shall return a value of -1 and set *errno* to  
64323 indicate the error.

64324 **ERRORS**64325 The *sem\_unlink()* function shall fail if:

64326 [EACCES] Permission is denied to unlink the named semaphore.

64327 [ENOENT] The named semaphore does not exist.

64328 The *sem\_unlink()* function may fail if:

64329 [ENAMETOOLONG]

64330 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems  
64331 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI  
64332 systems, or has a pathname component that is longer than  
64333 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or  
64334 longer than `{_XOPEN_NAME_MAX}` on XSI systems. A call to *sem\_unlink()*  
64335 with a *name* argument that contains the same semaphore name as was  
64336 previously used in a successful *sem\_open()* call shall not give an  
64337 [ENAMETOOLONG] error.

64338 **EXAMPLES**

64339 None.

64340 **APPLICATION USAGE**

64341 None.

64342 **RATIONALE**

64343 None.

64344 **FUTURE DIRECTIONS**

64345 A future version might require the *sem\_open()* and *sem\_unlink()* functions to have semantics  
64346 similar to normal file system operations.

64347 **SEE ALSO**64348 *semctl()*, *semget()*, *semop()*, *sem\_close()*, *sem\_open()*

64349 XBD &lt;semaphore.h&gt;



64350 **CHANGE HISTORY**

64351 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

64352 **Issue 6**

64353 The *sem\_unlink()* function is marked as part of the Semaphores option.

64354 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
64355 implementation does not support the Semaphores option.

64356 **Issue 7**

64357 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a  
64358 ``shall fail'' to a ``may fail'' error.

64359 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

64360 The *sem\_unlink()* function is moved from the Semaphores option to the Base.

64361 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0531 [37] is applied.

64362 **NAME**

64363           sem\_wait — lock a semaphore

64364 **SYNOPSIS**

64365           #include <semaphore.h>

64366           int sem\_wait(sem\_t \*sem);

64367 **DESCRIPTION**

64368           Refer to *sem\_trywait()*.

64369 **NAME**

64370 semctl — XSI semaphore control operations

64371 **SYNOPSIS**

```
64372 XSI #include <sys/sem.h>
64373 int semctl(int semid, int semnum, int cmd, ...);
```

64374 **DESCRIPTION**

64375 The *semctl()* function operates on XSI semaphores (see XBD [Section 4.20](#), on page 108). It is  
 64376 unspecified whether this function interoperates with the realtime interprocess communication  
 64377 facilities defined in [Section 2.8](#) (on page 527).

64378 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.  
 64379 The fourth argument is optional and depends upon the operation requested. If required, it is of  
 64380 type **union semun**, which the application shall explicitly declare:

```
64381 union semun {
64382     int val;
64383     struct semid_ds *buf;
64384     unsigned short *array;
64385 } arg;
```

64386 Each operation shall be performed atomically.

64387 The following semaphore control operations as specified by *cmd* are executed with respect to the  
 64388 semaphore specified by *semid* and *semnum*. The level of permission required for each operation  
 64389 is shown with each command; see [Section 2.7](#) (on page 526). The symbolic names for the values  
 64390 of *cmd* are defined in the `<sys/sem.h>` header:

64391	GETVAL	Return the value of <i>semval</i> ; see <code>&lt;sys/sem.h&gt;</code> . Requires read permission.
64392	SETVAL	Set the value of <i>semval</i> to <i>arg.val</i> , where <i>arg</i> is the value of the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. Also, the <i>sem_ctime</i> timestamp shall be set to the current time, as described in <a href="#">Section 2.7.1</a> (on page 526). Requires alter permission; see <a href="#">Section 2.7</a> (on page 526).
64393		
64394		
64395		
64396		
64397	GETPID	Return the value of <i>sempid</i> . Requires read permission.
64398	GETNCNT	Return the value of <i>semmcnt</i> . Requires read permission.
64399	GETZCNT	Return the value of <i>semzcnt</i> . Requires read permission.

64400 The following values of *cmd* operate on each *semval* in the set of semaphores:

64401	GETALL	Return the value of <i>semval</i> for each semaphore in the semaphore set and place into the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . Requires read permission.
64402		
64403		
64404	SETALL	Set the value of <i>semval</i> for each semaphore in the semaphore set according to the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> values corresponding to each specified semaphore in all processes are cleared. Also, the <i>sem_ctime</i> timestamp shall be set to the current time, as described in <a href="#">Section 2.7.1</a> (on page 526). Requires alter permission.
64405		
64406		
64407		
64408		
64409		

64410		The following values of <i>cmd</i> are also available:
64411	IPC_STAT	Place the current value of each member of the <b>semid_ds</b> data structure associated with <i>semid</i> into the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . The contents of this structure are defined in <b>&lt;sys/sem.h&gt;</b> . Requires read permission.
64412		
64413		
64414		
64415	IPC_SET	Set the value of the following members of the <b>semid_ds</b> data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :
64416		
64417		
64418		<code>sem_perm.uid</code>
64419		<code>sem_perm.gid</code>
64420		<code>sem_perm.mode</code>
64421		The mode bits specified in <a href="#">Section 2.7.1</a> (on page 526) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified. The <i>sem_ctime</i> timestamp shall be set to the current time, as described in <a href="#">Section 2.7.1</a> (on page 526).
64422		
64423		
64424		
64425		This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .
64426		
64427		
64428		
64429	IPC_RMID	Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and <b>semid_ds</b> data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the <b>semid_ds</b> data structure associated with <i>semid</i> .
64430		
64431		
64432		
64433		
64434		
64435	<b>RETURN VALUE</b>	
64436		If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:
64437	GETVAL	The value of <i>semval</i> .
64438	GETPID	The value of <i>sempid</i> .
64439	GETNCNT	The value of <i>semmcnt</i> .
64440	GETZCNT	The value of <i>semzcnt</i> .
64441	All others	0.
64442		Otherwise, <i>semctl()</i> shall return <code>-1</code> and set <i>errno</i> to indicate the error.
64443	<b>ERRORS</b>	
64444		The <i>semctl()</i> function shall fail if:
64445	[EACCES]	Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 526).
64446		
64447	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.
64448		
64449		
64450	[EPERM]	The argument <i>cmd</i> is equal to <code>IPC_RMID</code> or <code>IPC_SET</code> and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i> .
64451		
64452		
64453		

64454 [ERANGE] The argument *cmd* is equal to SETVAL or SETALL and the value to which  
64455 *semval* is to be set is greater than the system-imposed maximum.

#### 64456 EXAMPLES

64457 Refer to *semop()*.

#### 64458 APPLICATION USAGE

64459 The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a  
64460 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for  
64461 backwards-compatibility.

64462 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
64463 Application developers who need to use IPC should design their applications so that modules  
64464 using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to use the  
64465 alternative interfaces.

#### 64466 RATIONALE

64467 None.

#### 64468 FUTURE DIRECTIONS

64469 None.

#### 64470 SEE ALSO

64471 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), *semget()*, *semop()*, *sem\_close()*, *sem\_destroy()*,  
64472 *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_trywait()*, *sem\_unlink()*

64473 [XBD Section 4.20](#) (on page 108), [<sys/sem.h>](#)

#### 64474 CHANGE HISTORY

64475 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 64476 Issue 5

64477 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
64478 DIRECTIONS to the APPLICATION USAGE section.

#### 64479 Issue 7

64480 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0532 [345], XSH/TC1-2008/0533 [345],  
64481 XSH/TC1-2008/0534 [345], and XSH/TC1-2008/0535 [335] are applied.

64482 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0319 [532] is applied.

64483 **NAME**

64484 semget — get set of XSI semaphores

64485 **SYNOPSIS**

```
64486 XSI      #include <sys/sem.h>
64487      int semget(key_t key, int nsems, int semflg);
```

64488 **DESCRIPTION**

64489 The *semget()* function operates on XSI semaphores (see XBD [Section 4.20](#), on page 108). It is  
 64490 unspecified whether this function interoperates with the realtime interprocess communication  
 64491 facilities defined in [Section 2.8](#) (on page 527).

64492 The *semget()* function shall return the semaphore identifier associated with *key*.

64493 A semaphore identifier with its associated **semid\_ds** data structure and its associated set of  
 64494 *nsems* semaphores (see [<sys/sem.h>](#)) is created for *key* if one of the following is true:

- 64495 • The argument *key* is equal to `IPC_PRIVATE`.
- 64496 • The argument *key* does not already have a semaphore identifier associated with it and  
 64497 (*semflg* & `IPC_CREAT`) is non-zero.

64498 Upon creation, the **semid\_ds** data structure associated with the new semaphore identifier is  
 64499 initialized as follows:

- 64500 • In the operation permissions structure *sem\_perm.cuid*, *sem\_perm.uid*, *sem\_perm.cgid*, and  
 64501 *sem\_perm.gid* shall be set to the effective user ID and effective group ID, respectively, of the  
 64502 calling process.
- 64503 • The low-order 9 bits of *sem\_perm.mode* shall be set to the low-order 9 bits of *semflg*.
- 64504 • The variable *sem\_nsems* shall be set to the value of *nsems*.
- 64505 • The variable *sem\_otime* shall be set to 0 and *sem\_ctime* shall be set to the current time, as  
 64506 described in [Section 2.7.1](#) (on page 526).

64507 Upon creation, the value of the *semval*, *sempid*, *semncnt*, and *semzcnt* members of all *nsems*  
 64508 semaphores in the associated semaphore set shall be set to zero.

64509 **RETURN VALUE**

64510 Upon successful completion, *semget()* shall return a non-negative integer, namely a semaphore  
 64511 identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

64512 **ERRORS**

64513 The *semget()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 64514 | [EACCES] | A semaphore identifier exists for <i>key</i> , but operation permission as specified by the low-order 9 bits of <i>semflg</i> would not be granted; see <a href="#">Section 2.7</a> (on page 526).  |
| 64515 |          |   |
| 64516 |          |   |
| 64517 | [EEXIST] | A semaphore identifier exists for the argument <i>key</i> but $((semflg \& IPC\_CREAT) \&\& (semflg \& IPC\_EXCL))$ is non-zero.  |
| 64518 |          |   |
| 64519 | [EINVAL] | The value of <i>nsems</i> is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> and <i>nsems</i> is not equal to 0. |
| 64520 |          |   |
| 64521 |          |   |
| 64522 |          |   |

64523 [ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg*  
64524 &IPC\_CREAT) is equal to 0.

64525 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the  
64526 maximum number of allowed semaphores system-wide would be exceeded.

#### 64527 EXAMPLES

64528 Refer to *semop()*.

#### 64529 APPLICATION USAGE

64530 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
64531 Application developers who need to use IPC should design their applications so that modules  
64532 using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to use the  
64533 alternative interfaces.

#### 64534 RATIONALE

64535 None.

#### 64536 FUTURE DIRECTIONS

64537 None.

#### 64538 SEE ALSO

64539 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), *ftok()*, *semctl()*, *semop()*, *sem\_close()*,  
64540 *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_trywait()*, *sem\_unlink()*

64541 XBD [Section 4.20](#) (on page 108), [<sys/sem.h>](#)

#### 64542 CHANGE HISTORY

64543 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 64544 Issue 5

64545 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
64546 DIRECTIONS to a new APPLICATION USAGE section.

#### 64547 Issue 6

64548 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/122 is applied, updating the  
64549 DESCRIPTION from “each semaphore in the set shall not be initialized” to “each semaphore in  
64550 the set need not be initialized”.

#### 64551 Issue 7

64552 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0536 [335,439] and  
64553 XSH/TC1-2008/0537 [344] are applied.

#### 64554 Issue 8

64555 Austin Group Defect 377 is applied, adding a requirement that the value of the *semval*, *sempid*,  
64556 *semncnt*, and *semzcnt* members of all semaphores in a semaphore set be initialized to zero when a  
64557 call to *semget()* creates a semaphore set.

64558 **NAME**

64559 semop — XSI semaphore operations

64560 **SYNOPSIS**

```
64561 XSI #include <sys/sem.h>
64562 int semop(int semid, struct sembuf *sops, size_t nsops);
```

64563 **DESCRIPTION**

64564 The *semop()* function operates on XSI semaphores (see XBD [Section 4.20](#), on page 108). It is  
 64565 unspecified whether this function interoperates with the realtime interprocess communication  
 64566 facilities defined in [Section 2.8](#) (on page 527).

64567 The *semop()* function shall perform atomically a user-defined array of semaphore operations in  
 64568 array order on the set of semaphores associated with the semaphore identifier specified by the  
 64569 argument *semid*.

64570 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The  
 64571 implementation shall not modify elements of this array unless the application uses  
 64572 implementation-defined extensions.

64573 The argument *nsops* is the number of such structures in the array.

64574 Each structure, **sembuf**, includes the following members:

Member Type	Member Name	Description
<b>unsigned short</b>	<i>sem_num</i>	Semaphore number.
<b>short</b>	<i>sem_op</i>	Semaphore operation.
<b>short</b>	<i>sem_flg</i>	Operation flags.

64575 Each semaphore operation specified by *sem\_op* is performed on the corresponding semaphore  
 64580 specified by *semid* and *sem\_num*.

64581 If all of the semaphore operations complete successfully, *semop()* shall return 0. If a semaphore  
 64582 operation fails or blocks, all changes to *semadj* and *semval* values performed by completed  
 64583 semaphore operations in the array before the operation that failed or blocked shall be undone  
 64584 before *semop()* returns or blocks, respectively. When a semaphore operation blocks, the change  
 64585 to *semncnt* or *semzcnt* indicating which semaphore operation blocked shall not be undone until  
 64586 the operation unblocks.

64587 For each operation in the array of semaphore operations, the variable *sem\_op* specifies one of  
 64588 three semaphore operations:

- 64589 1. If *sem\_op* is a negative integer and the calling process has alter permission, one of the  
 64590 following shall occur:
  - 64591 • If *semval* (see `<sys/sem.h>`) is greater than or equal to the absolute value of *sem\_op*,  
 64592 the absolute value of *sem\_op* shall be subtracted from *semval*. Also, if (*sem\_flg* &  
 64593 SEM\_UNDO) is non-zero, the absolute value of *sem\_op* shall be added to the *semadj*  
 64594 value of the calling process for the specified semaphore. If this is not the last  
 64595 operation in the array of semaphore operations to be performed, processing shall  
 64596 continue with the next operation in the array.
  - 64597 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* & IPC\_NOWAIT) is  
 64598 non-zero, the operation shall fail with *errno* set to [EAGAIN].



- 64599 • If *semval* is less than the absolute value of *sem\_op* and (*sem\_flg* & IPC\_NOWAIT) is 0,  
 64600 *semop()* shall increment the *semncnt* associated with the specified semaphore and  
 64601 suspend execution of the calling thread (block) until one of the following conditions  
 64602 occurs:
- 64603 — The value of *semval* changes. When this occurs, the value of *semncnt* associated  
 64604 with the specified semaphore shall be decremented and the array of  
 64605 semaphore operations shall be reevaluated.
  - 64606 — The *semid* for which the calling thread is awaiting action is removed from the  
 64607 system. When this occurs, the operation shall fail with *errno* set to [EIDRM].
  - 64608 — The calling thread receives a signal that is to be caught. When this occurs, the  
 64609 value of *semncnt* associated with the specified semaphore shall be  
 64610 decremented, and the calling thread shall resume execution in the manner  
 64611 prescribed in *sigaction()*.
- 64612 2. If *sem\_op* is a positive integer and the calling process has alter permission, the value of  
 64613 *sem\_op* shall be added to *semval* and, if (*sem\_flg* & SEM\_UNDO) is non-zero, the value of  
 64614 *sem\_op* shall be subtracted from the *semadj* value of the calling process for the specified  
 64615 semaphore. If this is not the last operation in the array of semaphore operations to be  
 64616 performed, processing shall continue with the next operation in the array.
- 64617 3. If *sem\_op* is 0 and the calling process has read permission, one of the following shall occur:
- 64618 • If *semval* is 0, this is a successful operation. If this is not the last operation in the  
 64619 array of semaphore operations to be performed, processing shall continue with the  
 64620 next operation in the array.
  - 64621 • If *semval* is non-zero and (*sem\_flg* & IPC\_NOWAIT) is non-zero, the operation shall  
 64622 fail with *errno* set to [EAGAIN].
  - 64623 • If *semval* is non-zero and (*sem\_flg* & IPC\_NOWAIT) is 0, *semop()* shall increment the  
 64624 *semzcnt* associated with the specified semaphore and suspend execution of the  
 64625 calling thread (block) until one of the following occurs:
    - 64626 — The value of *semval* changes. When this occurs, the value of *semzcnt* associated  
 64627 with the specified semaphore shall be decremented and the array of  
 64628 semaphore operations shall be reevaluated.
    - 64629 — The *semid* for which the calling thread is awaiting action is removed from the  
 64630 system. When this occurs, the operation shall fail with *errno* set to [EIDRM].
    - 64631 — The calling thread receives a signal that is to be caught. When this occurs, the  
 64632 value of *semzcnt* associated with the specified semaphore shall be  
 64633 decremented, and the calling thread shall resume execution in the manner  
 64634 prescribed in *sigaction()*.

64635 Upon successful completion of all of the semaphore operations specified in the array pointed to  
 64636 by *sops*, the value of *sempid* for each semaphore specified in the array shall be set to the process  
 64637 ID of the calling process and the *sem\_otime* timestamp associated with the semaphore set shall be  
 64638 set to the current time, as described in [Section 2.7.1](#) (on page 526).

#### 64639 RETURN VALUE

64640 Upon successful completion, *semop()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 64641 indicate the error.

64642 **ERRORS**64643 The *semop()* function shall fail if:

64644	[E2BIG]	The value of <i>nsops</i> is greater than the system-imposed maximum.
64645	[EACCES]	Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 526).
64647	[EAGAIN]	The operation would result in suspension of the calling thread but ( <i>sem_flg</i> & <i>IPC_NOWAIT</i> ) is non-zero.
64649	[EFBIG]	The value of <i>sem_num</i> is greater than or equal to the number of semaphores in the set associated with <i>semid</i> .
64651	[EIDRM]	The semaphore identifier <i>semid</i> is removed from the system.
64652	[EINTR]	The <i>semop()</i> function was interrupted by a signal.
64653	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a <i>SEM_UNDO</i> would exceed the system-imposed limit.
64656	[ENOSPC]	The limit on the number of individual processes requesting a <i>SEM_UNDO</i> would be exceeded.
64658	[ERANGE]	An operation would cause a <i>semval</i> to overflow the system-imposed limit, or an operation would cause a <i>semadj</i> value to overflow the system-imposed limit.

64661 **EXAMPLES**64662 **Setting Values in Semaphores**64663 The following example sets the values of the two semaphores associated with the *semid* identifier to the values contained in the *sb* array.

```

64665 #include <sys/sem.h>
64666 ...
64667 int semid;
64668 struct sembuf sb[2];
64669 int nsops = 2;
64670 int result;

64671 // Code to initialize semid.
64672 ...

64673 // Adjust value of semaphore in the semaphore array semid.
64674 sb[0].sem_num = 0;
64675 sb[0].sem_op = -1;
64676 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
64677 sb[1].sem_num = 1;
64678 sb[1].sem_op = 1;
64679 sb[1].sem_flg = 0;

64680 result = semop(semid, sb, nsops);

```

64681 **Creating a Semaphore Identifier**

64682 The following example gets a semaphore key using the *ftok()* function, then creates or uses an  
 64683 existing semaphore set associated with that key using the *semget()* function.

64684 If this process creates the semaphore set, the program uses a call to *semop()* to initialize it to the  
 64685 value in the *sbuf* array. The number of processes that can execute concurrently is set to 2.

64686 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM\_UNDO  
 64687 option releases the semaphore when the process exits, waiting until there are less than two  
 64688 processes running concurrently.

```

64689 #include <errno.h>
64690 #include <stdio.h>
64691 #include <stdlib.h>
64692 #include <sys/sem.h>
64693 #include <sys/stat.h>
64694 ...
64695 struct sembuf sbuf;
64696 int semid;
64697 key_t semkey;
64698 ...
64699 // Get a key for the semaphore set.
64700 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
64701     perror("IPC error: ftok");
64702     exit(1);
64703 }
64704 // Create the semaphore set associated with this key
64705 if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
64706     S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1) {
64707     // Initialize the semaphore.
64708     sbuf.sem_num = 0;
64709     sbuf.sem_op = 2; // Set the number of runs without queuing.
64710     sbuf.sem_flg = 0;
64711     if (semop(semid, &sbuf, 1) == -1) {
64712         perror("IPC error: semop");
64713         exit(1);
64714     }
64715 } else if (errno == EEXIST) {
64716     // The semaphore set already exists; get its semaphore ID.
64717     if ((semid = semget(semkey, 0, 0)) == -1) {
64718         perror("IPC error 1: semget");
64719         exit(1);
64720     }
64721 } else {
64722     perror("IPC error 2: semget");
64723     exit(1);
64724 }
64725 // Since the semget() initialized the semaphore to 0, the
64726 // following semop() will block until the creating process

```

```

64727 // completes the initialization above. Processes will also
64728 // block in the following semop() call if two other processes
64729 // have already passed this point and are still running.
64730 sbuf.sem_num = 0;
64731 sbuf.sem_op = -1;
64732 sbuf.sem_flg = SEM_UNDO;
64733 if (semop(semid, &sbuf, 1) == -1) {
64734     perror("IPC Error: semop");
64735     exit(1);
64736 }

```

#### 64737 APPLICATION USAGE

64738 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
 64739 Application developers who need to use IPC should design their applications so that modules  
 64740 using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to use the  
 64741 alternative interfaces.

#### 64742 RATIONALE

64743 None.

#### 64744 FUTURE DIRECTIONS

64745 None.

#### 64746 SEE ALSO

64747 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), *exec*, *exit()*, *fork()*, *semctl()*, *semget()*,  
 64748 *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*, *sem\_trywait()*,  
 64749 *sem\_unlink()*

64750 XBD [Section 4.20](#) (on page 108), [<sys/ipc.h>](#), [<sys/sem.h>](#), [<sys/types.h>](#)

#### 64751 CHANGE HISTORY

64752 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 64753 Issue 5

64754 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
 64755 DIRECTIONS to a new APPLICATION USAGE section.

#### 64756 Issue 7

64757 SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the  
 64758 operations in *sops* will be performed when there are multiple operations.

64759 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0538 [329,429], XSH/TC1-2008/0539  
 64760 [345,428], XSH/TC1-2008/0540 [329,429], XSH/TC1-2008/0541 [335], and XSH/TC1-2008/0542  
 64761 [291,429] are applied.

#### 64762 Issue 8

64763 Austin Group Defect 377 is applied, changing the EXAMPLES section.

64764 Austin Group Defect 628 is applied, clarifying how *semop()* behaves when there is more than  
 64765 one operation in the array specified by *sops*.

64766 **NAME**

64767 send — send a message on a socket

64768 **SYNOPSIS**

64769 #include &lt;sys/socket.h&gt;

64770 ssize\_t send(int socket, const void \*buffer, size\_t length, int flags);

64771 **DESCRIPTION**

64772 The *send()* function shall initiate transmission of a message from the specified socket to its peer.  
 64773 The *send()* function shall send a message only when the socket is connected. If the socket is a  
 64774 connectionless-mode socket, the message shall be sent to the pre-specified peer address.

64775 The *send()* function takes the following arguments:64776 *socket* Specifies the socket file descriptor.64777 *buffer* Points to the buffer containing the message to send.64778 *length* Specifies the length of the message in bytes.

64779 *flags* Specifies the type of message transmission. Values of this argument are  
 64780 formed by logically OR'ing zero or more of the following flags:

64781 MSG\_EOR Terminates a record (if supported by the protocol).

64782 MSG\_OOB Sends out-of-band data on sockets that support out-of-  
 64783 band communications. The significance and semantics  
 64784 of out-of-band data are protocol-specific.

64785 MSG\_NOSIGNAL Requests not to send the SIGPIPE signal if an attempt to  
 64786 send is made on a stream-oriented socket that is no  
 64787 longer connected. The [EPIPE] error shall still be  
 64788 returned.

64789 The length of the message to be sent is specified by the *length* argument. If the message is too  
 64790 long to pass through the underlying protocol, *send()* shall fail and no data shall be transmitted.

64791 Successful completion of a call to *send()* does not guarantee delivery of the message. A return  
 64792 value of  $-1$  indicates only locally-detected errors.

64793 If space is not available at the sending socket to hold the message to be transmitted, and the  
 64794 socket file descriptor does not have O\_NONBLOCK set, *send()* shall block until space is  
 64795 available or a timeout occurs (see SO\_SNDTIMEO in [Section 2.10.16](#), on page 554). If space is  
 64796 not available at the sending socket to hold the message to be transmitted, and the socket file  
 64797 descriptor does have O\_NONBLOCK set, *send()* shall fail. The *select()* and *poll()* functions can  
 64798 be used to determine when it is possible to send more data.

64799 The socket in use may require the process to have appropriate privileges to use the *send()*  
 64800 function.

64801 **RETURN VALUE**

64802 Upon successful completion, *send()* shall return the number of bytes sent. Otherwise,  $-1$  shall be  
 64803 returned and *errno* set to indicate the error.

64804 **ERRORS**64805 The *send()* function shall fail if:

64806 [EAGAIN] or [EWOULDBLOCK]

64807 The socket's file descriptor is marked O\_NONBLOCK and the requested  
 64808 operation would block. See also SO\_SNDTIMEO in [Section 2.10.16](#) (on page  
 64809 554).

64810	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
64811	[ECONNRESET]	A connection was forcibly closed by a peer.
64812	[EDESTADDRREQ]	
64813		The socket is not connection-mode and no peer address is set.
64814	[EINTR]	A signal interrupted <i>send()</i> before any data was transmitted.
64815	[EMSGSIZE]	The message is too large to be sent all at once, as the socket requires.
64816	[ENOTCONN]	The socket is not connected.
64817	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
64818	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
64819		
64820	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.
64821		
64822		
64823		
64824		The <i>send()</i> function may fail if:
64825	[EACCES]	The calling process does not have appropriate privileges.
64826	[EIO]	An I/O error occurred while reading from or writing to the file system.
64827	[ENETDOWN]	The local network interface used to reach the destination is down.
64828	[ENETUNREACH]	
64829		No route to the network is present.
64830	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.

**EXAMPLES**

64831 None.  
64832

**APPLICATION USAGE**

64834 If the *socket* argument refers to a connection-mode socket, the *send()* function is equivalent to *sendto()* (with any value for the *dest\_addr* and *dest\_len* arguments, as they are ignored in this case). If the *socket* argument refers to a socket and the *flags* argument is 0, the *send()* function is equivalent to *write()*.  
64835  
64836  
64837

**RATIONALE**

64838 None.  
64839

**FUTURE DIRECTIONS**

64840 None.  
64841

**SEE ALSO**

64842 *connect()*, *getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *sendmsg()*, *sendto()*,  
64843 *setsockopt()*, *shutdown()*, *socket()*, *write()*  
64844

64845 XBD <[sys/socket.h](#)>

**CHANGE HISTORY**

64846 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.  
64847

64848 **Issue 7**

64849 Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify  
64850 the behavior when the socket is a connectionless-mode socket.

64851 The MSG\_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended  
64852 API Set Part 2.

64853 The [EPIPE] error is modified.

64854 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0543 [463] is applied.

64855 **Issue 8**

64856 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to  
64857 [Section 2.10.16](#) (on page 554).

64858 **NAME**

64859 sendmsg — send a message on a socket using a message structure

64860 **SYNOPSIS**

64861 #include &lt;sys/socket.h&gt;

64862 ssize\_t sendmsg(int socket, const struct msghdr \*message, int flags);

64863 **DESCRIPTION**

64864 The *sendmsg()* function shall send a message through a connection-mode or connectionless-mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the address specified by **msghdr** if no pre-specified peer address has been set. If a peer address has been pre-specified, either the message shall be sent to the address specified in **msghdr** (overriding the pre-specified peer address), or the function shall return  $-1$  and set *errno* to [EISCONN]. If the socket is connection-mode, the destination address in **msghdr** shall be ignored.

64871 The *sendmsg()* function takes the following arguments:

64872	<i>socket</i>	Specifies the socket file descriptor.
64873	<i>message</i>	Points to a <b>msghdr</b> structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored.
64874		
64875		
64876	<i>flags</i>	Specifies the type of message transmission. The application may specify 0 or the following flag:
64877		
64878	MSG_EOR	Terminates a record (if supported by the protocol).
64879	MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
64880		
64881		
64882	MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.
64883		
64884		
64885		

64886 The *msg\_iov* and *msg\_iovlen* fields of *message* specify zero or more buffers containing the data to be sent. *msg\_iov* points to an array of **iovec** structures; *msg\_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov\_base* field specifies a storage area and the *iov\_len* field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by *msg\_iov* is sent in turn.

64891 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A return value of  $-1$  indicates only locally-detected errors.

64893 If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O\_NONBLOCK set, the *sendmsg()* function shall block until space is available or a timeout occurs (see SO\_SNDTIMEO in [Section 2.10.16](#), on page 554). If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O\_NONBLOCK set, the *sendmsg()* function shall fail.

64898 If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendmsg()* shall fail if the SO\_BROADCAST option is not set for the socket.

64900 The socket in use may require the process to have appropriate privileges to use the *sendmsg()* function.



64902 **RETURN VALUE**

64903 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise, `-1`  
 64904 shall be returned and *errno* set to indicate the error.

64905 **ERRORS**

64906 The *sendmsg()* function shall fail if:

64907 [EAGAIN] or [EWOULDBLOCK]

64908 The socket's file descriptor is marked `O_NONBLOCK` and the requested  
 64909 operation would block. See also `SO_SNDTIMEO` in [Section 2.10.16](#) (on page  
 64910 554).

64911 [EAFNOSUPPORT]

64912 Addresses in the specified address family cannot be used with this socket.

64913 [EBADF] The *socket* argument is not a valid file descriptor.

64914 [ECONNRESET] A connection was forcibly closed by a peer.

64915 [EINTR] A signal interrupted *sendmsg()* before any data was transmitted.

64916 [EINVAL] The sum of the *iov\_len* values overflows an `ssize_t`.

64917 [EMSGSIZE] The message is too large to be sent all at once (as the socket requires), or the  
 64918 *msg\_iovlen* member of the `msghdr` structure pointed to by *message* is less than  
 64919 or equal to 0 or is greater than `{IOV_MAX}`.

64920 [ENOTCONN] The socket is connection-mode but is not connected.

64921 [ENOTSOCK] The *socket* argument does not refer to a socket.

64922 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
 64923 more of the values set in *flags*.

64924 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
 64925 no longer connected. In the latter case, and if the socket is of type  
 64926 `SOCK_STREAM` or `SOCK_SEQPACKET` and the `MSG_NOSIGNAL` flag is not  
 64927 set, the `SIGPIPE` signal is generated to the calling thread.

64928 If the address family of the socket is `AF_UNIX`, then *sendmsg()* shall fail if:

64929 [EIO] An I/O error occurred while reading from or writing to the file system.

64930 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname  
 64931 in the socket address.

64932 [ENAMETOOLONG]

64933 The length of a component of a pathname is longer than `{NAME_MAX}`.

64934 [ENOENT]

64935 A component of the pathname does not name an existing file or the path name  
 is an empty string.

64936 [ENOTDIR]

64937 A component of the path prefix of the pathname in the socket address names  
 64938 an existing file that is neither a directory nor a symbolic link to a directory, or  
 64939 the pathname in the socket address contains at least one non-`<slash>` character  
 64940 and ends with one or more trailing `<slash>` characters and the last pathname  
 64941 component names an existing file that is neither a directory nor a symbolic  
 link to a directory.

64942 The *sendmsg()* function may fail if:

64943 [EACCES] Search permission is denied for a component of the path prefix; or write access  
64944 to the named socket is denied.

64945 [EDESTADDRREQ]  
64946 The socket is not connection-mode and does not have its peer address set, and  
64947 no destination address was specified.

64948 [EHOSTUNREACH]  
64949 The destination host cannot be reached (probably because the host is down or  
64950 a remote router cannot reach it).

64951 [EIO] An I/O error occurred while reading from or writing to the file system.

64952 [EISCONN] A destination address was specified and the socket is already connected.

64953 [ENETDOWN] The local network interface used to reach the destination is down.

64954 [ENETUNREACH]  
64955 No route to the network is present.

64956 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

64957 [ENOMEM] Insufficient memory was available to fulfill the request.

64958 If the address family of the socket is AF\_UNIX, then *sendmsg()* may fail if:

64959 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
64960 resolution of the pathname in the socket address.

64961 [ENAMETOOLONG]  
64962 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
64963 symbolic link produced an intermediate result with a length that exceeds  
64964 {PATH\_MAX}.

**64965 EXAMPLES**

64966 Done.

**64967 APPLICATION USAGE**

64968 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.

**64969 RATIONALE**

64970 None.

**64971 FUTURE DIRECTIONS**

64972 None.

**64973 SEE ALSO**

64974 *getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*,  
64975 *shutdown()*, *socket()*

64976 XBD <[sys/socket.h](#)>

**64977 CHANGE HISTORY**

64978 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

64979 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
64980 [ELOOP] error condition is added.

64981 **Issue 7**

64982 Austin Group Interpretation 1003.1-2001 #073 is applied, updating the DESCRIPTION.

64983 Austin Group Interpretation 1003.1-2001 #143 is applied.

64984 The MSG\_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended  
64985 API Set Part 2.

64986 The [EPIPE] error is modified.

64987 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0544 [324] is applied.

64988 **Issue 8**

64989 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to  
64990 [Section 2.10.16](#) (on page 554).

64991 **NAME**

64992 sendto — send a message on a socket

64993 **SYNOPSIS**

```
64994 #include <sys/socket.h>
64995 ssize_t sendto(int socket, const void *message, size_t length,
64996               int flags, const struct sockaddr *dest_addr,
64997               socklen_t dest_len);
```

64998 **DESCRIPTION**64999 The *sendto()* function shall send a message through a connection-mode or connectionless-mode  
65000 socket.

65001 If the socket is a connectionless-mode socket, the message shall be sent to the address specified  
65002 by *dest\_addr* if no pre-specified peer address has been set. If a peer address has been pre-  
65003 specified, either the message shall be sent to the address specified by *dest\_addr* (overriding the  
65004 pre-specified peer address), or the function shall return  $-1$  and set *errno* to [EISCONN].

65005 If the socket is connection-mode, *dest\_addr* shall be ignored.65006 The *sendto()* function takes the following arguments:

65007	<i>socket</i>	Specifies the socket file descriptor.
65008	<i>message</i>	Points to a buffer containing the message to be sent.
65009	<i>length</i>	Specifies the size of the message in bytes.
65010	<i>flags</i>	Specifies the type of message transmission. Values of this argument are 65011 formed by logically OR'ing zero or more of the following flags:
65012	MSG_EOR	Terminates a record (if supported by the protocol).
65013	MSG_OOB	Sends out-of-band data on sockets that support out-of- 65014 band data. The significance and semantics of out-of- 65015 band data are protocol-specific.
65016	MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to 65017 send is made on a stream-oriented socket that is no 65018 longer connected. The [EPIPE] error shall still be 65019 returned.
65020	<i>dest_addr</i>	Points to a <b>sockaddr</b> structure containing the destination address. The length 65021 and format of the address depend on the address family of the socket.
65022	<i>dest_len</i>	Specifies the length of the <b>sockaddr</b> structure pointed to by the <i>dest_addr</i> 65023 argument.

65024 If the address family of the socket is AF\_UNIX, the application shall ensure that a null  
65025 terminator after the pathname is included in the *sun\_path* member of *dest\_addr* as a **sockaddr\_un**  
65026 structure, and that *dest\_len* is at least `offsetof(struct sockaddr_un, sun_path) + 1`  
65027 plus the length of the pathname.

65028 If the socket protocol supports broadcast and the specified address is a broadcast address for the  
65029 socket protocol, *sendto()* shall fail if the SO\_BROADCAST option is not set for the socket.

65030 The *dest\_addr* argument specifies the address of the target.65031 The *length* argument specifies the length of the message.65032 Successful completion of a call to *sendto()* does not guarantee delivery of the message. A return

- 65033 value of `-1` indicates only locally-detected errors.
- 65034 If space is not available at the sending socket to hold the message to be transmitted and the  
65035 socket file descriptor does not have `O_NONBLOCK` set, `sendto()` shall block until space is  
65036 available or a timeout occurs (see `SO_SNDTIMEO` in [Section 2.10.16](#), on page 554). If space is  
65037 not available at the sending socket to hold the message to be transmitted and the socket file  
65038 descriptor does have `O_NONBLOCK` set, `sendto()` shall fail.
- 65039 The socket in use may require the process to have appropriate privileges to use the `sendto()`  
65040 function.
- 65041 **RETURN VALUE**
- 65042 Upon successful completion, `sendto()` shall return the number of bytes sent. Otherwise, `-1` shall  
65043 be returned and `errno` set to indicate the error.
- 65044 **ERRORS**
- 65045 The `sendto()` function shall fail if:
- 65046 [EAFNOSUPPORT]  
65047 Addresses in the specified address family cannot be used with this socket.
- 65048 [EAGAIN] or [EWOULDBLOCK]  
65049 The socket's file descriptor is marked `O_NONBLOCK` and the requested  
65050 operation would block. See also `SO_SNDTIMEO` in [Section 2.10.16](#) (on page  
65051 554).
- 65052 [EBADF] The *socket* argument is not a valid file descriptor.
- 65053 [ECONNRESET] A connection was forcibly closed by a peer.
- 65054 [EINTR] A signal interrupted `sendto()` before any data was transmitted.
- 65055 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 65056 [ENOTCONN] The socket is connection-mode but is not connected.
- 65057 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 65058 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or  
65059 more of the values set in *flags*.
- 65060 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is  
65061 no longer connected. In the latter case, and if the socket is of type  
65062 `SOCK_STREAM` or `SOCK_SEQPACKET` and the `MSG_NOSIGNAL` flag is not  
65063 set, the `SIGPIPE` signal is generated to the calling thread.
- 65064 If the address family of the socket is `AF_UNIX`, then `sendto()` shall fail if:
- 65065 [EIO] An I/O error occurred while reading from or writing to the file system.
- 65066 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname  
65067 in the socket address.
- 65068 [ENAMETOOLONG]  
65069 The length of a component of a pathname is longer than `{NAME_MAX}`.
- 65070 [ENOENT] A component of the pathname does not name an existing file or the pathname  
65071 is an empty string.
- 65072 [ENOTDIR] A component of the path prefix of the pathname in the socket address names  
65073 an existing file that is neither a directory nor a symbolic link to a directory, or  
65074 the pathname in the socket address contains at least one non-`<slash>` character

65075 and ends with one or more trailing <slash> characters and the last pathname  
65076 component names an existing file that is neither a directory nor a symbolic  
65077 link to a directory.

65078 The *sendto()* function may fail if:

65079 [EACCES] Search permission is denied for a component of the path prefix; or write access  
65080 to the named socket is denied.

65081 [EDESTADDRREQ]  
65082 The socket is not connection-mode and does not have its peer address set, and  
65083 no destination address was specified.

65084 [EHOSTUNREACH]  
65085 The destination host cannot be reached (probably because the host is down or  
65086 a remote router cannot reach it).

65087 [EINVAL] The *dest\_len* argument is not a valid length for the address family.

65088 [EIO] An I/O error occurred while reading from or writing to the file system.

65089 [EISCONN] A destination address was specified and the socket is already connected.

65090 [ENETDOWN] The local network interface used to reach the destination is down.

65091 [ENETUNREACH]  
65092 No route to the network is present.

65093 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

65094 [ENOMEM] Insufficient memory was available to fulfill the request.

65095 If the address family of the socket is AF\_UNIX, then *sendto()* may fail if:

65096 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
65097 resolution of the pathname in the socket address.

65098 [ENAMETOOLONG]  
65099 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
65100 symbolic link produced an intermediate result with a length that exceeds  
65101 {PATH\_MAX}.

#### 65102 EXAMPLES

65103 None.

#### 65104 APPLICATION USAGE

65105 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.

65106 For AF\_UNIX sockets, some implementations support an extension where *dest\_len* does not have  
65107 to include a null terminator for the pathname stored in *sun\_path*, which in turn allows a  
65108 pathname to be one byte longer. However, such usage is not portable, and carries a risk of  
65109 accessing beyond the intended bounds of the pathname length.

#### 65110 RATIONALE

65111 None.

#### 65112 FUTURE DIRECTIONS

65113 None.

65114 **SEE ALSO**

65115 [getsockopt\(\)](#), [poll\(\)](#), [pselect\(\)](#), [recv\(\)](#), [recvfrom\(\)](#), [recvmsg\(\)](#), [send\(\)](#), [sendmsg\(\)](#), [setsockopt\(\)](#),  
65116 [shutdown\(\)](#), [socket\(\)](#)

65117 XBD <[sys/socket.h](#)>

65118 **CHANGE HISTORY**

65119 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

65120 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
65121 [ELOOP] error condition is added.

65122 **Issue 7**

65123 Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, updating the [EISCONN]  
65124 error and the DESCRIPTION.

65125 Austin Group Interpretation 1003.1-2001 #143 is applied, clarifying the [ENAMETOOLONG]  
65126 error condition.

65127 The MSG\_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended  
65128 API Set Part 2.

65129 The [EPIPE] error is modified.

65130 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0545 [324] is applied.

65131 **Issue 8**

65132 Austin Group Defect 561 is applied, changing the requirements for the *sun\_path* member of the  
65133 **sockaddr\_un** structure.

65134 Austin Group Defect 1429 is applied, clarifying the behavior on timeout by adding references to  
65135 [Section 2.10.16](#) (on page 554).

65136 **NAME**

65137 setbuf — assign buffering to a stream

65138 **SYNOPSIS**

65139 #include &lt;stdio.h&gt;

65140 void setbuf(FILE \*restrict stream, char \*restrict buf);

65141 **DESCRIPTION**

65142 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
65143 conflict between the requirements described here and the ISO C standard is unintentional. This  
65144 volume of POSIX.1-2024 defers to the ISO C standard.

65145 Except that it returns no value, the function call:

65146 setbuf(stream, buf)

65147 shall be equivalent to:

65148 setvbuf(stream, buf, \_IOFBF, BUFSIZ)

65149 if *buf* is not a null pointer, or to:

65150 setvbuf(stream, buf, \_IONBF, BUFSIZ)

65151 if *buf* is a null pointer.65152 **RETURN VALUE**65153 The *setbuf()* function shall not return a value.65154 **ERRORS**

65155 Although the *setvbuf()* interface may set *errno* in defined ways, the value of *errno* after a call to  
65156 *setbuf()* is unspecified.

65157 **EXAMPLES**

65158 None.

65159 **APPLICATION USAGE**

65160 A common source of error is allocating buffer space as an “automatic” variable in a code block,  
65161 and then failing to close the stream in the same block.

65162 With *setbuf()*, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ  
65163 bytes are used for the buffer area.

65164 Since *errno* is not required to be unchanged on success, in order to correctly detect and possibly  
65165 recover from errors, applications should use *setvbuf()* instead of *setbuf()*.

65166 **RATIONALE**

65167 None.

65168 **FUTURE DIRECTIONS**

65169 None.

65170 **SEE ALSO**65171 [Section 2.5](#) (on page 521), *fopen()*, *setvbuf()*65172 XBD [<stdio.h>](#)65173 **CHANGE HISTORY**

65174 First released in Issue 1. Derived from Issue 1 of the SVID.



65175 **Issue 6**

65176 The prototype for *setbuf()* is updated for alignment with the ISO/IEC 9899:1999 standard.

65177 **Issue 7**

65178 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0546 [397], XSH/TC1-2008/0547 [397],  
65179 and XSH/TC1-2008/0548 [14] are applied.

65180 **NAME**

65181 setegid — set the effective group ID

65182 **SYNOPSIS**

65183 #include &lt;unistd.h&gt;

65184 int setegid(gid\_t gid);

65185 **DESCRIPTION**

65186 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate  
65187 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group  
65188 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

65189 The *setegid()* function shall not affect the supplementary group list in any way.

65190 **RETURN VALUE**

65191 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
65192 indicate the error.

65193 **ERRORS**

65194 The *setegid()* function shall fail if:

65195 [EINVAL] The value of the *gid* argument is invalid and is not supported by the  
65196 implementation.

65197 [EPERM] The process does not have appropriate privileges and *gid* does not match the  
65198 real group ID or the saved set-group-ID.

65199 **EXAMPLES**

65200 None.

65201 **APPLICATION USAGE**

65202 None.

65203 **RATIONALE**

65204 Refer to the RATIONALE section in *setuid()*.

65205 **FUTURE DIRECTIONS**

65206 None.

65207 **SEE ALSO**

65208 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*,  
65209 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

65210 XBD <sys/types.h>, <unistd.h>

65211 **CHANGE HISTORY**

65212 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

65213 **Issue 8**

65214 Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to  
65215 SEE ALSO.

65216 **NAME**

65217 setenv — add or change environment variable

65218 **SYNOPSIS**

```
65219 CX #include <stdlib.h>
65220 int setenv(const char *envname, const char *envval, int overwrite);
```

65221 **DESCRIPTION**

65222 The *setenv()* function shall update or add a variable in the environment of the calling process.  
 65223 The *envname* argument points to a string containing the name of an environment variable to be  
 65224 added or altered. The environment variable shall be set to the value to which *envval* points. The  
 65225 function shall fail if *envname* points to a string which contains an '=' character. If the  
 65226 environment variable named by *envname* already exists and the value of *overwrite* is non-zero,  
 65227 the function shall return success and the environment shall be updated. If the environment  
 65228 variable named by *envname* already exists and the value of *overwrite* is zero, the function shall  
 65229 return success and the environment shall remain unchanged.

65230 The *setenv()* function shall update the list of pointers to which *environ* points.

65231 The strings described by *envname* and *envval* are copied by this function.

65232 The *setenv()* function need not be thread-safe.

65233 **RETURN VALUE**

65234 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
 65235 indicate the error, and the environment shall be unchanged.

65236 **ERRORS**

65237 The *setenv()* function shall fail if:

65238 [EINVAL] The *envname* argument points to an empty string or points to a string  
 65239 containing an '=' character.

65240 [ENOMEM] Insufficient memory was available to add a variable or its value to the  
 65241 environment.

65242 **EXAMPLES**

65243 None.

65244 **APPLICATION USAGE**

65245 See *exec()* for restrictions on changing the environment in multi-threaded applications.

65246 **RATIONALE**

65247 Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular, if  
 65248 the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an  
 65249 obsolete copy of the environment (as may any other copy of *environ*). However, other than the  
 65250 aforementioned restriction, the standard developers intended that the traditional method of  
 65251 walking through the environment by way of the *environ* pointer must be supported.

65252 It was decided that *setenv()* should be required by this version because it addresses a piece of  
 65253 missing functionality, and does not impose a significant burden on the implementor.

65254 There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()*  
 65255 function should be required as a mandatory function. The *setenv()* function was chosen because  
 65256 it permitted the implementation of the *unsetenv()* function to delete environmental variables,  
 65257 without specifying an additional interface. The *putenv()* function is available as part of the XSI  
 65258 option.

65259 The standard developers considered requiring that *setenv()* indicate an error when a call to it  
65260 would result in exceeding {ARG\_MAX}. The requirement was rejected since the condition might  
65261 be temporary, with the application eventually reducing the environment size. The ultimate  
65262 success or failure depends on the size at the time of a call to *exec*, which returns an indication of  
65263 this error condition.

65264 See also the RATIONALE section in *getenv()*.

#### 65265 FUTURE DIRECTIONS

65266 None.

#### 65267 SEE ALSO

65268 *exec*, *getenv()*, *putenv()*, *unsetenv()*

65269 XBD <stdlib.h>, <sys/types.h>, <unistd.h>

#### 65270 CHANGE HISTORY

65271 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

65272 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in  
65273 the APPLICATION USAGE and SEE ALSO sections.

#### 65274 Issue 7

65275 Austin Group Interpretation 1003.1-2001 #156 is applied.

65276 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0549 [167], XSH/TC1-2008/0550 [185],  
65277 XSH/TC1-2008/0551 [167], and XSH/TC1-2008/0552 [38] are applied.

65278 **NAME**

65279            seteuid — set effective user ID

65280 **SYNOPSIS**

65281            #include &lt;unistd.h&gt;

65282            int seteuid(uid\_t uid);

65283 **DESCRIPTION**

65284            If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate  
 65285 privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID  
 65286 and saved set-user-ID shall remain unchanged.

65287            The *seteuid()* function shall not affect the supplementary group list in any way.

65288 **RETURN VALUE**

65289            Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to  
 65290 indicate the error.

65291 **ERRORS**

65292            The *seteuid()* function shall fail if:

65293            [EINVAL]            The value of the *uid* argument is invalid and is not supported by the  
 65294 implementation.

65295            [EPERM]            The process does not have appropriate privileges and *uid* does not match the  
 65296 real user ID or the saved set-user-ID.

65297 **EXAMPLES**

65298            None.

65299 **APPLICATION USAGE**

65300            None.

65301 **RATIONALE**65302            Refer to the RATIONALE section in *setuid()*.65303 **FUTURE DIRECTIONS**

65304            None.

65305 **SEE ALSO**

65306            *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*,  
 65307 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

65308            XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

65309 **CHANGE HISTORY**

65310            First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

65311            IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/123 is applied, making an editorial  
 65312 correction to the [EPERM] error in the ERRORS section.

65313 **Issue 8**

65314            Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to  
 65315 SEE ALSO.

65316 **NAME**

65317 setgid — set-group-ID

65318 **SYNOPSIS**

65319 #include &lt;unistd.h&gt;

65320 int setgid(gid\_t gid);

65321 **DESCRIPTION**65322 If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID,  
65323 and the saved set-group-ID of the calling process to *gid*.65324 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the  
65325 saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved  
65326 set-group-ID shall remain unchanged.65327 The *setgid()* function shall not affect the supplementary group list in any way.

65328 Any supplementary group IDs of the calling process shall remain unchanged.

65329 **RETURN VALUE**65330 Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to  
65331 indicate the error.65332 **ERRORS**65333 The *setgid()* function shall fail if:65334 [EINVAL] The value of the *gid* argument is invalid and is not supported by the  
65335 implementation.65336 [EPERM] The process does not have appropriate privileges and *gid* does not match the  
65337 real group ID or the saved set-group-ID.65338 **EXAMPLES**

65339 None.

65340 **APPLICATION USAGE**

65341 None.

65342 **RATIONALE**65343 Refer to the RATIONALE section in *setuid()*.65344 **FUTURE DIRECTIONS**

65345 None.

65346 **SEE ALSO**65347 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*,  
65348 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

65349 XBD &lt;sys/types.h&gt;, &lt;unistd.h&gt;

65350 **CHANGE HISTORY**

65351 First released in Issue 1. Derived from Issue 1 of the SVID.

65352 **Issue 6**

65353 In the SYNOPSIS, the optional include of the &lt;sys/types.h&gt; header is removed.

65354 The following new requirements on POSIX implementations derive from alignment with the  
65355 Single UNIX Specification:

65356 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
65357 required for conforming implementations of previous POSIX specifications, it was not  
65358 required for UNIX applications.

65359 • Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS  
65360 requirement.

65361 The following changes were made to align with the IEEE P1003.1a draft standard:

65362 • The effects of `setgid()` in processes without appropriate privileges are changed.

65363 • A requirement that the supplementary group list is not affected is added.

65364 **Issue 8**

65365 Austin Group Defect 1344 is applied, adding `getresgid()`, `getresuid()`, `setresgid()`, and `setresuid()` to  
65366 SEE ALSO.

65367 **NAME**

65368           setgrent — reset the group database to the first entry

65369 **SYNOPSIS**

```
65370 XSI       #include <grp.h>  
65371       void setgrent(void);
```

65372 **DESCRIPTION**

65373       Refer to *endgrent()*.



65374 **NAME**

65375 sethostent — network host database functions

65376 **SYNOPSIS**

65377 #include &lt;netdb.h&gt;

65378 void sethostent(int *stayopen*);65379 **DESCRIPTION**65380 Refer to *endhostent()*.

65381 **NAME**

65382 setjmp — set jump point for a non-local goto

65383 **SYNOPSIS**

65384 #include &lt;setjmp.h&gt;

65385 int setjmp(jmp\_buf env);

65386 **DESCRIPTION**

65387 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 65388 conflict between the requirements described here and the ISO C standard is unintentional. This  
 65389 volume of POSIX.1-2024 defers to the ISO C standard.

65390 A call to *setjmp()* shall save the calling environment in its *env* argument for later use by  
 65391 *longjmp()*.

65392 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in  
 65393 order to access an actual function, or a program defines an external identifier with the name  
 65394 *setjmp*, the behavior is undefined.

65395 An application shall ensure that an invocation of *setjmp()* appears in one of the following  
 65396 contexts only:

- 65397 • The entire controlling expression of a selection or iteration statement
- 65398 • One operand of a relational or equality operator with the other operand an integral  
 65399 constant expression, with the resulting expression being the entire controlling expression  
 65400 of a selection or iteration statement
- 65401 • The operand of a unary '!' operator with the resulting expression being the entire  
 65402 controlling expression of a selection or iteration
- 65403 • The entire expression of an expression statement (possibly cast to **void**)

65404 If the invocation appears in any other context, the behavior is undefined.

65405 **RETURN VALUE**

65406 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to  
 65407 *longjmp()*, *setjmp()* shall return a non-zero value.

65408 **ERRORS**

65409 No errors are defined.

65410 **EXAMPLES**

65411 None.

65412 **APPLICATION USAGE**

65413 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-  
 65414 level subroutine of a program.

65415 **RATIONALE**

65416 None.

65417 **FUTURE DIRECTIONS**

65418 None.

65419 **SEE ALSO**

65420 [longjmp\(\)](#), [sigsetjmp\(\)](#)

65421 XBD [<setjmp.h>](#)

65422 **CHANGE HISTORY**

65423 First released in Issue 1. Derived from Issue 1 of the SVID.

65424 **Issue 6**

65425 The normative text is updated to avoid use of the term “must” for application requirements.

65426 **NAME**65427 setkey — set encoding key (**CRYPT**)65428 **SYNOPSIS**

```
65429 OB XSI #include <stdlib.h>  
65430 void setkey(const char *key);
```

65431 **DESCRIPTION**

65432 The *setkey()* function provides access to an implementation-defined encoding algorithm. The  
65433 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical  
65434 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is  
65435 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used  
65436 with the algorithm to encode a string *block* passed to *encrypt()*.

65437 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to  
65438 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on  
65439 return, an error has occurred.

65440 The *setkey()* function need not be thread-safe.

65441 **RETURN VALUE**

65442 No values are returned.

65443 **ERRORS**

65444 The *setkey()* function shall fail if:

65445 [ENOSYS] The functionality is not supported on this implementation.

65446 **EXAMPLES**

65447 None.

65448 **APPLICATION USAGE**

65449 Decoding need not be implemented in all environments. This is related to government  
65450 restrictions in some countries on encryption and decryption routines. Historical practice has  
65451 been to ship a different version of the encryption library without the decryption feature in the  
65452 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

65453 **RATIONALE**

65454 None.

65455 **FUTURE DIRECTIONS**

65456 The *setkey()* function may be removed in a future version.

65457 **SEE ALSO**

65458 [crypt\(\)](#), [encrypt\(\)](#)

65459 XBD [<stdlib.h>](#)

65460 **CHANGE HISTORY**

65461 First released in Issue 1. Derived from Issue 1 of the SVID.

65462 **Issue 5**

65463 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

65464 **Issue 7**

65465 Austin Group Interpretation 1003.1-2001 #156 is applied.

65466 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0320 [899] is applied.

65467 **Issue 8**  
65468

Austin Group Defect 1192 is applied, marking the *setkey()* function as obsolescent.

65469 **NAME**

65470 setlocale — set program locale

65471 **SYNOPSIS**

65472 #include &lt;locale.h&gt;

65473 char \*setlocale(int category, const char \*locale);

65474 **DESCRIPTION**

65475 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 65476 conflict between the requirements described here and the ISO C standard is unintentional. This  
 65477 volume of POSIX.1-2024 defers to the ISO C standard.

65478 The *setlocale()* function selects the appropriate piece of the global locale, as specified by the  
 65479 *category* and *locale* arguments, and can be used to change or query the entire global locale or  
 65480 portions thereof. The value LC\_ALL for *category* names the entire global locale; other values for  
 65481 *category* name only a part of the global locale:

65482 LC\_COLLATE Affects the behavior of regular expressions and the collation functions.

65483 LC\_CTYPE Affects the behavior of regular expressions, character classification, character  
 65484 conversion functions, and wide-character functions.

65485 CX LC\_MESSAGES Affects the affirmative and negative response expressions returned by  
 65486 *nl\_langinfo()* and the way message catalogs are located. It may also affect the  
 65487 behavior of functions that return or write message strings.

65488 LC\_MONETARY Affects the behavior of functions that handle monetary values.

65489 LC\_NUMERIC Affects the behavior of functions that handle numeric values.

65490 LC\_TIME Affects the behavior of the time conversion functions.

65491 The *locale* argument is a pointer to a character string containing the required setting of *category*.  
 65492 The contents of this string are implementation-defined. In addition, the following preset values  
 65493 of *locale* are defined for all settings of *category*:

65494 CX "POSIX" Specifies the minimal environment for C-language translation called the  
 65495 POSIX locale. The POSIX locale is the default global locale at entry to *main()*.

65496 "C" Equivalent to "POSIX".

65497 CX "" Specifies an implementation-defined native environment. The determination  
 65498 of the name of the new locale for the specified category depends on the value  
 65499 of the associated environment variables, *LC\_\** and *LANG*; see XBD Chapter 7  
 65500 (on page 127) and Chapter 8 (on page 167).

65501 A null pointer Directs *setlocale()* to query the current global locale setting and return the  
 65502 name of the locale if *category* is not LC\_ALL, or a string which encodes the  
 65503 locale name(s) for all of the individual categories if *category* is LC\_ALL.

65504 CX Setting all of the categories of the global locale is similar to successively setting each individual  
 65505 category of the global locale, except that all error checking is done before any actions are  
 65506 performed. To set all the categories of the global locale, *setlocale()* can be invoked as:

```
65507 setlocale(LC_ALL, "");
```

65508 In this case, *setlocale()* shall first verify that the values of all the environment variables it needs  
 65509 according to the precedence rules (described in XBD Chapter 8, on page 167) indicate supported  
 65510 locales. If the value of any of these environment variable searches yields a locale that is not  
 65511 supported (and non-null), *setlocale()* shall return a null pointer and the global locale shall not be

65512 changed. If all environment variables name supported locales, *setlocale()* shall proceed as if it  
 65513 had been called for each category, using the appropriate value from the associated environment  
 65514 variable or from the implementation-defined default if there is no such value.

65515 The global locale established using *setlocale()* shall only be used in threads for which no current  
 65516 locale has been set using *uselocale()* or whose current locale has been set to the global locale  
 65517 using *uselocale(LC\_GLOBAL\_LOCALE)*.

65518 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 65519 *setlocale()*.

65520 The *setlocale()* function need not be thread-safe; however, it shall avoid data races with all  
 65521 function calls that do not affect and are not affected by the global locale.

#### 65522 RETURN VALUE

65523 Upon successful completion, *setlocale()* shall return the string associated with the specified  
 65524 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the global locale  
 65525 shall not be changed.

65526 A null pointer for *locale* shall cause *setlocale()* to return a pointer to the string associated with the  
 65527 specified *category* for the current global locale. The global locale shall not be changed.

65528 The string returned by *setlocale()* is such that a subsequent call with that string and its associated  
 65529 *category* shall restore that part of the global locale. The application shall not modify the string  
 65530 CX returned. The returned string pointer might be invalidated or the string content might be  
 65531 CX overwritten by a subsequent call to *setlocale()*. The returned pointer might also be invalidated if  
 65532 the calling thread is terminated.

#### 65533 ERRORS

65534 No errors are defined.

#### 65535 EXAMPLES

65536 None.

#### 65537 APPLICATION USAGE

65538 The following code illustrates how a program can initialize the international environment for  
 65539 one language, while selectively modifying the global locale such that regular expressions and  
 65540 string operations can be applied to text recorded in a different language:

```
65541 setlocale(LC_ALL, "De");  
65542 setlocale(LC_COLLATE, "Fr@dict");
```

65543 Internationalized programs can initiate language operation according to environment variable  
 65544 settings (see XBD Section 8.2, on page 169) by calling *setlocale()* as follows:

```
65545 setlocale(LC_ALL, "");
```

65546 Changing the setting of *LC\_MESSAGES* has no effect on catalogs that have already been opened  
 65547 by calls to *catopen()*.

65548 In order to make use of different locale settings while multiple threads are running, applications  
 65549 should use *uselocale()* in preference to *setlocale()*.

#### 65550 RATIONALE

65551 References to the international environment or locale in the following text relate to the global  
 65552 locale for the process. This can be overridden for individual threads using *uselocale()*.

65553 The ISO C standard defines a collection of functions to support internationalization. One of the  
 65554 most significant aspects of these functions is a facility to set and query the *international*  
 65555 *environment*. The international environment is a repository of information that affects the

65556 behavior of certain functionality, namely:

- 65557 1. Character handling
- 65558 2. Collating
- 65559 3. Date/time formatting
- 65560 4. Numeric editing
- 65561 5. Monetary formatting
- 65562 6. Messaging

65563 The `setlocale()` function provides the application developer with the ability to set all or portions,  
65564 called *categories*, of the international environment. These categories correspond to the areas of  
65565 functionality mentioned above. The syntax for `setlocale()` is as follows:

```
65566 char *setlocale(int category, const char *locale);
```

65567 where *category* is the name of one of following categories, namely:

```
65568     LC_COLLATE
65569     LC_CTYPE
65570     LC_MESSAGES
65571     LC_MONETARY
65572     LC_NUMERIC
65573     LC_TIME
```

65574 In addition, a special value called `LC_ALL` directs `setlocale()` to set all categories.

65575 There are two primary uses of `setlocale()`:

- 65576 1. Querying the international environment to find out what it is set to
- 65577 2. Setting the international environment, or *locale*, to a specific value

65578 The behavior of `setlocale()` in these two areas is described below. Since it is difficult to describe  
65579 the behavior in words, examples are used to illustrate the behavior of specific uses.

65580 To query the international environment, `setlocale()` is invoked with a specific category and the  
65581 null pointer as the locale. The null pointer is a special directive to `setlocale()` that tells it to query  
65582 rather than set the international environment. The following syntax is used to query the name of  
65583 the international environment:

```
65584 setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
65585          LC_NUMERIC, LC_TIME}, (char *) NULL);
```

65586 The `setlocale()` function shall return the string corresponding to the current international  
65587 environment. This value may be used by a subsequent call to `setlocale()` to reset the international  
65588 environment to this value. However, it should be noted that the return value from `setlocale()`  
65589 may be a pointer to a static area within the function and is not guaranteed to remain unchanged  
65590 (that is, it may be modified by a subsequent call to `setlocale()`). Therefore, if the purpose of  
65591 calling `setlocale()` is to save the value of the current international environment so it can be  
65592 changed and reset later, the return value should be copied to an array of `char` in the calling  
65593 program.

65594 There are three ways to set the international environment with `setlocale()`:



65595 *setlocale(category, string)*

65596 This usage sets a specific *category* in the international environment to a specific value  
65597 corresponding to the value of the *string*. A specific example is provided below:

```
65598 setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

65599 In this example, all categories of the international environment are set to the locale  
65600 corresponding to the string "fr\_FR.ISO-8859-1", or to the French language as spoken in  
65601 France using the ISO/IEC 8859-1:1998 standard codeset.

65602 If the string does not correspond to a valid locale, *setlocale()* shall return a null pointer and  
65603 the international environment is not changed. Otherwise, *setlocale()* shall return the name of  
65604 the locale just set.

65605 *setlocale(category, "C")*

65606 The ISO C standard states that one locale must exist on all conforming implementations.  
65607 The name of the locale is C and corresponds to a minimal international environment needed  
65608 to support the C programming language.

65609 *setlocale(category, "")*

65610 This sets a specific category to an implementation-defined default. This corresponds to the  
65611 value of the environment variables.

#### 65612 FUTURE DIRECTIONS

65613 None.

#### 65614 SEE ALSO

65615 *catopen()*, *exec*, *fprintf()*, *fscanf()*, *getlocalename\_l()*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*,  
65616 *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *iswalnum()*, *iswalpha()*,  
65617 *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,  
65618 *iswspace()*, *iswupper()*, *iswxdigit()*, *isxdigit()*, *localeconv()*, *mblen()*, *mbstowcs()*, *mbtowc()*,  
65619 *newlocale()*, *nl\_langinfo()*, *perror()*, *psiginfo()*, *strcoll()*, *strerror()*, *strfmon()*, *strsignal()*, *strtod()*,  
65620 *strxfrm()*, *tolower()*, *toupper()*, *towlower()*, *towupper()*, *uselocale()*, *wscoll()*, *wcstod()*, *wcstombs()*,  
65621 *wcsxfrm()*, *wctomb()*

65622 XBD Chapter 7 (on page 127), Chapter 8 (on page 167), [<langinfo.h>](#), [<locale.h>](#)

#### 65623 CHANGE HISTORY

65624 First released in Issue 3.

#### 65625 Issue 5

65626 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

#### 65627 Issue 6

65628 Extensions beyond the ISO C standard are marked.

65629 The normative text is updated to avoid use of the term ``must'' for application requirements.

65630 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/124 is applied, updating the  
65631 DESCRIPTION to clarify the behavior of:

```
65632 setlocale(LC_ALL, "");
```

#### 65633 Issue 7

65634 Functionality relating to the Threads option is moved to the Base.

65635 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0553 [302], XSH/TC1-2008/0554 [303],  
65636 XSH/TC1-2008/0555 [302], XSH/TC1-2008/0556 [302], XSH/TC1-2008/0557 [302],  
65637 XSH/TC1-2008/0558 [302], XSH/TC1-2008/0559 [302], XSH/TC1-2008/0560 [288],  
65638 XSH/TC1-2008/0561 [302], XSH/TC1-2008/0562 [302], XSH/TC1-2008/0563 [302],

65639 XSH/TC1-2008/0564 [302], XSH/TC1-2008/0565 [302], XSH/TC1-2008/0566 [302],  
65640 XSH/TC1-2008/0567 [288], XSH/TC1-2008/0568 [288], and XSH/TC1-2008/0569 [303] are  
65641 applied.

65642 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0321 [826], XSH/TC2-2008/0322 [826],  
65643 and XSH/TC2-2008/0323 [596] are applied.

65644 **Issue 8**

65645 Austin Group Defect 1220 is applied, adding *getlocalename\_l()* to the SEE ALSO section.

65646 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
65647 standard.

65648 **NAME**

65649 setlogmask — set the log priority mask

65650 **SYNOPSIS**65651 XSI `#include <syslog.h>`65652 `int setlogmask(int maskpri);`65653 **DESCRIPTION**65654 Refer to *closelog()*.

65655 **NAME**  
65656           setnetent — network database function

65657 **SYNOPSIS**  
65658           #include <netdb.h>  
65659           void setnetent(int *stayopen*);

65660 **DESCRIPTION**  
65661           Refer to *endnetent()*.

65662 **NAME**

65663 setpgid — set process group ID for job control

65664 **SYNOPSIS**

65665 #include &lt;unistd.h&gt;

65666 int setpgid(pid\_t pid, pid\_t pgid);

65667 **DESCRIPTION**65668 The *setpgid()* function shall either join an existing process group or create a new process group  
65669 within the session of the calling process.

65670 The process group ID of a session leader shall not change.

65671 Upon successful completion, the process group ID of the process with a process ID that matches  
65672 *pid* shall be set to *pgid*.65673 As a special case, if *pid* is 0, the process ID of the calling process shall be used. Also, if *pgid* is 0,  
65674 the process ID of the indicated process shall be used.65675 **RETURN VALUE**65676 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*  
65677 shall be set to indicate the error.65678 **ERRORS**65679 The *setpgid()* function shall fail if:65680 [EACCES] The value of the *pid* argument matches the process ID of a child process of the  
65681 calling process and the child process has successfully executed one of the *exec*  
65682 functions.65683 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by  
65684 the implementation.65685 [EPERM] The process indicated by the *pid* argument is a session leader.65686 [EPERM] The value of the *pid* argument matches the process ID of a child process of the  
65687 calling process and the child process is not in the same session as the calling  
65688 process.65689 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of  
65690 the process indicated by the *pid* argument and there is no process with a  
65691 process group ID that matches the value of the *pgid* argument in the same  
65692 session as the calling process.65693 [ESRCH] The value of the *pid* argument does not match the process ID of the calling  
65694 process or of a child process of the calling process.65695 **EXAMPLES**

65696 None.

65697 **APPLICATION USAGE**

65698 None.

65699 **RATIONALE**65700 The *setpgid()* function shall group processes together for the purpose of signaling, placement in  
65701 foreground or background, and other job control actions.65702 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed  
65703 the specified new process group to assume any value. This presents certain security problems  
65704 and is more flexible than necessary to support job control.

65705 To provide tighter security, *setpgid()* only allows the calling process to join a process group  
65706 already in use inside its session or create a new process group whose process group ID was  
65707 equal to its process ID.

65708 When a job control shell spawns a new job, the processes in the job must be placed into a new  
65709 process group via *setpgid()*. There are two timing constraints involved in this action:

- 65710 1. The new process must be placed in the new process group before the appropriate  
65711 program is launched via one of the *exec* functions.
- 65712 2. The new process must be placed in the new process group before the shell can correctly  
65713 send signals to the new process group.

65714 To address these constraints, the following actions are performed. The new processes call  
65715 *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first  
65716 constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of  
65717 *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that  
65718 the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization  
65719 property was considered, but it was decided instead to merely allow the parent shell process to  
65720 adjust the process group of its child processes via *setpgid()*. Both timing constraints are now  
65721 satisfied by having both the parent shell and the child attempt to adjust the process group of the  
65722 child process; it does not matter which succeeds first.

65723 Since it would be confusing to an application to have its process group change after it began  
65724 executing (that is, after *exec*), and because the child process would already have adjusted its  
65725 process group before this, the [EACCES] error was added to disallow this.

65726 One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original  
65727 process group (the one in effect when the job control shell was executed). A job control shell  
65728 does this before returning control back to its parent when it is terminating or suspending itself as  
65729 a way of restoring its job control “state” back to what its parent would expect. (Note that the  
65730 original process group of the job control shell typically matches the process group of its parent,  
65731 but this is not necessarily always the case.)

#### 65732 FUTURE DIRECTIONS

65733 None.

#### 65734 SEE ALSO

65735 *exec*, *getpgrp()*, *setsid()*, *tcsetpgrp()*

65736 XBD [<sys/types.h>](#), [<unistd.h>](#)

#### 65737 CHANGE HISTORY

65738 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 65739 Issue 6

65740 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

65741 The following new requirements on POSIX implementations derive from alignment with the  
65742 Single UNIX Specification:

- 65743 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
65744 required for conforming implementations of previous POSIX specifications, it was not  
65745 required for UNIX applications.
- 65746 • The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be  
65747 defined in this version. This is a FIPS requirement.

65748  
65749  
65750  
65751

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in the DESCRIPTION from “the process group ID of the indicated process shall be used” to “the process ID of the indicated process shall be used”. This change reverts the wording to as in the ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

65752 **NAME**

65753           setpriority — set the nice value

65754 **SYNOPSIS**

```
65755 XSI       #include <sys/resource.h>  
65756       int setpriority(int which, id_t who, int nice);
```

65757 **DESCRIPTION**

65758       Refer to *getpriority()*.



65759 **NAME**  
65760 setprotoent — network protocol database functions

65761 **SYNOPSIS**  
65762 #include <netdb.h>  
65763 void setprotoent(int stayopen);

65764 **DESCRIPTION**  
65765 Refer to *endprotoent()*.

65766 **NAME**

65767           setpwent — user database function

65768 **SYNOPSIS**

```
65769 XSI       #include <pwd.h>
```

```
65770       void setpwent(void);
```

65771 **DESCRIPTION**

65772       Refer to *endpwent()*.

65773 **NAME**

65774 setregid — set real and effective group IDs

65775 **SYNOPSIS**

```
65776 XSI #include <unistd.h>
65777 int setregid(gid_t rgid, gid_t egid);
```

65778 **DESCRIPTION**65779 The *setregid()* function shall set the real and effective group IDs of the calling process.65780 If *rgid* is  $-1$ , the real group ID shall not be changed; if *egid* is  $-1$ , the effective group ID shall not  
65781 be changed.

65782 The real and effective group IDs may be set to different values in the same call.

65783 Only a process with appropriate privileges can set the real group ID and the effective group ID  
65784 to any valid value.65785 A non-privileged process can set either the real group ID to the saved set-group-ID from one of  
65786 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group  
65787 ID.65788 If the real group ID is being set (*rgid* is not  $-1$ ), or the effective group ID is being set to a value  
65789 not equal to the real group ID, then the saved set-group-ID of the current process shall be set  
65790 equal to the new effective group ID.

65791 Any supplementary group IDs of the calling process remain unchanged.

65792 **RETURN VALUE**65793 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
65794 indicate the error, and neither of the group IDs are changed.65795 **ERRORS**65796 The *setregid()* function shall fail if:65797 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.65798 [EPERM] The process does not have appropriate privileges and a change other than  
65799 changing the real group ID to the saved set-group-ID, or changing the  
65800 effective group ID to the real group ID or the saved set-group-ID, was  
65801 requested.65802 **EXAMPLES**

65803 None.

65804 **APPLICATION USAGE**65805 If a non-privileged set-group-ID process sets its effective group ID to its real group ID, it can  
65806 only set its effective group ID back to the previous value if *rgid* was  $-1$  in the *setregid()* call, since  
65807 the saved-group-ID is not changed in that case. If *rgid* was equal to the real group ID in the  
65808 *setregid()* call, then the saved set-group-ID will also have been changed to the real user ID.65809 **RATIONALE**65810 Earlier versions of this standard did not specify whether the saved set-group-ID was affected by  
65811 *setregid()* calls. This version specifies common existing practice that constitutes an important  
65812 security feature. The ability to set both the effective group ID and saved set-group-ID to be the  
65813 same as the real group ID means that any security weakness in code that is executed after that  
65814 point cannot result in malicious code being executed with the previous effective group ID.  
65815 Privileged applications could already do this using just *setgid()*, but for non-privileged

65816 applications the only standard method available is to use this feature of *setregid()*.

65817 **FUTURE DIRECTIONS**

65818 None.

65819 **SEE ALSO**

65820 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*,  
65821 *setresgid()*, *setresuid()*, *setreuid()*, *setuid()*

65822 XBD <**unistd.h**>

65823 **CHANGE HISTORY**

65824 First released in Issue 4, Version 2.

65825 **Issue 5**

65826 Moved from X/OPEN UNIX extension to BASE.

65827 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the  
65828 *exec* family of functions, not just *execve()*.

65829 **Issue 7**

65830 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-  
65831 group-ID to be the same as the real group ID.

65832 **Issue 8**

65833 Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to  
65834 SEE ALSO.

65835 **NAME**

65836 setresgid — set real group ID, effective group ID, and saved set-group-ID

65837 **SYNOPSIS**

```
65838 XSI #include <unistd.h>
65839 int setresgid(gid_t rgid, gid_t egid, gid_t sgid);
```

65840 **DESCRIPTION**65841 The *setresgid()* function shall set the real group ID, effective group ID, and saved set-group-ID of  
65842 the calling process to the values specified by *rgid*, *egid*, and *sgid*, respectively.65843 If an argument is  $-1$ , the corresponding ID shall not be changed.65844 Only a process with appropriate privileges can set the real group ID, effective group ID, and  
65845 saved set-group-ID to any valid value.65846 A non-privileged process can set its real group ID, effective group ID, and saved set-group-ID,  
65847 each to one of the values that it currently holds in its real group ID, effective group ID, or saved  
65848 set-group-ID.65849 The real group ID, effective group ID, and saved set-group-ID can be set to different values in  
65850 the same call.

65851 Any supplementary group IDs of the calling process shall remain unchanged.

65852 **RETURN VALUE**65853 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
65854 indicate the error, and none of the IDs shall be changed.65855 **ERRORS**65856 The *setresgid()* function shall fail if:65857 [EINVAL] The value of the *rgid*, *egid*, or *sgid* argument is invalid or out-of-range.65858 [EPERM] The calling process does not have appropriate privileges and an attempt was  
65859 made to change the real group ID, effective group ID, or saved set-group-ID to  
65860 a value that is not currently present in one of those IDs.65861 **EXAMPLES**

65862 None.

65863 **APPLICATION USAGE**

65864 None.

65865 **RATIONALE**

65866 None.

65867 **FUTURE DIRECTIONS**

65868 None.

65869 **SEE ALSO**65870 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*,  
65871 *setregid()*, *setresuid()*, *setreuid()*, *setuid()*

65872 XBD &lt;unistd.h&gt;

65873 **CHANGE HISTORY**  
65874 First released in Issue 8.

65875 **NAME**

65876 setresuid — set real user ID, effective user ID, and saved set-user-ID

65877 **SYNOPSIS**

```
65878 XSI #include <unistd.h>
65879 int setresuid(uid_t ruid, uid_t euid, uid_t suid);
```

65880 **DESCRIPTION**

65881 The *setresuid()* function shall set the real user ID, effective user ID, and saved set-user-ID of the  
 65882 calling process to the values specified by *ruid*, *euid*, and *suid*, respectively.

65883 If an argument is  $-1$ , the corresponding ID shall not be changed.

65884 Only a process with appropriate privileges can set the real user ID, effective user ID, and saved  
 65885 set-user-ID to any valid value.

65886 A non-privileged process can set its real user ID, effective user ID, and saved set-user-ID, each to  
 65887 one of the values that it currently holds in its real user ID, effective user ID, or saved set-user-ID.

65888 The real user ID, effective user ID, and saved set-user-ID can be set to different values in the  
 65889 same call.

65890 **RETURN VALUE**

65891 Upon successful completion, 0 shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to  
 65892 indicate the error, and none of the IDs shall be changed.

65893 **ERRORS**

65894 The *setresuid()* function shall fail if:

65895 [EINVAL] The value of the *ruid*, *euid*, or *suid* argument is invalid or out-of-range.

65896 [EPERM] The calling process does not have appropriate privileges and an attempt was  
 65897 made to change the real user ID, effective user ID, or saved set-user-ID to a  
 65898 value that is not currently present in one of those IDs or an attempt was made  
 65899 to change the real user ID to a value not permitted by the implementation.

65900 **EXAMPLES**

65901 None.

65902 **APPLICATION USAGE**

65903 None.

65904 **RATIONALE**

65905 None.

65906 **FUTURE DIRECTIONS**

65907 None.

65908 **SEE ALSO**

65909 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*,  
 65910 *setregid()*, *setresgid()*, *setreuid()*, *setuid()*

65911 XBD [<unistd.h>](#)

65912 **CHANGE HISTORY**

65913 First released in Issue 8.

65914 **NAME**

65915 setreuid — set real and effective user IDs

65916 **SYNOPSIS**

```
65917 XSI #include <unistd.h>
65918 int setreuid(uid_t ruid, uid_t euid);
```

65919 **DESCRIPTION**

65920 The *setreuid()* function shall set the real and effective user IDs of the current process to the  
65921 values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is *-1*, the corresponding effective  
65922 or real user ID of the current process shall be left unchanged.

65923 A process with appropriate privileges can set either ID to any value. An unprivileged process  
65924 can only set the effective user ID if the *euid* argument is equal to either the real, effective, or  
65925 saved user ID of the process.

65926 If the real user ID is being set (*ruid* is not *-1*), or the effective user ID is being set to a value not  
65927 equal to the real user ID, then the saved set-user-ID of the current process shall be set equal to  
65928 the new effective user ID.

65929 It is unspecified whether a process without appropriate privileges is permitted to change the real  
65930 user ID to match the current effective user ID or saved set-user-ID of the process.

65931 **RETURN VALUE**

65932 Upon successful completion, 0 shall be returned. Otherwise, *-1* shall be returned and *errno* set to  
65933 indicate the error.

65934 **ERRORS**

65935 The *setreuid()* function shall fail if:

65936 [EINVAL] The value of the *ruid* or *euid* argument is invalid or out-of-range.

65937 [EPERM] The current process does not have appropriate privileges, and either an  
65938 attempt was made to change the effective user ID to a value other than the real  
65939 user ID or the saved set-user-ID or an attempt was made to change the real  
65940 user ID to a value not permitted by the implementation.

65941 **EXAMPLES**65942 **Setting the Effective User ID to the Real User ID**

65943 The following example sets the effective user ID of the calling process to the real user ID, so that  
65944 files created later will be owned by the current user. It also sets the saved set-user-ID to the real  
65945 user ID, so any future attempt to set the effective user ID back to its previous value will fail.

```
65946 #include <unistd.h>
65947 #include <sys/types.h>
65948 ...
65949 setreuid(getuid(), getuid());
65950 ...
```

65951 **APPLICATION USAGE**

65952 None.



65953 **RATIONALE**

65954 Earlier versions of this standard did not specify whether the saved set-user-ID was affected by  
65955 *setreuid()* calls. This version specifies common existing practice that constitutes an important  
65956 security feature. The ability to set both the effective user ID and saved set-user-ID to be the same  
65957 as the real user ID means that any security weakness in code that is executed after that point  
65958 cannot result in malicious code being executed with the previous effective user ID. Privileged  
65959 applications could already do this using just *setuid()*, but for non-privileged applications the  
65960 only standard method available is to use this feature of *setreuid()*.

65961 **FUTURE DIRECTIONS**

65962 None.

65963 **SEE ALSO**

65964 *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*,  
65965 *setregid()*, *setresgid()*, *setresuid()*, *setuid()*

65966 XBD <[unistd.h](#)>

65967 **CHANGE HISTORY**

65968 First released in Issue 4, Version 2.

65969 **Issue 5**

65970 Moved from X/OPEN UNIX extension to BASE.

65971 **Issue 7**

65972 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved  
65973 set-user-ID to be the same as the real user ID.

65974 **Issue 8**

65975 Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to  
65976 SEE ALSO.

65977 **NAME**

65978           setrlimit — control maximum resource consumption

65979 **SYNOPSIS**

65980           #include <sys/resource.h>

65981           int setrlimit(int *resource*, const struct rlimit \*rlp);

65982 **DESCRIPTION**

65983           Refer to [getrlimit\(\)](#).

65984 **NAME**  
65985       setservent — network services database functions

65986 **SYNOPSIS**  
65987       #include <netdb.h>  
65988       void setservent(int *stayopen*);

65989 **DESCRIPTION**  
65990       Refer to *endservent()*.

65991 **NAME**

65992 setsid — create session and set process group ID

65993 **SYNOPSIS**

65994 #include &lt;unistd.h&gt;

65995 pid\_t setsid(void);

65996 **DESCRIPTION**

65997 The *setsid()* function shall create a new session, if the calling process is not a process group leader. Upon return the calling process shall be the session leader of this new session, shall be the process group leader of a new process group, and shall have no controlling terminal. The process group ID of the calling process shall be set equal to the process ID of the calling process.

66001 The calling process shall be the only process in the new process group and the only process in the new session.

66003 **RETURN VALUE**

66004 Upon successful completion, *setsid()* shall return the value of the new process group ID of the calling process. Otherwise, it shall return `-1` and set *errno* to indicate the error.

66006 **ERRORS**66007 The *setsid()* function shall fail if:

66008 [EPERM] The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

66011 **EXAMPLES**

66012 None.

66013 **APPLICATION USAGE**

66014 None.

66015 **RATIONALE**

66016 The *setsid()* function is similar to the *setpgrp()* function of System V. System V, without job control, groups processes into process groups and creates new process groups via *setpgrp()*; only one process group may be part of a login session.

66019 Job control allows multiple process groups within a login session. In order to limit job control actions so that they can only affect processes in the same login session, this volume of POSIX.1-2024 adds the concept of a session that is created via *setsid()*. The *setsid()* function also creates the initial process group contained in the session. Additional process groups can be created via the *setpgid()* function. A System V process group would correspond to a POSIX System Interfaces session containing a single POSIX process group. Note that this function requires that the calling process not be a process group leader. The usual way to ensure this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function guarantees that the process ID of the new process does not match any existing process group ID.

66028 **FUTURE DIRECTIONS**

66029 None.

66030 **SEE ALSO**66031 [getsid\(\)](#), [setpgid\(\)](#)66032 XBD [<sys/types.h>](#), [<unistd.h>](#)

66033 **CHANGE HISTORY**

66034 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

66035 **Issue 6**

66036 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

66037 The following new requirements on POSIX implementations derive from alignment with the  
66038 Single UNIX Specification:

- 66039 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
66040 required for conforming implementations of previous POSIX specifications, it was not  
66041 required for UNIX applications.

66042 **Issue 7**

66043 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0570 [421] is applied.

66044 **Issue 8**

66045 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

66046 **NAME**

66047 setsockopt — set the socket options

66048 **SYNOPSIS**

66049 #include &lt;sys/socket.h&gt;

```
66050 int setsockopt(int socket, int level, int option_name,
66051               const void *option_value, socklen_t option_len);
```

66052 **DESCRIPTION**

66053 The *setsockopt()* function shall set the option specified by the *option\_name* argument, at the  
 66054 protocol level specified by the *level* argument, to the value pointed to by the *option\_value*  
 66055 argument for the socket associated with the file descriptor specified by the *socket* argument.

66056 The *level* argument specifies the protocol level at which the option resides. To set options at the  
 66057 socket level, specify the *level* argument as SOL\_SOCKET. To set options at other levels, supply  
 66058 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate  
 66059 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO\_TCP  
 66060 as defined in the <netinet/in.h> header.

66061 The *option\_name* argument specifies a single option to set. It can be one of the socket-level  
 66062 options defined in <sys/socket.h> and described in Section 2.10.16 (on page 554). If *option\_name*  
 66063 is equal to SO\_RCVTIMEO or SO\_SNDTIMEO and the implementation supports setting the  
 66064 option, it is unspecified whether the **struct timeval** pointed to by *option\_value* is stored as  
 66065 provided by this function or is rounded up to align with the resolution of the clock being used. If  
 66066 *setsockopt()* is called with *option\_name* equal to SO\_ACCEPTCONN, SO\_ERROR, or SO\_TYPE,  
 66067 the behavior is unspecified.

66068 **RETURN VALUE**

66069 Upon successful completion, *setsockopt()* shall return 0. Otherwise, -1 shall be returned and  
 66070 *errno* set to indicate the error.

66071 **ERRORS**

66072 The *setsockopt()* function shall fail if:

- |       |               |   |
|-------|---------------|---|
| 66073 | [EBADF]       | The <i>socket</i> argument is not a valid file descriptor.  |
| 66074 | [EDOM]        | The send and receive timeout values are too big to fit into the timeout fields in the socket structure. |
| 66075 |               |   |
| 66076 | [EINVAL]      | The specified option is invalid at the specified socket level or the socket has been shut down.         |
| 66077 |               |   |
| 66078 | [EISCONN]     | The socket is already connected, and a specified option cannot be set while the socket is connected.    |
| 66079 |               |   |
| 66080 | [ENOPROTOOPT] | The option is not supported by the protocol.  |
| 66081 |               |   |
| 66082 | [ENOTSOCK]    | The <i>socket</i> argument does not refer to a socket.  |
| 66083 |               | The <i>setsockopt()</i> function may fail if:   |
| 66084 | [ENOMEM]      | There was insufficient memory available for the operation to complete.                                  |
| 66085 | [ENOBUFS]     | Insufficient resources are available in the system to complete the call.                                |

**66086 EXAMPLES**

66087 None.

**66088 APPLICATION USAGE**

66089 The *setsockopt()* function provides an application program with the means to control socket  
66090 behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts,  
66091 or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options  
66092 available to *setsockopt()*.

66093 Options may exist at multiple protocol levels. The `SO_` options are always present at the  
66094 uppermost socket level.

66095 It is implementation-defined which socket options, if any, are inherited from a listening socket to  
66096 an accepted socket by *accept()* or *accept4()*.

**66097 RATIONALE**

66098 None.

**66099 FUTURE DIRECTIONS**

66100 None.

**66101 SEE ALSO**

66102 [Section 2.10](#) (on page 549), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*

66103 XBD `<netinet/in.h>`, `<sys/socket.h>`

**66104 CHANGE HISTORY**

66105 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

66106 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/125 is applied, updating the `SO_LINGER`  
66107 option in the DESCRIPTION to refer to the calling thread rather than the process.

**66108 Issue 7**

66109 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options  
66110 that is now in [Section 2.10.16](#) (on page 554).

66111 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0571 [369] is applied.

**66112 Issue 8**

66113 Austin Group Defect 1337 is applied, clarifying socket option default values.

66114 **NAME**

66115            setstate — switch pseudo-random number generator state arrays

66116 **SYNOPSIS**

```
66117 XSI        #include <stdlib.h>  
66118            char *setstate(char *state);
```

66119 **DESCRIPTION**

66120            Refer to *initstate()*.



66121 **NAME**

66122 setuid — set user ID

66123 **SYNOPSIS**

66124 #include &lt;unistd.h&gt;

66125 int setuid(uid\_t uid);

66126 **DESCRIPTION**66127 If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and  
66128 the saved set-user-ID of the calling process to *uid*.66129 If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the  
66130 saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-  
66131 user-ID shall remain unchanged.66132 The *setuid()* function shall not affect the supplementary group list in any way.66133 **RETURN VALUE**66134 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
66135 indicate the error.66136 **ERRORS**66137 The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more  
66138 of the following are true:66139 [EINVAL] The value of the *uid* argument is invalid and not supported by the  
66140 implementation.66141 [EPERM] The process does not have appropriate privileges and *uid* does not match the  
66142 real user ID or the saved set-user-ID.66143 **EXAMPLES**

66144 None.

66145 **APPLICATION USAGE**

66146 None.

66147 **RATIONALE**66148 The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged  
66149 processes reflect the behavior of different historical implementations. For portability, it is  
66150 recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions  
66151 instead.66152 The saved set-user-ID capability allows a program to regain the effective user ID established at  
66153 the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the  
66154 effective group ID established at the last *exec* call. These capabilities are derived from System V.  
66155 Without them, a program might have to run as superuser in order to perform the same  
66156 functions, because superuser can write on the user's files. This is a problem because such a  
66157 program can write on any user's files, and so must be carefully written to emulate the  
66158 permissions of the calling process properly. In System V, these capabilities have traditionally  
66159 been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The  
66160 fact that the behavior of those functions was different for privileged processes made them  
66161 difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently  
66162 for privileged and unprivileged users. When the caller had appropriate privileges, the function  
66163 set the real user ID, effective user ID, and saved set-user ID of the calling process on  
66164 implementations that supported it. When the caller did not have appropriate privileges, the  
66165 function set only the effective user ID, subject to permission checks. The former use is generally  
66166 needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

66167 scope of POSIX.1-2024. These utilities wish to change the user ID irrevocably to a new value,  
66168 generally that of an unprivileged user. The latter use is needed for conforming applications that  
66169 are installed with the set-user-ID bit and need to perform operations using the real user ID.

66170 POSIX.1-2024 augments the latter functionality with a mandatory feature named  
66171 `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user  
66172 ID back and forth between the values of its *exec*-time real user ID and effective user ID.  
66173 Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this  
66174 feature to work properly when it happened to be executed with (implementation-defined)  
66175 appropriate privileges. Furthermore, the application did not even have a means to tell whether it  
66176 had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as  
66177 evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by POSIX.1-2024.  
66178 However, there are implementors who have been reluctant to support it given the limitation  
66179 described above.

66180 The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which  
66181 always sets both the real and effective user IDs, like *setuid()* in POSIX.1-2024 for privileged  
66182 users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in POSIX.1-2024  
66183 for non-privileged users). This separation of functionality into distinct functions seems desirable.  
66184 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of  
66185 switching the effective user ID back and forth via *setreuid()*, which permits reversing the real  
66186 and effective user IDs. This model seems less desirable than the saved set-user-ID because the  
66187 real user ID changes as a side-effect. The current 4.4BSD includes saved effective IDs and uses  
66188 them for *seteuid()* and *setegid()* as described above. The *setreuid()* and *setregid()* functions will be  
66189 deprecated or removed.

66190 The solution here is:

- 66191 • Require that all implementations support the functionality of the saved set-user-ID, which  
66192 is set by the *exec* functions and by privileged calls to *setuid()*.
- 66193 • Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for  
66194 non-privileged and privileged processes.

66195 Historical systems have provided two mechanisms for a set-user-ID process to change its  
66196 effective user ID to be the same as its real user ID in such a way that it could return to the  
66197 original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID,  
66198 or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs.  
66199 The changes included in POSIX.1-2024 provide a new mechanism using *seteuid()* in conjunction  
66200 with a saved set-user-ID. Thus, all implementations with the new *seteuid()* mechanism will have  
66201 a saved set-user-ID for each process, and most of the behavior controlled by  
66202 `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined.  
66203 The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally  
66204 be required to maintain compatibility with the older mechanisms previously supported by their  
66205 systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS`  
66206 behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID  
66207 allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the  
66208 saved set-user-ID unmodified, the process would then have an effective user ID equal to the  
66209 original real user ID, and both real and saved set-user-ID would be equal to the original effective  
66210 user ID. In that state, the real user would be unable to kill the process, even though the effective  
66211 user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS`  
66212 was used. This is obviously not acceptable. The alternative choice, which is used in at least one  
66213 implementation, is to change the saved set-user-ID to the effective user ID during most calls to  
66214 *setreuid()*. The standard developers considered that alternative to be less correct than the  
66215 retention of the old behavior of *kill()* in such systems. Current conforming applications shall

66216 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to  
66217 check the saved set-user-ID rather than the effective user ID.

#### 66218 FUTURE DIRECTIONS

66219 None.

#### 66220 SEE ALSO

66221 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getresgid()*, *getresuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*,  
66222 *setregid()*, *setresgid()*, *setresuid()*, *setreuid()*

66223 XBD [<sys/types.h>](#), [<unistd.h>](#)

#### 66224 CHANGE HISTORY

66225 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 66226 Issue 6

66227 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

66228 The following new requirements on POSIX implementations derive from alignment with the  
66229 Single UNIX Specification:

- 66230 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
66231 required for conforming implementations of previous POSIX specifications, it was not  
66232 required for UNIX applications.
- 66233 • The functionality associated with `_POSIX_SAVED_IDS` is now mandatory. This is a FIPS  
66234 requirement.

66235 The following changes were made to align with the IEEE P1003.1a draft standard:

- 66236 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 66237 • A requirement that the supplementary group list is not affected is added.

#### 66238 Issue 8

66239 Austin Group Defect 1344 is applied, adding *getresgid()*, *getresuid()*, *setresgid()*, and *setresuid()* to  
66240 SEE ALSO.

66241 **NAME**

66242 setutxent — reset the user accounting database to the first entry

66243 **SYNOPSIS**

```
66244 XSI #include <utmpx.h>  
66245 void setutxent(void);
```

66246 **DESCRIPTION**

66247 Refer to *endutxent()*.

66248 **NAME**

66249 setvbuf — assign buffering to a stream

66250 **SYNOPSIS**

66251 #include &lt;stdio.h&gt;

66252 int setvbuf(FILE \*restrict stream, char \*restrict buf, int type,  
66253 size\_t size);66254 **DESCRIPTION**66255 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
66256 conflict between the requirements described here and the ISO C standard is unintentional. This  
66257 volume of POSIX.1-2024 defers to the ISO C standard.66258 The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an  
66259 open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is  
66260 performed on the stream. The argument *type* determines how *stream* shall be buffered, as  
66261 follows:

- 66262
- {\_IOFBF} shall cause input/output to be fully buffered.
  - {\_IOLBF} shall cause input/output to be line buffered.
  - {\_IONBF} shall cause input/output to be unbuffered.

66265 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by  
66266 *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the  
66267 size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are  
66268 unspecified.66269 For information about streams, see [Section 2.5](#) (on page 521).66270 **RETURN VALUE**66271 Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value  
66272 CX if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to  
66273 indicate the error.66274 **ERRORS**66275 The *setvbuf()* function may fail if:66276 CX [EBADF] The stream is not a memory stream and the file descriptor underlying the  
66277 stream is not valid.66278 **EXAMPLES**

66279 None.

66280 **APPLICATION USAGE**66281 A common source of error is allocating buffer space as an “automatic” variable in a code block,  
66282 and then failing to close the stream in the same block.66283 With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are  
66284 used for the buffer area.66285 Applications should note that many implementations only provide line buffering on input from  
66286 terminal devices.66287 **RATIONALE**

66288 None.

66289 **FUTURE DIRECTIONS**

66290 None.

66291 **SEE ALSO**66292 [Section 2.5](#) (on page 521), [fopen\(\)](#), [setbuf\(\)](#)66293 XBD [<stdio.h>](#)66294 **CHANGE HISTORY**

66295 First released in Issue 1. Derived from Issue 1 of the SVID.

66296 **Issue 6**

66297 Extensions beyond the ISO C standard are marked.

66298 The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.66299 **Issue 8**

66300 Austin Group Defect 1144 is applied, changing the [EBADF] error condition.

66301 **NAME**66302 shm\_open — open a shared memory object (**REALTIME**)66303 **SYNOPSIS**

```
66304 SHM #include <sys/mman.h>
66305 int shm_open(const char *name, int oflag, mode_t mode);
```

66306 **DESCRIPTION**

66307 The *shm\_open()* function shall establish a connection between a shared memory object and a file  
 66308 descriptor. It shall create an open file description that refers to the shared memory object and a  
 66309 file descriptor that refers to that open file description. The file descriptor shall be allocated as  
 66310 described in [Section 2.6](#) (on page 525), and can be used by other functions to refer to that shared  
 66311 memory object. The *name* argument points to a string naming a shared memory object. It is  
 66312 unspecified whether the name appears in the file system and is visible to other functions that  
 66313 take pathnames as arguments. The *name* argument conforms to the construction rules for a  
 66314 pathname, except that the interpretation of <slash> characters other than the leading <slash>  
 66315 character in *name* is implementation-defined, and that the length limits for the *name* argument  
 66316 are implementation-defined and need not be the same as the pathname limits {PATH\_MAX} and  
 66317 {NAME\_MAX}. If *name* begins with the <slash> character, then processes calling *shm\_open()*  
 66318 with the same value of *name* refer to the same shared memory object, as long as that name has  
 66319 not been removed. If *name* does not begin with the <slash> character, the effect is  
 66320 implementation-defined.

66321 If successful, *shm\_open()* shall return a file descriptor for the shared memory object. The open  
 66322 file description is new, and therefore the file descriptor does not share it with any other  
 66323 processes. It is unspecified whether the file offset is set. The FD\_CLOEXEC file descriptor flag  
 66324 associated with the new file descriptor shall be set.

66325 The file status flags and file access modes of the open file description shall be set according to  
 66326 the value of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags.  
 66327 Applications specify exactly one of the first two values (access modes) below in the value of  
 66328 *oflag*:

66329 O\_RDONLY Open for read access only.

66330 O\_RDWR Open for read or write access.

66331 Any combination of the remaining flags can be specified in the value of *oflag*:

66332 O\_CREAT If the shared memory object exists, this flag has no effect, except as noted  
 66333 under O\_EXCL below. Otherwise, the shared memory object is created. The  
 66334 user ID of the shared memory object shall be set to the effective user ID of the  
 66335 process. The group ID of the shared memory object shall be set to the effective  
 66336 group ID of the process; however, if the *name* argument is visible in the file  
 66337 system, the group ID may be set to the group ID of the containing directory.  
 66338 The permission bits of the shared memory object shall be set to the value of  
 66339 the *mode* argument except those set in the file mode creation mask of the  
 66340 process. When bits in *mode* other than the file permission bits are set, the effect  
 66341 is unspecified. The *mode* argument does not affect whether the shared memory  
 66342 object is opened for reading, for writing, or for both. The shared memory  
 66343 object has a size of zero.

66344 O\_EXCL If O\_EXCL and O\_CREAT are set, *shm\_open()* fails if the shared memory  
 66345 object exists. The check for the existence of the shared memory object and the  
 66346 creation of the object if it does not exist is atomic with respect to other

66347 processes executing *shm\_open()* naming the same shared memory object with  
 66348 O\_EXCL and O\_CREAT set. If O\_EXCL is set and O\_CREAT is not set, the  
 66349 result is undefined.

66350 O\_TRUNC If the shared memory object exists, and it is successfully opened O\_RDWR, the  
 66351 object shall be truncated to zero length and the mode and owner shall be  
 66352 unchanged by this function call. The result of using O\_TRUNC with  
 66353 O\_RDONLY is undefined.

66354 The following functions shall be atomic with respect to each other in the effects specified in  
 66355 POSIX.1-2024 when they operate on shared memory objects:

66356 *close()*, *ftruncate()*, *mmap()*, *shm\_open()*, *shm\_unlink()*

66357 If two threads each call one of these functions, each call shall either see all of the specified effects  
 66358 of the other call, or none of them. The requirement on the *close()* function shall also apply  
 66359 whenever a file descriptor is successfully closed, however caused (for example, as a consequence  
 66360 of calling *close()*, calling *dup2()*, or of process termination).

66361 When a shared memory object is created, the state of the shared memory object, including all  
 66362 data associated with the shared memory object, persists until the shared memory object is  
 66363 unlinked and all other references are gone. It is unspecified whether the name and shared  
 66364 memory object state remain valid after a system reboot.

#### 66365 RETURN VALUE

66366 Upon successful completion, the *shm\_open()* function shall return a non-negative integer  
 66367 representing the file descriptor. Otherwise, it shall return  $-1$  and set *errno* to indicate the error.

#### 66368 ERRORS

66369 The *shm\_open()* function shall fail if:

66370 [EACCES] The shared memory object exists and the permissions specified by *oflag* are  
 66371 denied, or the shared memory object does not exist and permission to create  
 66372 the shared memory object is denied, or O\_TRUNC is specified and write  
 66373 permission is denied.

66374 [EEXIST] O\_CREAT and O\_EXCL are set and the named shared memory object already  
 66375 exists.

66376 [EINTR] The *shm\_open()* operation was interrupted by a signal.

66377 [EINVAL] The *shm\_open()* operation is not supported for the given name.

66378 [EMFILE] All file descriptors available to the process are currently open.

66379 [ENFILE] Too many shared memory objects are currently open in the system.

66380 [ENOENT] O\_CREAT is not set and the named shared memory object does not exist.

66381 [ENOSPC] There is insufficient space for the creation of the new shared memory object.

66382 The *shm\_open()* function may fail if:

66383 [ENAMETOOLONG]

66384 The length of the *name* argument exceeds  $\{\_POSIX\_PATH\_MAX\}$  on systems  
 66385 XSI that do not support the XSI option `or exceeds  $\{\_XOPEN\_PATH\_MAX\}$  on XSI`  
 66386 `systems,` or has a pathname component that is longer than  
 66387 XSI  $\{\_POSIX\_NAME\_MAX\}$  on systems that do not support the XSI option `or`  
 66388 `longer than  $\{\_XOPEN\_NAME\_MAX\}$  on XSI systems.`



66389 **EXAMPLES**66390 **Creating and Mapping a Shared Memory Object**

66391 The following code segment demonstrates the use of *shm\_open()* to create a shared memory  
66392 object which is then sized using *ftruncate()* before being mapped into the process address space  
66393 using *mmap()*:

```
66394 #include <unistd.h>
66395 #include <sys/mman.h>
66396 ...
66397 #define MAX_LEN 10000
66398 struct region {          /* Defines "structure" of shared memory */
66399     int len;
66400     char buf[MAX_LEN];
66401 };
66402 struct region *rptr;
66403 int fd;
66404
66404 /* Create shared memory object and set its size */
66405 fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
66406 if (fd == -1)
66407     /* Handle error */;
66408
66408 if (ftruncate(fd, sizeof(struct region)) == -1)
66409     /* Handle error */;
66410
66410 /* Map shared memory object */
66411 rptr = mmap(NULL, sizeof(struct region),
66412            PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
66413 if (rptr == MAP_FAILED)
66414     /* Handle error */;
66415
66415 /* Now we can refer to mapped region using fields of rptr;
66416    for example, rptr->len */
66417 ...
```

66418 **APPLICATION USAGE**

66419 None.

66420 **RATIONALE**

66421 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a  
66422 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared  
66423 Memory Objects option is supported, the *shm\_open()* function shall obtain a descriptor to the  
66424 shared memory object to be mapped.

66425 There is ample precedent for having a file descriptor represent several types of objects. In the  
66426 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.  
66427 Many implementations simply have an operations vector, which is indexed by the file descriptor  
66428 type and does very different operations. Note that in some cases the file descriptor passed to  
66429 generic operations on file descriptors is returned by *open()* or *creat()* and in some cases returned  
66430 by alternate functions, such as *pipe()*. The latter technique is used by *shm\_open()*.

66431 Note that such shared memory objects can actually be implemented as mapped files. In both  
66432 cases, the size can be set after the open using *ftruncate()*. The *shm\_open()* function itself does not

66433 create a shared object of a specified size because this would duplicate an extant function that set  
66434 the size of an object referenced by a file descriptor.

66435 On implementations where memory objects are implemented using the existing file system, the  
66436 *shm\_open()* function may be implemented using a macro that invokes *open()*, and the  
66437 *shm\_unlink()* function may be implemented using a macro that invokes *unlink()*.

66438 For implementations without a permanent file system, the definition of the name of the memory  
66439 objects is allowed not to survive a system reboot. Note that this allows systems with a  
66440 permanent file system to implement memory objects as data structures internal to the  
66441 implementation as well.

66442 On implementations that choose to implement memory objects using memory directly, a  
66443 *shm\_open()* followed by an *truncate()* and *close()* can be used to preallocate a shared memory  
66444 area and to set the size of that preallocation. This may be necessary for systems without virtual  
66445 memory hardware support in order to ensure that the memory is contiguous.

66446 The set of valid open flags to *shm\_open()* was restricted to *O\_RDONLY*, *O\_RDWR*, *O\_CREAT*,  
66447 and *O\_TRUNC* because these could be easily implemented on most memory mapping systems.  
66448 This volume of POSIX.1-2024 is silent on the results if the implementation cannot supply the  
66449 requested file access because of implementation-defined reasons, including hardware ones. The  
66450 *O\_CLOEXEC* open flag is not listed, because all shared memory objects are created with the  
66451 *FD\_CLOEXEC* flag already set; an application can later use *fcntl()* to clear that flag to allow the  
66452 shared memory file descriptor to be preserved across the *exec* family of functions.

66453 The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the  
66454 implementation cannot complete a request.

66455 [EACCES] indicates for implementation-defined reasons, probably hardware-related, that the  
66456 implementation cannot comply with a requested mode because it conflicts with another  
66457 requested mode. An example might be that an application desires to open a memory object two  
66458 times, mapping different areas with different access modes. If the implementation cannot map a  
66459 single area into a process space in two places, which would be required if different access modes  
66460 were required for the two areas, then the implementation may inform the application at the time  
66461 of the second open.

66462 [ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the  
66463 implementation cannot comply with a requested mode at all. An example would be that the  
66464 hardware of the implementation cannot support write-only shared memory areas.

66465 On all implementations, it may be desirable to restrict the location of the memory objects to  
66466 specific file systems for performance (such as a RAM disk) or implementation-defined reasons  
66467 (shared memory supported directly only on certain file systems). The *shm\_open()* function may  
66468 be used to enforce these restrictions. There are a number of methods available to the application  
66469 to determine an appropriate name of the file or the location of an appropriate directory. One way  
66470 is from the environment via *getenv()*. Another would be from a configuration file.

66471 This volume of POSIX.1-2024 specifies that memory objects have initial contents of zero when  
66472 created. This is consistent with current behavior for both files and newly allocated memory. For  
66473 those implementations that use physical memory, it would be possible that such  
66474 implementations could simply use available memory and give it to the process uninitialized.  
66475 This, however, is not consistent with standard behavior for the uninitialized data area, the stack,  
66476 and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security  
66477 reasons. Thus, initializing memory objects to zero is required.

**66478 FUTURE DIRECTIONS**

66479 A future version might require the *shm\_open()* and *shm\_unlink()* functions to have semantics  
66480 similar to normal file system operations.

**66481 SEE ALSO**

66482 Section 2.6 (on page 525), *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*,  
66483 *shm\_unlink()*, *umask()*

66484 XBD <fcntl.h>, <sys/mman.h>

**66485 CHANGE HISTORY**

66486 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**66487 Issue 6**

66488 The *shm\_open()* function is marked as part of the Shared Memory Objects option.

66489 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
66490 implementation does not support the Shared Memory Objects option.

66491 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/126 is applied, adding the example to the  
66492 EXAMPLES section.

**66493 Issue 7**

66494 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and  
66495 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

66496 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

66497 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the  
66498 user ID and group ID of the shared memory object.

66499 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0324 [835], XSH/TC2-2008/0325 [835],  
66500 and XSH/TC2-2008/0326 [835] are applied.

**66501 Issue 8**

66502 Austin Group Defect 411 is applied, adding a sentence about O\_CLOEXEC to the RATIONALE  
66503 section.

66504 Austin Group Defect 593 is applied, removing a reference to <fcntl.h> from the DESCRIPTION  
66505 section.

66506 Austin Group Defect 695 is applied, adding atomicity requirements to operations on shared  
66507 memory objects.

66508 **NAME**66509 shm\_unlink — remove a shared memory object (**REALTIME**)66510 **SYNOPSIS**

```
66511 SHM #include <sys/mman.h>
66512 int shm_unlink(const char *name);
```

66513 **DESCRIPTION**

66514 The *shm\_unlink()* function shall remove the name of the shared memory object named by the  
66515 string pointed to by *name*.

66516 If one or more references to the shared memory object exist when the object is unlinked, the  
66517 name shall be removed before *shm\_unlink()* returns, but the removal of the memory object  
66518 contents shall be postponed until all open and map references to the shared memory object have  
66519 been removed.

66520 Even if the object continues to exist after the last *shm\_unlink()*, reuse of the name shall  
66521 subsequently cause *shm\_open()* to behave as if no shared memory object of this name exists (that  
66522 is, *shm\_open()* shall fail if **O\_CREAT** is not set, or shall create a new shared memory object if  
66523 **O\_CREAT** is set).

66524 **RETURN VALUE**

66525 Upon successful completion, a value of zero shall be returned. Otherwise, a value of  $-1$  shall be  
66526 returned and *errno* set to indicate the error. If  $-1$  is returned, the named shared memory object  
66527 shall not be changed by this function call.

66528 **ERRORS**

66529 The *shm\_unlink()* function shall fail if:

66530 [EACCES] Permission is denied to unlink the named shared memory object.

66531 [ENOENT] The named shared memory object does not exist.

66532 The *shm\_unlink()* function may fail if:

66533 [ENAMETOOLONG]

66534 The length of the *name* argument exceeds  $\{\_POSIX\_PATH\_MAX\}$  on systems  
66535 XSI that do not support the XSI option or exceeds  $\{\_XOPEN\_PATH\_MAX\}$  on XSI  
66536 systems, or has a pathname component that is longer than  
66537 XSI  $\{\_POSIX\_NAME\_MAX\}$  on systems that do not support the XSI option or  
66538 longer than  $\{\_XOPEN\_NAME\_MAX\}$  on XSI systems. A call to *shm\_unlink()*  
66539 with a *name* argument that contains the same shared memory object name as  
66540 was previously used in a successful *shm\_open()* call shall not give an  
66541 [ENAMETOOLONG] error.

66542 **EXAMPLES**

66543 None.

66544 **APPLICATION USAGE**

66545 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual  
66546 fashion. Names of memory objects that were allocated with *shm\_open()* are deleted with  
66547 *shm\_unlink()*. Note that the actual memory object is not destroyed until the last close and  
66548 unmap on it have occurred if it was already in use.

**66549 RATIONALE**

66550 None.

**66551 FUTURE DIRECTIONS**

66552 A future version might require the *shm\_open()* and *shm\_unlink()* functions to have semantics  
66553 similar to normal file system operations.

**66554 SEE ALSO**

66555 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm\_open()*

66556 XBD <[sys/mman.h](#)>

**66557 CHANGE HISTORY**

66558 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

**66559 Issue 6**

66560 The *shm\_unlink()* function is marked as part of the Shared Memory Objects option.

66561 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm\_unlink()*  
66562 will not attach to the old shared memory object.

66563 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
66564 implementation does not support the Shared Memory Objects option.

**66565 Issue 7**

66566 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a  
66567 ``shall fail'' to a ``may fail'' error.

66568 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

66569 **NAME**

66570 shmat — XSI shared memory attach operation

66571 **SYNOPSIS**

```
66572 XSI #include <sys/shm.h>
66573 void *shmat(int shmId, const void *shmaddr, int shmflg);
```

66574 **DESCRIPTION**

66575 The *shmat()* function operates on XSI shared memory (see XBD [Section 3.332](#), on page 80). It is  
 66576 unspecified whether this function interoperates with the realtime interprocess communication  
 66577 facilities defined in [Section 2.8](#) (on page 527).

66578 The *shmat()* function attaches the shared memory segment associated with the shared memory  
 66579 identifier specified by *shmId* to the address space of the calling process. The segment is attached  
 66580 at the address specified by one of the following criteria:

- 66581 • If *shmaddr* is a null pointer, the segment is attached at the first available address as selected  
 66582 by the system.
- 66583 • If *shmaddr* is not a null pointer and (*shmflg* & SHM\_RND) is non-zero, the segment is  
 66584 attached at the address given by ((**char** \*)*shmaddr* - ((**uintptr\_t**)*shmaddr* % SHMLBA)). The  
 66585 character '%' is the C-language remainder operator.
- 66586 • If *shmaddr* is not a null pointer and (*shmflg* & SHM\_RND) is 0, the segment is attached at  
 66587 the address given by *shmaddr*.
- 66588 • The segment is attached for reading if (*shmflg* & SHM\_RDONLY) is non-zero and the  
 66589 calling process has read permission; otherwise, if (*shmflg* & SHM\_RDONLY) is 0 and the  
 66590 calling process has read and write permission, the segment is attached for reading and  
 66591 writing.

66592 **RETURN VALUE**

66593 Upon successful completion, *shmat()* shall increment the value of *shm\_nattch* in the data  
 66594 structure associated with the shared memory ID of the attached shared memory segment and  
 66595 return the segment's start address. Also, the *shm\_atime* timestamp shall be set to the current  
 66596 time, as described in [Section 2.7.1](#) (on page 526).

66597 Otherwise, the shared memory segment shall not be attached, *shmat()* shall return  
 66598 SHM\_FAILED, and *errno* shall be set to indicate the error.

66599 **ERRORS**

66600 The *shmat()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 66601 | [EACCES] | Operation permission is denied to the calling process; see <a href="#">Section 2.7</a> (on page 526).  |
| 66602 |          |  |
| 66603 | [EINVAL] | The value of <i>shmId</i> is not a valid shared memory identifier, the <i>shmaddr</i> is not a null pointer, and the value of (( <b>char</b> *) <i>shmaddr</i> - (( <b>uintptr_t</b> ) <i>shmaddr</i> % SHMLBA)) is an illegal address for attaching shared memory; or the <i>shmaddr</i> is not a null pointer, ( <i>shmflg</i> & SHM_RND) is 0, and the value of <i>shmaddr</i> is an illegal address for attaching shared memory. |
| 66604 |          |  |
| 66605 |          |  |
| 66606 |          |  |
| 66607 |          |  |
| 66608 | [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit.  |
| 66609 |          |  |
| 66610 | [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment.   |
| 66611 |          |  |

**66612 EXAMPLES**

66613 None.

**66614 APPLICATION USAGE**

66615 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
66616 Application developers who need to use IPC should design their applications so that modules  
66617 using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to use the  
66618 alternative interfaces.

**66619 RATIONALE**

66620 The symbol SHM\_FAILED is used for the failure return of *shmat()* for consistency with  
66621 MAP\_FAILED for *mmap()*. However, SHM\_FAILED is required to have the same value as  
66622 ((**void \***)(**intptr\_t**)-1) to provide backwards compatibility for applications written to earlier  
66623 versions of this standard, where the failure return was specified as (**void \***)-1. This means that  
66624 implementations need to ensure that *shmat()* cannot return ((**void \***)(**intptr\_t**)-1) on a successful  
66625 call.

**66626 FUTURE DIRECTIONS**

66627 None.

**66628 SEE ALSO**

66629 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*,  
66630 *shmget()*, *shm\_open()*, *shm\_unlink()*

66631 XBD [Section 3.332](#) (on page 80), [<sys/shm.h>](#)

**66632 CHANGE HISTORY**

66633 First released in Issue 2. Derived from Issue 2 of the SVID.

**66634 Issue 5**

66635 Moved from SHARED MEMORY to BASE.

66636 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
66637 DIRECTIONS to a new APPLICATION USAGE section.

**66638 Issue 6**

66639 The Open Group Corrigendum U021/13 is applied.

**66640 Issue 7**

66641 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0572 [345] is applied.

66642 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0327 [522] is applied.

**66643 Issue 8**

66644 Austin Group Defect 1237 is applied, changing ``*shmaddr - ...*'' to ``(char \*)*shmaddr - ...*''.

66645 Austin Group Defect 1238 is applied, changing the DESCRIPTION to avoid an ambiguous use of  
66646 ``it''.

66647 Austin Group Defect 1239 is applied, adding SHM\_FAILED.

66648 **NAME**

66649 shmctl — XSI shared memory control operations

66650 **SYNOPSIS**

```
66651 XSI      #include <sys/shm.h>
66652      int shmctl(int shmid, int cmd, struct shmids *buf);
```

66653 **DESCRIPTION**

66654 The *shmctl()* function operates on XSI shared memory (see XBD [Section 3.332](#), on page 80). It is  
 66655 unspecified whether this function interoperates with the realtime interprocess communication  
 66656 facilities defined in [Section 2.8](#) (on page 527).

66657 The *shmctl()* function provides a variety of shared memory control operations as specified by  
 66658 *cmd*. The following values for *cmd* are available:

66659 **IPC\_STAT** Place the current value of each member of the **shmids** data structure  
 66660 associated with *shmid* into the structure pointed to by *buf*. The contents of the  
 66661 structure are defined in **<sys/shm.h>**.

66662 **IPC\_SET** Set the value of the following members of the **shmids** data structure  
 66663 associated with *shmid* to the corresponding value found in the structure  
 66664 pointed to by *buf*:

```
66665     shm_perm.uid
66666     shm_perm.gid
66667     shm_perm.mode     Low-order nine bits.
```

66668 Also, the *shm\_ctime* timestamp shall be set to the current time, as described in  
 66669 [Section 2.7.1](#) (on page 526).

66670 **IPC\_SET** can only be executed by a process that has an effective user ID equal  
 66671 to either that of a process with appropriate privileges or to the value of  
 66672 *shm\_perm.cuid* or *shm\_perm.uid* in the **shmids** data structure associated with  
 66673 *shmid*.

66674 **IPC\_RMID** Remove the shared memory identifier specified by *shmid* from the system. The  
 66675 shared memory segment and **shmids** data structure associated with it shall  
 66676 be destroyed when all processes with the segment attached have either  
 66677 detached the segment or terminated. If the segment is not attached to any  
 66678 process, it shall be destroyed immediately. **IPC\_RMID** can only be executed by  
 66679 a process that has an effective user ID equal to either that of a process with  
 66680 appropriate privileges or to the value of *shm\_perm.cuid* or *shm\_perm.uid* in the  
 66681 **shmids** data structure associated with *shmid*.

66682 **RETURN VALUE**

66683 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return -1 and set *errno* to  
 66684 indicate the error.

66685 **ERRORS**

66686 The *shmctl()* function shall fail if:

66687 **[EACCES]** The argument *cmd* is equal to **IPC\_STAT** and the calling process does not have  
 66688 read permission; see [Section 2.7](#) (on page 526).

66689 **[EINVAL]** The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*  
 66690 is not a valid command.



66691 [EPERM] The argument *cmd* is equal to IPC\_RMID or IPC\_SET and the effective user ID  
66692 of the calling process is not equal to that of a process with appropriate  
66693 privileges and it is not equal to the value of *shm\_perm.cuid* or *shm\_perm.uid* in  
66694 the data structure associated with *shmid*.

66695 The *shmctl()* function may fail if:

66696 [EOVERFLOW] The *cmd* argument is IPC\_STAT and the *gid* or *uid* value is too large to be  
66697 stored in the structure pointed to by the *buf* argument.

#### 66698 EXAMPLES

66699 None.

#### 66700 APPLICATION USAGE

66701 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
66702 Application developers who need to use IPC should design their applications so that modules  
66703 using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to use the  
66704 alternative interfaces.

#### 66705 RATIONALE

66706 None.

#### 66707 FUTURE DIRECTIONS

66708 None.

#### 66709 SEE ALSO

66710 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), [shmat\(\)](#), [shmdt\(\)](#), [shmget\(\)](#), [shm\\_open\(\)](#),  
66711 [shm\\_unlink\(\)](#)

66712 XBD [Section 3.332](#) (on page 80), [<sys/shm.h>](#)

#### 66713 CHANGE HISTORY

66714 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 66715 Issue 5

66716 Moved from SHARED MEMORY to BASE.

66717 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
66718 DIRECTIONS to a new APPLICATION USAGE section.

#### 66719 Issue 7

66720 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0573 [345] is applied.

#### 66721 Issue 8

66722 Austin Group Defect 1240 is applied, clarifying the description of IPC\_RMID.

66723 **NAME**

66724 shmdt — XSI shared memory detach operation

66725 **SYNOPSIS**

```
66726 XSI #include <sys/shm.h>
66727 int shmdt(const void *shmaddr);
```

66728 **DESCRIPTION**

66729 The *shmdt()* function operates on XSI shared memory (see XBD [Section 3.332](#), on page 80). It is  
66730 unspecified whether this function interoperates with the realtime interprocess communication  
66731 facilities defined in [Section 2.8](#) (on page 527).

66732 The *shmdt()* function detaches the shared memory segment located at the address specified by  
66733 *shmaddr* from the address space of the calling process.

66734 **RETURN VALUE**

66735 Upon successful completion, *shmdt()* shall decrement the value of *shm\_nattch* in the data  
66736 structure associated with the shared memory ID of the attached shared memory segment and  
66737 return 0. Also, the *shm\_dtime* timestamp shall be set to the current time, as described in [Section](#)  
66738 [2.7.1](#) (on page 526).

66739 Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return  $-1$ , and *errno*  
66740 shall be set to indicate the error.

66741 **ERRORS**66742 The *shmdt()* function shall fail if:

66743 [EINVAL] The value of *shmaddr* is not the data segment start address of a shared memory  
66744 segment.

66745 **EXAMPLES**

66746 None.

66747 **APPLICATION USAGE**

66748 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
66749 Application developers who need to use IPC should design their applications so that modules  
66750 using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to use the  
66751 alternative interfaces.

66752 **RATIONALE**

66753 None.

66754 **FUTURE DIRECTIONS**

66755 None.

66756 **SEE ALSO**

66757 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*,  
66758 *shmget()*, *shm\_open()*, *shm\_unlink()*

66759 XBD [Section 3.332](#) (on page 80), [<sys/shm.h>](#)

66760 **CHANGE HISTORY**

66761 First released in Issue 2. Derived from Issue 2 of the SVID.

66762 **Issue 5**

66763 Moved from SHARED MEMORY to BASE.

66764 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
66765 DIRECTIONS to a new APPLICATION USAGE section.

66766 **Issue 7**

66767 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0574 [345] is applied.

66768 **NAME**

66769 shmget — get an XSI shared memory segment

66770 **SYNOPSIS**

```
66771 XSI #include <sys/shm.h>
66772 int shmget(key_t key, size_t size, int shmflg);
```

66773 **DESCRIPTION**

66774 The *shmget()* function operates on XSI shared memory (see XBD [Section 3.332](#), on page 80). It is  
 66775 unspecified whether this function interoperates with the realtime interprocess communication  
 66776 facilities defined in [Section 2.8](#) (on page 527).

66777 The *shmget()* function shall return the shared memory identifier associated with *key*.

66778 A shared memory identifier, associated data structure, and shared memory segment of at least  
 66779 *size* bytes (see [<sys/shm.h>](#)) are created for *key* if one of the following is true:

- 66780 • The argument *key* is equal to `IPC_PRIVATE`.
- 66781 • The argument *key* does not already have a shared memory identifier associated with it and  
 66782 (*shmflg* & `IPC_CREAT`) is non-zero.

66783 Upon creation, the data structure associated with the new shared memory identifier shall be  
 66784 initialized as follows:

- 66785 • The values of *shm\_perm.cuid*, *shm\_perm.uid*, *shm\_perm.cgid*, and *shm\_perm.gid* are set to the  
 66786 effective user ID and effective group ID, respectively, of the calling process.
- 66787 • The low-order nine bits of *shm\_perm.mode* are set to the low-order nine bits of *shmflg*.
- 66788 • The value of *shm\_segsz* is set to the value of *size*.
- 66789 • The values of *shm\_lpid*, *shm\_nattch*, *shm\_atime*, and *shm\_dtime* are set to 0.
- 66790 • The value of *shm\_ctime* is set to the current time, as described in [Section 2.7.1](#) (on page 526).

66791 When the shared memory segment is created, it shall be initialized with all zero values.

66792 **RETURN VALUE**

66793 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared  
 66794 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

66795 **ERRORS**

66796 The *shmget()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 66797 | [EACCES] | A shared memory identifier exists for <i>key</i> but operation permission as specified by the low-order nine bits of <i>shmflg</i> would not be granted; see <a href="#">Section 2.7</a> (on page 526). |
| 66798 |          |   |
| 66799 |          |   |
| 66800 | [EEXIST] | A shared memory identifier exists for the argument <i>key</i> but ( <i>shmflg</i> & <code>IPC_CREAT</code> ) && ( <i>shmflg</i> & <code>IPC_EXCL</code> ) is non-zero.                                  |
| 66801 |          |   |
| 66802 | [EINVAL] | A shared memory segment is to be created and the value of <i>size</i> is less than the system-imposed minimum or greater than the system-imposed maximum.   |
| 66803 |          |   |
| 66804 | [EINVAL] | No shared memory segment is to be created and a shared memory segment exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> .                                   |
| 66805 |          |   |
| 66806 | [ENOENT] | A shared memory identifier does not exist for the argument <i>key</i> and ( <i>shmflg</i> & <code>IPC_CREAT</code> ) is 0.  |
| 66807 |          |   |

66808 [ENOMEM] A shared memory identifier and associated shared memory segment are to be  
66809 created, but the amount of available physical memory is not sufficient to fill  
66810 the request.

66811 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on  
66812 the maximum number of allowed shared memory identifiers system-wide  
66813 would be exceeded.

#### 66814 EXAMPLES

66815 None.

#### 66816 APPLICATION USAGE

66817 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.  
66818 Application developers who need to use IPC should design their applications so that modules  
66819 using the IPC routines described in [Section 2.7](#) (on page 526) can be easily modified to use the  
66820 alternative interfaces.

#### 66821 RATIONALE

66822 None.

#### 66823 FUTURE DIRECTIONS

66824 None.

#### 66825 SEE ALSO

66826 [Section 2.7](#) (on page 526), [Section 2.8](#) (on page 527), [ftok\(\)](#), [shmat\(\)](#), [shmctl\(\)](#), [shmdt\(\)](#), [shm\\_open\(\)](#),  
66827 [shm\\_unlink\(\)](#)

66828 XBD [Section 3.332](#) (on page 80), [<sys/shm.h>](#)

#### 66829 CHANGE HISTORY

66830 First released in Issue 2. Derived from Issue 2 of the SVID.

#### 66831 Issue 5

66832 Moved from SHARED MEMORY to BASE.

66833 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE  
66834 DIRECTIONS to a new APPLICATION USAGE section.

#### 66835 Issue 7

66836 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0575 [345], XSH/TC1-2008/0576 [363],  
66837 and XSH/TC1-2008/0577 [344] are applied.

66838 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0328 [640] is applied.

66839 **NAME**

66840 shutdown — shut down socket send and receive operations

66841 **SYNOPSIS**

66842 #include &lt;sys/socket.h&gt;

66843 int shutdown(int *socket*, int *how*);66844 **DESCRIPTION**66845 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket  
66846 associated with the file descriptor *socket* to be shut down.66847 The *shutdown()* function takes the following arguments:66848 *socket* Specifies the file descriptor of the socket.66849 *how* Specifies the type of shutdown. The values are as follows:

66850 SHUT\_RD Disables further receive operations.

66851 SHUT\_WR Disables further send operations.

66852 SHUT\_RDWR Disables further send and receive operations.

66853 The *shutdown()* function disables subsequent send and/or receive operations on a socket,  
66854 depending on the value of the *how* argument.66855 **RETURN VALUE**66856 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*  
66857 set to indicate the error.66858 **ERRORS**66859 The *shutdown()* function shall fail if:66860 [EBADF] The *socket* argument is not a valid file descriptor.66861 [EINVAL] The *how* argument is invalid.

66862 [ENOTCONN] The socket is not connected.

66863 [ENOTSOCK] The *socket* argument does not refer to a socket.66864 The *shutdown()* function may fail if:

66865 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

66866 **EXAMPLES**

66867 None.

66868 **APPLICATION USAGE**66869 The file descriptor remains open after *shutdown()* returns to the calling application.66870 **RATIONALE**

66871 None.

66872 **FUTURE DIRECTIONS**

66873 None.

66874 **SEE ALSO**66875 *close()*, *getsockopt()*, *pselect()*, *read()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*,  
66876 *socket()*, *write()*

66877 XBD &lt;sys/socket.h&gt;

66878 **CHANGE HISTORY**

66879 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

66880 **Issue 8**

66881 Austin Group Defect 1475 is applied, changing the APPLICATION USAGE section and adding  
66882 *close()* to the SEE ALSO section.

66883 **NAME**

66884 sig2str, str2sig — translate between signal names and numbers

66885 **SYNOPSIS**

```
66886 CX #include <signal.h>
66887 int sig2str(int signum, char *str);
66888 int str2sig(const char *restrict str, int *restrict pnum);
```

66889 **DESCRIPTION**

66890 The *sig2str()* function shall translate the signal number specified by *signum* to a signal name and  
 66891 shall store this string in the location specified by *str*. The application shall ensure that *str* points  
 66892 to a location that can store the string including the terminating null byte. The symbolic constant  
 66893 SIG2STR\_MAX defined in **<signal.h>** gives the maximum number of bytes required.

66894 If *signum* is equal to 0, the behavior is unspecified.

66895 If *signum* is equal to one of the symbolic constants listed in the table of signal numbers in  
 66896 **<signal.h>**, the stored signal name shall be the name of the symbolic constant without the **SIG**  
 66897 prefix.

66898 If *signum* is equal to SIGRTMIN or SIGRTMAX, the stored string shall be "RTMIN" or "RTMAX",  
 66899 respectively.

66900 If *signum* is between SIGRTMIN+1 and (SIGRTMIN+SIGRTMAX)/2 inclusive, the stored string  
 66901 shall be of the form "RTMIN+n", where *n* is the shortest decimal representation of the value of  
 66902 *signum*-SIGRTMIN.

66903 If *signum* is between (SIGRTMIN+SIGRTMAX)/2 + 1 and SIGRTMAX-1 inclusive, the stored  
 66904 string shall be either of the form "RTMIN+n" or of the form "RTMAX-m", where *n* is the shortest  
 66905 decimal representation of the value of *signum*-SIGRTMIN and *m* is the shortest decimal  
 66906 representation of the value of SIGRTMAX-*signum*.

66907 If *signum* is a valid, supported signal number, is either less than SIGRTMIN or greater than  
 66908 SIGRTMAX, and is not equal to one of the symbolic constants listed in the table of signal  
 66909 numbers in **<signal.h>**, the stored string shall uniquely identify the signal number *signum* in an  
 66910 unspecified manner.

66911 The *str2sig()* function shall translate the signal name in the string pointed to by *str* to a signal  
 66912 number and shall store this value in the location specified by *pnum*.

66913 If *str* points to a string containing the name of one of the symbolic constants listed in the table of  
 66914 signal numbers in **<signal.h>**, without the **SIG** prefix, the stored signal number shall be equal to  
 66915 the value of the symbolic constant.

66916 If *str* points to the string "RTMIN" or "RTMAX", the stored value shall be equal to SIGRTMIN or  
 66917 SIGRTMAX, respectively.

66918 If *str* points to a string of the form "RTMIN+n", where *n* is a decimal representation of a number  
 66919 between 1 and SIGRTMAX-SIGRTMIN-1 inclusive, the stored value shall be equal to  
 66920 SIGRTMIN+n.

66921 If *str* points to a string of the form "RTMAX-n", where *n* is a decimal representation of a number  
 66922 between 1 and SIGRTMAX-SIGRTMIN-1 inclusive, the stored value shall be equal to  
 66923 SIGRTMAX-n.

66924 If *str* points to a string containing a decimal representation of a valid, supported signal number,  
 66925 the value stored in the location pointed to by *pnum* shall be equal to that number.



66926 If *str* points to a string containing a decimal representation of the value 0 and the string was not  
66927 returned by a previous successful call to *sig2str()* with a *signum* argument of 0, the behavior is  
66928 unspecified.

66929 If *str* points to a string returned by a previous successful call to *sig2str(signum, str)*, the value  
66930 stored in the location pointed to by *pnum* shall be equal to *signum*.

66931 If *str* points to a string that does not meet any of the above criteria, *str2sig()* shall store a value in  
66932 the location pointed to by *pnum* if and only if it recognizes the string as an additional  
66933 implementation-dependent form of signal name.

#### 66934 RETURN VALUE

66935 If *signum* is a valid, supported signal number (that is, one for which *kill()* does not return  $-1$   
66936 with *errno* set to [EINVAL]), the *sig2str()* function shall return 0; otherwise, if *signum* is not equal  
66937 to 0, it shall return  $-1$ .

66938 If *str2sig()* stores a value in the location pointed to by *pnum*, it shall return 0; otherwise, it shall  
66939 return  $-1$ .

#### 66940 ERRORS

66941 No errors are defined.

#### 66942 EXAMPLES

66943 None.

#### 66944 APPLICATION USAGE

66945 None.

#### 66946 RATIONALE

66947 Historical versions of these functions translated a *signum* value 0 to "EXIT" (and vice versa), so  
66948 that they could be used by the shell for the *trap* utility. When adding the functions to this  
66949 standard, the standard developers felt that they should be aimed at more general-purpose use,  
66950 and consequently requiring this behavior did not seem appropriate and so the behavior in this  
66951 case has been made unspecified.

#### 66952 FUTURE DIRECTIONS

66953 None.

#### 66954 SEE ALSO

66955 [\*kill\(\)\*](#), [\*sigaction\(\)\*](#), [\*strsignal\(\)\*](#)

66956 XBD [\*\*<signal.h>\*\*](#)

#### 66957 CHANGE HISTORY

66958 First released in Issue 8.

## 66959 NAME

66960 sigaction — examine and change a signal action

## 66961 SYNOPSIS

```
66962 CX #include <signal.h>
66963 int sigaction(int sig, const struct sigaction *restrict act,
66964 struct sigaction *restrict oact);
```

## 66965 DESCRIPTION

66966 The *sigaction()* function allows the calling process to examine and/or specify the action to be  
 66967 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are  
 66968 defined in `<signal.h>`.

66969 The structure **sigaction**, used to describe an action to be taken, is defined in the `<signal.h>`  
 66970 header to include at least the following members:

Member Type	Member Name	Description
<b>void(*) (int)</b>	<i>sa_handler</i>	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
<b>sigset_t</b>	<i>sa_mask</i>	Additional set of signals to be blocked during execution of signal-catching function.
<b>int</b>	<i>sa_flags</i>	Special flags to affect behavior of signal.
<b>void(*) (int, siginfo_t *, void *)</b>	<i>sa_sigaction</i>	Pointer to a signal-catching function.

66980 The storage occupied by *sa\_handler* and *sa\_sigaction* may overlap, and a conforming application  
 66981 shall not use both simultaneously.

66982 If the argument *act* is not a null pointer, it points to a structure specifying the action to be  
 66983 associated with the specified signal. If the argument *oact* is not a null pointer, the action  
 66984 previously associated with the signal is stored in the location pointed to by the argument *oact*. If  
 66985 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to  
 66986 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall  
 66987 not be added to the signal mask using this mechanism; this restriction shall be enforced by the  
 66988 system without causing an error to be indicated.

66989 If the SA\_SIGINFO flag (see below) is cleared in the *sa\_flags* field of the **sigaction** structure, the  
 66990 *sa\_handler* field identifies the action to be associated with the specified signal. If the  
 66991 SA\_SIGINFO flag is set in the *sa\_flags* field, the *sa\_sigaction* field specifies a signal-catching  
 66992 function.

66993 The *sa\_flags* field can be used to modify the behavior of the specified signal.

66994 The following flags, defined in the `<signal.h>` header, can be set in *sa\_flags*:

66995 XSI SA\_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children  
 66996 continue.

66997 If *sig* is SIGCHLD and the SA\_NOCLDSTOP flag is not set in *sa\_flags*, and  
 66998 the implementation supports the SIGCHLD signal, then a SIGCHLD  
 66999 signal shall be generated for the calling process whenever any of its child  
 67000 XSI processes stop and a SIGCHLD signal may be generated for the calling  
 67001 process whenever any of its stopped child processes are continued. If *sig*  
 67002 is SIGCHLD and the SA\_NOCLDSTOP flag is set in *sa\_flags*, then the  
 67003 implementation shall not generate a SIGCHLD signal in this way.

67004	XSI	<b>SA_ONSTACK</b>	If set and an alternate signal stack has been declared with <i>sigaltstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.
67005			
67006			
67007		<b>SA_RESETHAND</b>	If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.
67008			
67009		<b>Note:</b>	SIGILL and SIGTRAP cannot be automatically reset when delivered;
67010			the system silently enforces this restriction.
67011			Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.
67012			
67013			In addition, if this flag is set, <i>sigaction()</i> may behave as if the SA_NODEFER flag were also set.
67014			
67015		<b>SA_RESTART</b>	This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If an interruptible function which uses a timeout is restarted, the duration of the timeout following the restart is set to an unspecified value that does not exceed the original timeout value. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].
67016			
67017			
67018			
67019			
67020			
67021			
67022			
67023		<b>SA_SIGINFO</b>	If cleared and the signal is caught, the signal-catching function shall be entered as:
67024			
67025			<pre>void func(int signo);</pre>
67026			where <i>signo</i> is the only argument to the signal-catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal-catching function and the application shall not modify the <i>sa_sigaction</i> member.
67027			
67028			
67029			
67030			If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:
67031			
67032			<pre>void func(int signo, siginfo_t *info, void *context);</pre>
67033			where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type <b>siginfo_t</b> explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type <b>ucontext_t</b> to refer to the receiving thread's context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal-catching function and the application shall not modify the <i>sa_handler</i> member.
67034			
67035			
67036			
67037			
67038			
67039			
67040			
67041			The <i>si_signo</i> member contains the system-generated signal number.
67042	XSI		The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
67043			
67044			
67045			The <i>si_code</i> member contains a code identifying the cause of the signal, as described in <a href="#">Section 2.4.3</a> (on page 516).
67046			

67047 XSI SA\_NOCLDWAIT If *sig* does not equal SIGCHLD, the behavior is unspecified. Otherwise,  
67048 the behavior of the SA\_NOCLDWAIT flag is as specified in [Consequences](#)  
67049 [of Process Termination](#) (on page 568).

67050 SA\_NODEFER If set and *sig* is caught, *sig* shall not be added to the thread's signal mask  
67051 on entry to the signal handler unless it is included in *sa\_mask*. Otherwise,  
67052 *sig* shall always be added to the thread's signal mask on entry to the  
67053 signal handler.

67054 When a signal is caught by a signal-catching function installed by *sigaction()*, a new signal mask  
67055 is calculated and installed for the duration of the signal-catching function (or until a call to either  
67056 *sigprocmask()* or *sigsuspend()* is made). This mask is formed by taking the union of the current  
67057 signal mask and the value of the *sa\_mask* for the signal being delivered, and unless  
67058 SA\_NODEFER or SA\_RESETHAND is set, then including the signal being delivered. If and  
67059 when the user's signal handler returns normally, the original signal mask is restored.

67060 Once an action is installed for a specific signal, it shall remain installed until another action is  
67061 explicitly requested (by another call to *sigaction()*), until the SA\_RESETHAND flag causes  
67062 resetting of the handler, or until one of the *exec* functions is called.

67063 If the previous action for *sig* had been established by *signal()*, the values of the fields returned in  
67064 the structure pointed to by *oact* are unspecified, and in particular *oact->sa\_handler* is not  
67065 necessarily the same value passed to *signal()*. However, if a pointer to the same structure or a  
67066 copy thereof is passed to a subsequent call to *sigaction()* via the *act* argument, handling of the  
67067 signal shall be as if the original call to *signal()* were repeated.

67068 If *sigaction()* fails, no new signal handler is installed.

67069 It is unspecified whether an attempt to set the action for a signal that cannot be caught or  
67070 ignored to SIG\_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

67071 If SA\_SIGINFO is not set in *sa\_flags*, then the disposition of subsequent occurrences of *sig* when  
67072 it is already pending is implementation-defined; the signal-catching function shall be invoked  
67073 with a single argument. If SA\_SIGINFO is set in *sa\_flags*, then subsequent occurrences of *sig*  
67074 generated by *sigqueue()* or as a result of any signal-generating function that supports the  
67075 specification of an application-defined value (when *sig* is already pending) shall be queued in  
67076 FIFO order until delivered or accepted; the signal-catching function shall be invoked with three  
67077 arguments. The application specified value is passed to the signal-catching function as the  
67078 *si\_value* member of the **siginfo\_t** structure.

67079 The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the  
67080 same signal is unspecified.

67081 **RETURN VALUE**

67082 Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall  
67083 be set to indicate the error, and no new signal-catching function shall be installed.

67084 **ERRORS**67085 The *sigaction()* function shall fail if:67086 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a  
67087 signal that cannot be caught or ignore a signal that cannot be ignored.67088 The *sigaction()* function may fail if:67089 [EINVAL] An attempt was made to set the action to SIG\_DFL for a signal that cannot be  
67090 caught or ignored (or both).67091 In addition, on systems that do not support the XSI option, the *sigaction()* function may fail if the  
67092 SA\_SIGINFO flag is set in the *sa\_flags* field of the **sigaction** structure for a signal not in the range  
67093 SIGRTMIN to SIGRTMAX.67094 **EXAMPLES**67095 **Establishing a Signal Handler**67096 The following example demonstrates the use of *sigaction()* to establish a handler for the SIGINT  
67097 signal.

```

67098 #include <signal.h>
67099 static void handler(int signum)
67100 {
67101     /* Take appropriate actions for signal delivery */
67102 }
67103 int main(void)
67104 {
67105     struct sigaction sa;
67106     sa.sa_handler = handler;
67107     sigemptyset(&sa.sa_mask);
67108     sa.sa_flags = SA_RESTART; /* Restart functions if
67109                             interrupted by handler */
67110     if (sigaction(SIGINT, &sa, NULL) == -1)
67111         /* Handle error */;
67112     /* Further code */
67113 }

```

67114 **APPLICATION USAGE**

67115 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In  
67116 particular, *sigaction()* and *signal()* should not be used in the same process to control the same  
67117 signal. The behavior of async-signal-safe functions, as defined in their respective  
67118 DESCRIPTION sections, is as specified by this volume of POSIX.1-2024, regardless of invocation  
67119 from a signal-catching function. This is the only intended meaning of the statement that async-  
67120 signal-safe functions may be used in signal-catching functions without restrictions. Applications  
67121 must still consider all effects of such functions on such things as data structures, files, and  
67122 process state. In particular, application developers need to consider the restrictions on  
67123 interactions when interrupting *sleep()* and interactions among multiple handles for a file  
67124 description. The fact that any specific function is listed as async-signal-safe does not necessarily  
67125 mean that invocation of that function from a signal-catching function is recommended.

67126 In order to prevent errors arising from interrupting non-async-signal-safe function calls,  
67127 applications should protect calls to these functions either by blocking the appropriate signals or

67128 through the use of some programmatic semaphore (see *semget()*, *sem\_init()*, *sem\_open()*, and so  
 67129 on). Note in particular that even the “safe” functions may modify *errno*; the signal-catching  
 67130 function, if not executing as an independent thread, should save and restore its value in order to  
 67131 avoid the possibility that delivery of a signal in between an error return from a function that sets  
 67132 *errno* and the subsequent examination of *errno* could result in the signal-catching function  
 67133 changing the value of *errno*. Naturally, the same principles apply to the async-signal-safety of  
 67134 application routines and asynchronous data access. Note that *longjmp()* and *siglongjmp()* are not  
 67135 in the list of async-signal-safe functions. This is because the code executing after *longjmp()* and  
 67136 *siglongjmp()* can call any unsafe functions with the same danger as calling those unsafe  
 67137 functions directly from the signal handler. Applications that use *longjmp()* and *siglongjmp()* from  
 67138 within signal handlers require rigorous protection in order to be portable. Many of the other  
 67139 functions that are excluded from the list are traditionally implemented using either *malloc()* or  
 67140 *free()* functions or the standard I/O library, both of which traditionally use data structures in a  
 67141 non-async-signal-safe manner. Since any combination of different functions using a common  
 67142 data structure can cause async-signal-safety problems, this volume of POSIX.1-2024 does not  
 67143 define the behavior when any unsafe function is called in a signal handler that interrupts an  
 67144 unsafe function.

67145 Usually, the signal is executed on the stack that was in effect before the signal was delivered. An  
 67146 alternate stack may be specified to receive a subset of the signals being caught.

67147 When the signal handler returns, the receiving thread resumes execution at the point it was  
 67148 interrupted unless the signal handler makes other arrangements. If *longjmp()* is used to leave the  
 67149 signal handler, then the signal mask must be explicitly restored.

67150 This volume of POSIX.1-2024 defines the third argument of a signal handling function when  
 67151 SA\_SIGINFO is set as a **void \*** instead of a **ucontext\_t \***, but without requiring type checking.  
 67152 New applications should explicitly cast the third argument of the signal handling function to  
 67153 **ucontext\_t \***.

67154 The BSD optional four argument signal handling function is not supported by this volume of  
 67155 POSIX.1-2024. The BSD declaration would be:

```
67156 void handler(int sig, int code, struct sigcontext *scp,  
67157             char *addr);
```

67158 where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer  
 67159 to the **sigcontext** structure, and *addr* is additional address information. Much the same  
 67160 information is available in the objects pointed to by the second argument of the signal handler  
 67161 specified when SA\_SIGINFO is set.

67162 Since the *sigaction()* function is allowed but not required to set SA\_NODEFER when the  
 67163 application sets the SA\_RESETHAND flag, applications which depend on the SA\_RESETHAND  
 67164 functionality for the newly installed signal handler must always explicitly set SA\_NODEFER  
 67165 when they set SA\_RESETHAND in order to be portable.

67166 See also the rationale for Realtime Signal Generation and Delivery in XRAT [Section B.2.4.2](#) (on  
 67167 page 3749).

## 67168 RATIONALE

67169 Although this volume of POSIX.1-2024 requires that signals that cannot be ignored shall not be  
 67170 added to the signal mask when a signal-catching function is entered, there is no explicit  
 67171 requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact*  
 67172 argument. In other words, if SIGKILL is included in the *sa\_mask* field of *act*, it is unspecified  
 67173 whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa\_mask*  
 67174 field of *oact*.

67175 The SA\_NOCLDSTOP flag, when supplied in the *act->sa\_flags* parameter, allows overloading  
 67176 SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated  
 67177 child. Most conforming applications that catch SIGCHLD are expected to install signal-catching  
 67178 functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on  
 67179 each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of  
 67180 interest, the use of the SA\_NOCLDSTOP flag can prevent the overhead from invoking the  
 67181 signal-catching routine when they stop.

67182 Some historical implementations also define other mechanisms for stopping processes, such as  
 67183 the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when  
 67184 processes stop due to this mechanism; however, that is beyond the scope of this volume of  
 67185 POSIX.1-2024.

67186 This volume of POSIX.1-2024 requires that calls to *sigaction()* that supply a NULL *act* argument  
 67187 succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or  
 67188 SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases  
 67189 and, in this respect, their behavior varies from *sigaction()*.

67190 This volume of POSIX.1-2024 requires that *sigaction()* properly save and restore a signal action  
 67191 set up by the ISO C standard *signal()* function. However, there is no guarantee that the reverse is  
 67192 true, nor could there be given the greater amount of information conveyed by the **sigaction**  
 67193 structure. Because of this, applications should avoid using both functions for the same signal in  
 67194 the same process. Since this cannot always be avoided in case of general-purpose library  
 67195 routines, they should always be implemented with *sigaction()*.

67196 It was intended that the *signal()* function should be implementable as a library routine using  
 67197 *sigaction()*.

67198 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990  
 67199 standard to allow the application to request on a per-signal basis via an additional signal action  
 67200 flag that the extra parameters, including the application-defined signal value, if any, be passed to  
 67201 the signal-catching function.

#### 67202 FUTURE DIRECTIONS

67203 None.

#### 67204 SEE ALSO

67205 Section 2.4 (on page 513), *exec*, *\_Exit()*, *kill()*, *longjmp()*, *pthread\_sigmask()*, *raise()*, *semget()*,  
 67206 *sem\_init()*, *sem\_open()*, *sig2str()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,  
 67207 *sigismember()*, *signal()*, *sigsuspend()*, *wait()*, *waitid()*

67208 XBD <**signal.h**>

#### 67209 CHANGE HISTORY

67210 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 67211 Issue 5

67212 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX  
 67213 Threads Extension.

67214 In the DESCRIPTION, the second argument to *func* when SA\_SIGINFO is set is no longer  
 67215 permitted to be NULL, and the description of permitted **siginfo\_t** contents is expanded by  
 67216 reference to <**signal.h**>.

67217 Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP]  
 67218 error is deleted.

67219 **Issue 6**

67220 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on  
67221 Other Functions”, a reference to *sigpending()* is added.

67222 In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal  
67223 Effects on Other Functions” are moved to a separate section of this volume of POSIX.1-2024.

67224 Text describing functionality from the Realtime Signals Extension option is marked.

67225 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 67226 • The [ENOTSUP] error condition is added.

67227 The normative text is updated to avoid use of the term “must” for application requirements.

67228 The **restrict** keyword is added to the *sigaction()* prototype for alignment with the  
67229 ISO/IEC 9899: 1999 standard.

67230 References to the *wait3()* function are removed.

67231 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
67232 extension over the ISO C standard.

67233 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table  
67234 describing the **sigaction** structure.

67235 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/127 is applied, removing text from the  
67236 DESCRIPTION duplicated later in the same section.

67237 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/128 is applied, updating the  
67238 DESCRIPTION and APPLICATION USAGE sections. Changes are made to refer to the thread  
67239 rather than the process.

67240 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/129 is applied, adding the example to the  
67241 EXAMPLES section.

67242 **Issue 7**

67243 Austin Group Interpretation 1003.1-2001 #004 is applied.

67244 Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the  
67245 SA\_NODEFER flag with respect to the signal mask, and clarifying the SA\_RESTART flag for  
67246 interrupted functions which use timeouts.

67247 Austin Group Interpretation 1003.1-2001 #156 is applied.

67248 SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section.

67249 SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement  
67250 that when the SA\_RESETHAND flag is set, *sigaction()* shall behave as if the SA\_NODEFER flag  
67251 were also set.

67252 Functionality relating to the Realtime Signals Extension option is moved to the Base.

67253 The description of the *si\_code* member is replaced with a reference to [Section 2.4.3](#) (on page 516).

67254 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0578 [66] and XSH/TC1-2008/0579  
67255 [140] are applied.

67256 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0329 [690] and XSH/TC2-2008/0330  
67257 [491] are applied.



67258 **Issue 8**

67259 Austin Group Defect 1138 is applied, adding *sig2str()* to the SEE ALSO section.

67260 Austin Group Defect 1195 is applied, changing `main()` to `main(void)`.

67261 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

67262 **NAME**

67263 sigaddset — add a signal to a signal set

67264 **SYNOPSIS**

```
67265 CX #include <signal.h>
67266 int sigaddset(sigset_t *set, int signo);
```

67267 **DESCRIPTION**

67268 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed  
67269 to by *set*.

67270 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
67271 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
67272 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
67273 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
67274 *sigwaitinfo()*, the results are undefined.

67275 **RETURN VALUE**

67276 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return -1 and set *errno*  
67277 to indicate the error.

67278 **ERRORS**

67279 The *sigaddset()* function may fail if:

67280 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

67281 **EXAMPLES**

67282 None.

67283 **APPLICATION USAGE**

67284 None.

67285 **RATIONALE**

67286 None.

67287 **FUTURE DIRECTIONS**

67288 None.

67289 **SEE ALSO**

67290 Section 2.4 (on page 513), *pthread\_sigmask()*, *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,  
67291 *sigismember()*, *sigpending()*, *sigsuspend()*

67292 XBD [<signal.h>](#)

67293 **CHANGE HISTORY**

67294 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67295 **Issue 5**

67296 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
67297 previous issues.

67298 **Issue 6**

67299 The normative text is updated to avoid use of the term “must” for application requirements.

67300 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
67301 extension over the ISO C standard.

67302 **NAME**

67303 sigaltstack — set and get signal alternate stack context

67304 **SYNOPSIS**

```
67305 XSI #include <signal.h>
67306 int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

67307 **DESCRIPTION**

67308 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack  
 67309 for signal handlers for the current thread. Signals that have been explicitly declared to execute  
 67310 on the alternate stack shall be delivered on the alternate stack.

67311 If *ss* is not a null pointer, it points to a **stack\_t** structure that specifies the alternate signal stack  
 67312 that shall take effect upon return from *sigaltstack()*. The *ss\_flags* member specifies the new stack  
 67313 state. If it is set to *SS\_DISABLE*, the stack is disabled and *ss\_sp* and *ss\_size* are ignored.  
 67314 Otherwise, the stack shall be enabled, and the *ss\_sp* and *ss\_size* members specify the new address  
 67315 and size of the stack.

67316 The range of addresses starting at *ss\_sp* up to but not including *ss\_sp+ss\_size* is available to the  
 67317 implementation for use as the stack. This function makes no assumptions regarding which end  
 67318 is the stack base and in which direction the stack grows as items are pushed.

67319 If *oss* is not a null pointer, upon successful completion it shall point to a **stack\_t** structure that  
 67320 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss\_sp*  
 67321 and *ss\_size* members specify the address and size of that stack. The *ss\_flags* member specifies the  
 67322 stack's state, and may contain one of the following values:

67323 **SS\_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to  
 67324 modify the alternate signal stack while the process is executing on it fail. This  
 67325 flag shall not be modified by processes.

67326 **SS\_DISABLE** The alternate signal stack is currently disabled.

67327 The value *SIGSTKSZ* is a system default specifying the number of bytes that would be used to  
 67328 cover the usual case when manually allocating an alternate stack area. The value *MINSIGSTKSZ*  
 67329 is defined to be the minimum stack size for a signal handler. In computing an alternate stack  
 67330 size, a program should add that amount to its stack requirements to allow for the system  
 67331 implementation overhead. The constants *SS\_ONSTACK*, *SS\_DISABLE*, *SIGSTKSZ*, and  
 67332 *MINSIGSTKSZ* are defined in **<signal.h>**.

67333 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new  
 67334 process image.

67335 In some implementations, a signal (whether or not indicated to execute on the alternate stack)  
 67336 shall always execute on the alternate stack if it is delivered while another signal is being caught  
 67337 using the alternate stack.

67338 Use of this function by library threads that are not bound to kernel-scheduled entities results in  
 67339 undefined behavior.

67340 **RETURN VALUE**

67341 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return *-1* and set *errno*  
 67342 to indicate the error.

67343 **ERRORS**67344 The *sigaltstack()* function shall fail if:67345 [EINVAL] The *ss* argument is not a null pointer, and the *ss\_flags* member pointed to by *ss*  
67346 has *SS\_ONSTACK* or invalid flags set.67347 [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*.

67348 [EPERM] An attempt was made to modify an active stack.

67349 **EXAMPLES**67350 **Allocating Memory for an Alternate Stack**

67351 The following example illustrates a method for allocating memory for an alternate stack.

```
67352 #include <signal.h>
67353 ...
67354 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
67355     /* Error return. */
67356     sigstk.ss_size = SIGSTKSZ;
67357     sigstk.ss_flags = 0;
67358     if (sigaltstack(&sigstk, (stack_t *)0) < 0)
67359         perror("sigaltstack");
```

67360 **APPLICATION USAGE**

67361 On some implementations, stack space is automatically extended as needed. On those  
67362 implementations, automatic extension is typically not available for an alternate stack. If the stack  
67363 overflows, the behavior is undefined.

67364 **RATIONALE**

67365 None.

67366 **FUTURE DIRECTIONS**

67367 None.

67368 **SEE ALSO**67369 [Section 2.4](#) (on page 513), *exec*, *sigaction()*, *sigsetjmp()*67370 XBD [<signal.h>](#)67371 **CHANGE HISTORY**

67372 First released in Issue 4, Version 2.

67373 **Issue 5**

67374 Moved from X/OPEN UNIX extension to BASE.

67375 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in  
67376 previous issues.

67377 **Issue 6**

67378 The normative text is updated to avoid use of the term “must” for application requirements.

67379 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the  
67380 ISO/IEC 9899:1999 standard.

67381 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence  
67382 to include “for the current thread”.

67383 **Issue 8**  
67384

Austin Group Defect 1187 is applied, changing the description of the [EINVAL] error.

67385 **NAME**

67386 sigdelset — delete a signal from a signal set

67387 **SYNOPSIS**

```
67388 CX #include <signal.h>
67389 int sigdelset(sigset_t *set, int signo);
```

67390 **DESCRIPTION**

67391 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set  
 67392 pointed to by *set*.

67393 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
 67394 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
 67395 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
 67396 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
 67397 *sigwaitinfo()*, the results are undefined.

67398 **RETURN VALUE**

67399 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno*  
 67400 to indicate the error.

67401 **ERRORS**

67402 The *sigdelset()* function may fail if:

67403 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
 67404 number.

67405 **EXAMPLES**

67406 None.

67407 **APPLICATION USAGE**

67408 None.

67409 **RATIONALE**

67410 None.

67411 **FUTURE DIRECTIONS**

67412 None.

67413 **SEE ALSO**

67414 Section 2.4 (on page 513), *pthread\_sigmask()*, *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*,  
 67415 *sigismember()*, *sigpending()*, *sigsuspend()*

67416 XBD [<signal.h>](#)

67417 **CHANGE HISTORY**

67418 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67419 **Issue 5**

67420 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
 67421 previous issues.

67422 **Issue 6**

67423 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
 67424 extension over the ISO C standard.

**67425 NAME**

67426 sigemptyset — initialize and empty a signal set

**67427 SYNOPSIS**

```
67428 CX #include <signal.h>  
67429 int sigemptyset(sigset_t *set);
```

**67430 DESCRIPTION**

67431 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined  
67432 in POSIX.1-2024 are excluded.

**67433 RETURN VALUE**

67434 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set  
67435 *errno* to indicate the error.

**67436 ERRORS**

67437 No errors are defined.

**67438 EXAMPLES**

67439 None.

**67440 APPLICATION USAGE**

67441 None.

**67442 RATIONALE**

67443 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or set)  
67444 all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the  
67445 structure, such as a version field, to permit binary-compatibility between releases where the size  
67446 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any  
67447 other use of the signal set, even if such use is read-only (for example, as an argument to  
67448 *sigpending()*). This function is not intended for dynamic allocation.

67449 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or  
67450 exclude) all the signals defined in this volume of POSIX.1-2024. Although it is outside the scope  
67451 of this volume of POSIX.1-2024 to place this requirement on signals that are implemented as  
67452 extensions, it is recommended that implementation-defined signals also be affected by these  
67453 functions. However, there may be a good reason for a particular signal not to be affected. For  
67454 example, blocking or ignoring an implementation-defined signal may have undesirable side-  
67455 effects, whereas the default action for that signal is harmless. In such a case, it would be  
67456 preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.

67457 In early proposals there was no distinction between invalid and unsupported signals (the names  
67458 of optional signals that were not supported by an implementation were not defined by that  
67459 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.  
67460 With that distinction, it is not necessary to require implementations of these functions to  
67461 determine whether an optional signal is actually supported, as that could have a significant  
67462 performance impact for little value. The error could have been required for invalid signals and  
67463 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is  
67464 optional in both cases.

**67465 FUTURE DIRECTIONS**

67466 None.

67467 **SEE ALSO**

67468       Section 2.4 (on page 513), *pthread\_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,  
67469       *sigismember()*, *sigpending()*, *sigsuspend()*

67470       XBD <signal.h>

67471 **CHANGE HISTORY**

67472       First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67473 **Issue 6**

67474       The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an  
67475       extension over the ISO C standard.



67476 **NAME**

67477 sigfillset — initialize and fill a signal set

67478 **SYNOPSIS**

```
67479 CX #include <signal.h>
67480 int sigfillset(sigset_t *set);
```

67481 **DESCRIPTION**

67482 The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals  
67483 defined in this volume of POSIX.1-2024 are included.

67484 **RETURN VALUE**

67485 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*  
67486 to indicate the error.

67487 **ERRORS**

67488 No errors are defined.

67489 **EXAMPLES**

67490 None.

67491 **APPLICATION USAGE**

67492 None.

67493 **RATIONALE**

67494 Refer to *sigemptyset()* (on page 2055).

67495 **FUTURE DIRECTIONS**

67496 None.

67497 **SEE ALSO**

67498 Section 2.4 (on page 513), *pthread\_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*,  
67499 *sigismember()*, *sigpending()*, *sigsuspend()*

67500 XBD [<signal.h>](#)

67501 **CHANGE HISTORY**

67502 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67503 **Issue 6**

67504 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
67505 extension over the ISO C standard.

67506 **NAME**

67507 sigismember — test for a signal in a signal set

67508 **SYNOPSIS**

```
67509 CX #include <signal.h>
67510 int sigismember(const sigset_t *set, int signo);
```

67511 **DESCRIPTION**

67512 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set  
67513 pointed to by *set*.

67514 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type  
67515 **sigset\_t** prior to any other use of that object. If such an object is not initialized in this way, but is  
67516 nonetheless supplied as an argument to any of *pthread\_sigmask()*, *sigaction()*, *sigaddset()*,  
67517 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or  
67518 *sigwaitinfo()*, the results are undefined.

67519 **RETURN VALUE**

67520 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of  
67521 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.

67522 **ERRORS**

67523 The *sigismember()* function may fail if:

67524 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal  
67525 number.

67526 **EXAMPLES**

67527 None.

67528 **APPLICATION USAGE**

67529 None.

67530 **RATIONALE**

67531 None.

67532 **FUTURE DIRECTIONS**

67533 None.

67534 **SEE ALSO**

67535 Section 2.4 (on page 513), *pthread\_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,  
67536 *sigemptyset()*, *sigpending()*, *sigsuspend()*

67537 XBD [<signal.h>](#)

67538 **CHANGE HISTORY**

67539 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67540 **Issue 5**

67541 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
67542 previous issues.

67543 **Issue 6**

67544 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
67545 extension over the ISO C standard.

67546 **NAME**

67547 siglongjmp — non-local goto with signal handling

67548 **SYNOPSIS**

```
67549 CX #include <setjmp.h>
67550 _Noreturn void siglongjmp(sigjmp_buf env, int val);
```

67551 **DESCRIPTION**67552 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:

- 67553 • References to *setjmp()* shall be equivalent to *sigsetjmp()*.
- 67554 • The *siglongjmp()* function shall restore the saved signal mask if and only if the *env*
- 67555 argument was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

67556 **RETURN VALUE**

67557 After *siglongjmp()* is completed, thread execution shall continue as if the corresponding

67558 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function

67559 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

67560 **ERRORS**

67561 No errors are defined.

67562 **EXAMPLES**

67563 None.

67564 **APPLICATION USAGE**

67565 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant

67566 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

67567 **RATIONALE**

67568 None.

67569 **FUTURE DIRECTIONS**

67570 None.

67571 **SEE ALSO**67572 [longjmp\(\)](#), [pthread\\_sigmask\(\)](#), [setjmp\(\)](#), [sigsetjmp\(\)](#), [sigsuspend\(\)](#)67573 XBD [<setjmp.h>](#)67574 **CHANGE HISTORY**

67575 First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

67576 **Issue 5**

67577 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

67578 **Issue 6**67579 The DESCRIPTION is rewritten in terms of *longjmp()*.

67580 The SYNOPSIS is marked CX since the presence of this function in the [<setjmp.h>](#) header is an

67581 extension over the ISO C standard.

67582 **Issue 8**67583 Austin Group Defect 1302 is applied, adding `_Noreturn` to the SYNOPSIS.

67584 **NAME**

67585 signal — signal management

67586 **SYNOPSIS**

67587 #include &lt;signal.h&gt;

67588 void (\*signal(int sig, void (\*func)(int)))(int);

67589 **DESCRIPTION**

67590 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 67591 conflict between the requirements described here and the ISO C standard is unintentional. This  
 67592 volume of POSIX.1-2024 defers to the ISO C standard.

67593 The *signal()* function chooses one of three ways in which receipt of the signal number *sig* is to be  
 67594 subsequently handled. If the value of *func* is SIG\_DFL, default handling for that signal shall  
 67595 occur. If the value of *func* is SIG\_IGN, the signal shall be ignored. Otherwise, the application  
 67596 shall ensure that *func* points to a function to be called when that signal occurs. An invocation of  
 67597 such a function because of a signal, or (recursively) of any further functions called by that  
 67598 invocation (other than functions in the standard library), is called a “signal handler”.

67599 When a signal occurs, and *func* points to a function, it is implementation-defined whether the  
 67600 equivalent of a:

67601 `signal(sig, SIG_DFL);`

67602 is executed or the implementation prevents some implementation-defined set of signals (at least  
 67603 including *sig*) from occurring until the current signal handling has completed. (If the value of *sig*  
 67604 is SIGILL, the implementation may alternatively define that no action is taken.) Next the  
 67605 equivalent of:

67606 `(*func)(sig);`

67607 is executed. If and when the function returns, if the value of *sig* was SIGFPE, SIGILL, or  
 67608 SIGSEGV or any other implementation-defined value corresponding to a computational  
 67609 exception, the behavior is undefined. Otherwise, the program shall resume execution at the  
 67610 point it was interrupted. The ISO C standard places a restriction on applications relating to the  
 67611 CX use of *raise()* from signal handlers. This restriction does not apply to POSIX applications, as  
 67612 POSIX.1-2024 requires *raise()* to be async-signal-safe (see [Section 2.4.3](#), on page 516).

67613 CX If the process is multi-threaded, or if the process is single-threaded and a signal handler is  
 67614 executed other than as the result of:

67615 CX • The process calling *abort()*, *raise()*, *kill()*, *pthread\_kill()*, or *sigqueue()* to generate a signal  
 67616 that is not blocked

67617 CX • A pending signal being unblocked and being delivered before the call that unblocked it  
 67618 returns

67619 the behavior is undefined if:

67620 CX • The signal handler refers to any object other than *errno* with static or thread storage  
 67621 CX duration that is not a lock-free atomic object, and not a non-modifiable object (for  
 67622 example, string literals, objects that were defined with a const-qualified type, and objects  
 67623 in memory that is mapped read-only), other than by assigning a value to an object  
 67624 CX declared as **volatile sig\_atomic\_t**, unless the previous modification (if any) to the object  
 67625 happens before the signal handler is called and the return from the signal handler happens  
 67626 before the next modification (if any) to the object.

67627 CX       • The signal handler calls any function defined in this standard other than one of the  
67628               functions listed in Section 2.4 (on page 513).

67629       At program start-up, the equivalent of:

67630       `signal(sig, SIG_IGN);`

67631       is executed for some signals, and the equivalent of:

67632       `signal(sig, SIG_DFL);`

67633 CX       is executed for all other signals (see *exec*).

67634       The *signal()* function shall not change the setting of *errno* if successful.

67635 CX       The *signal()* function is required to be thread-safe. (See Section 2.9.1 (on page 537).)

#### 67636 RETURN VALUE

67637       If the request can be honored, *signal()* shall return the value of *func* for the most recent call to  
67638       *signal()* for the specified signal *sig*. Otherwise, *SIG\_ERR* shall be returned and a positive value  
67639       shall be stored in *errno*.

#### 67640 ERRORS

67641       The *signal()* function shall fail if:

67642 CX       [EINVAL]       The *sig* argument is not a valid signal number or an attempt is made to catch a  
67643               signal that cannot be caught or ignore a signal that cannot be ignored.

67644       The *signal()* function may fail if:

67645 CX       [EINVAL]       An attempt was made to set the action to *SIG\_DFL* for a signal that cannot be  
67646               caught or ignored (or both).

#### 67647 EXAMPLES

67648       None.

#### 67649 APPLICATION USAGE

67650       The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling  
67651       signals; new applications should use *sigaction()* rather than *signal()*.

#### 67652 RATIONALE

67653       The ISO C standard says that the use of *signal()* in a multi-threaded program results in  
67654       undefined behavior. However, POSIX.1 has required *signal()* to be thread-safe since before  
67655       threads were added to the ISO C standard.

#### 67656 FUTURE DIRECTIONS

67657       None.

#### 67658 SEE ALSO

67659       Section 2.4 (on page 513), *exec*, *pause()*, *raise()*, *sigaction()*, *sigsuspend()*, *waitid()*

67660       XBD <signal.h>

#### 67661 CHANGE HISTORY

67662       First released in Issue 1. Derived from Issue 1 of the SVID.

#### 67663 Issue 5

67664       Moved from X/OPEN UNIX extension to BASE.

67665       The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of  
67666       the process to its original state before returning.

67667       The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends

67668 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to  
67669 [EINTR].

67670 **Issue 6**

67671 Extensions beyond the ISO C standard are marked.

67672 The normative text is updated to avoid use of the term “must” for application requirements.

67673 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

67674 References to the *wait3()* function are removed.

67675 The *sighold()*, *sigignore()*, *sigrelse()*, and *sigset()* functions are split out onto their own reference  
67676 page.

67677 **Issue 7**

67678 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0580 [275], XSH/TC1-2008/0581 [66],  
67679 and XSH/TC1-2008/0582 [105] are applied.

67680 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0331 [785] is applied.

67681 **Issue 8**

67682 Austin Group Defect 728 is applied, reducing the set of circumstances in which undefined  
67683 behavior results when a signal handler refers to an object with static or thread storage duration.

67684 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
67685 standard.

67686 **NAME**

67687 signbit — test sign

67688 **SYNOPSIS**

67689 #include &lt;math.h&gt;

67690 int signbit(real-floating x);

67691 **DESCRIPTION**

67692 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
67693 conflict between the requirements described here and the ISO C standard is unintentional. This  
67694 volume of POSIX.1-2024 defers to the ISO C standard.

67695 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,  
67696 zeros, and infinities have a sign bit.

67697 **RETURN VALUE**

67698 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is  
67699 negative.

67700 **ERRORS**

67701 No errors are defined.

67702 **EXAMPLES**

67703 None.

67704 **APPLICATION USAGE**

67705 None.

67706 **RATIONALE**

67707 None.

67708 **FUTURE DIRECTIONS**

67709 None.

67710 **SEE ALSO**67711 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*

67712 XBD &lt;math.h&gt;

67713 **CHANGE HISTORY**

67714 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

67715 **NAME**

67716 signgam — log gamma function

67717 **SYNOPSIS**

```
67718 XSI #include <math.h>  
67719 extern int signgam;
```

67720 **DESCRIPTION**

67721 Refer to *lgamma()*.



67722 **NAME**

67723 sigpending — examine pending signals

67724 **SYNOPSIS**

```
67725 CX #include <signal.h>
67726 int sigpending(sigset_t *set);
```

67727 **DESCRIPTION**

67728 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of  
67729 signals that are blocked from delivery to the calling thread and that are pending on the process  
67730 or the calling thread.

67731 **RETURN VALUE**

67732 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and  
67733 *errno* set to indicate the error.

67734 **ERRORS**

67735 No errors are defined.

67736 **EXAMPLES**

67737 None.

67738 **APPLICATION USAGE**

67739 None.

67740 **RATIONALE**

67741 None.

67742 **FUTURE DIRECTIONS**

67743 None.

67744 **SEE ALSO**67745 *exec*, *pthread\_sigmask()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*67746 XBD [<signal.h>](#)67747 **CHANGE HISTORY**

67748 First released in Issue 3.

67749 **Issue 5**

67750 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

67751 **Issue 6**

67752 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
67753 extension over the ISO C standard.

67754 **NAME**

67755 sigprocmask — examine and change blocked signals

67756 **SYNOPSIS**

```
67757 CX #include <signal.h>
67758 int sigprocmask(int how, const sigset_t *restrict set,
67759 sigset_t *restrict oset);
```

67760 **DESCRIPTION**

67761 Refer to [pthread\\_sigmask\(\)](#).

67762 **NAME**

67763 sigqueue — queue a signal to a process

67764 **SYNOPSIS**

```
67765 CX #include <signal.h>
67766 int sigqueue(pid_t pid, int signo, union sigval value);
```

67767 **DESCRIPTION**

67768 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value  
 67769 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking  
 67770 is performed but no signal is actually sent. The null signal can be used to check the validity of  
 67771 *pid*.

67772 The conditions required for a process to have permission to queue a signal to another process are  
 67773 the same as for the *kill()* function.

67774 The *sigqueue()* function shall return immediately. If SA\_SIGINFO is set for *signo* and if the  
 67775 resources were available to queue the signal, the signal shall be queued and sent to the receiving  
 67776 process. If SA\_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving  
 67777 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this  
 67778 call.

67779 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked  
 67780 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*  
 67781 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the  
 67782 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the  
 67783 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.  
 67784 The selection order between realtime and non-realtime signals, or between multiple pending  
 67785 non-realtime signals, is unspecified.

67786 **RETURN VALUE**

67787 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*  
 67788 function shall return a value of zero. Otherwise, the function shall return a value of -1 and set  
 67789 *errno* to indicate the error.

67790 **ERRORS**

67791 The *sigqueue()* function shall fail if:

67792 [EAGAIN] No resources are available to queue the signal. The process has already  
 67793 queued {SIGQUEUE\_MAX} signals that are still pending at the receiver(s), or  
 67794 a system-wide resource limit has been exceeded.

67795 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

67796 [EPERM] The process does not have appropriate privileges to send the signal to the  
 67797 receiving process.

67798 [ESRCH] The process *pid* does not exist.

**67799 EXAMPLES**

67800 None.

**67801 APPLICATION USAGE**

67802 None.

**67803 RATIONALE**

67804 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another  
67805 process, specifying the application-defined value. This is common practice in realtime  
67806 applications on existing realtime systems. It was felt that specifying another function in the  
67807 *sig...* name space already carved out for signals was preferable to extending the interface to  
67808 *kill()*.

67809 Such a function became necessary when the put/get event function of the message queues was  
67810 removed. It should be noted that the *sigqueue()* function implies reduced performance in a  
67811 security-conscious implementation as the access permissions between the sender and receiver  
67812 have to be checked on each send when the *pid* is resolved into a target process. Such access  
67813 checks were necessary only at message queue open in the previous interface.

67814 The standard developers required that *sigqueue()* have the same semantics with respect to the  
67815 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty  
67816 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the  
67817 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function  
67818 queues a signal to a single process specified by the *pid* argument.

67819 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An  
67820 explicit limit on the number of queued signals that a process could send was introduced. While  
67821 the limit is “per-sender”, this volume of POSIX.1-2024 does not specify that the resources be part  
67822 of the state of the sender. This would require either that the sender be maintained after exit until  
67823 all signals that it had sent to other processes were handled or that all such signals that had not  
67824 yet been acted upon be removed from the queue(s) of the receivers. This volume of  
67825 POSIX.1-2024 does not preclude this behavior, but an implementation that allocated queuing  
67826 resources from a system-wide pool (with per-sender limits) and that leaves queued signals  
67827 pending after the sender exits is also permitted.

**67828 FUTURE DIRECTIONS**

67829 None.

**67830 SEE ALSO**

67831 [Section 2.8.1](#) (on page 528)

67832 XBD [<signal.h>](#)

**67833 CHANGE HISTORY**

67834 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
67835 POSIX Threads Extension.

**67836 Issue 6**

67837 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

67838 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
67839 implementation does not support the Realtime Signals Extension option.

**67840 Issue 7**

67841 The *sigqueue()* function is moved from the Realtime Signals Extension option to the Base.

67842 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0332 [844] is applied.

67843 **NAME**

67844 sigsetjmp — set jump point for a non-local goto

67845 **SYNOPSIS**

```
67846 CX #include <setjmp.h>
67847 int sigsetjmp(sigjmp_buf env, int savemask);
```

67848 **DESCRIPTION**67849 The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:

- 67850 • References to *setjmp()* are equivalent to *sigsetjmp()*.
- 67851 • References to *longjmp()* are equivalent to *siglongjmp()*.
- 67852 • If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask of the calling thread as part of the calling environment.

67854 **RETURN VALUE**

67855 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from  
 67856 a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

67857 **ERRORS**

67858 No errors are defined.

67859 **EXAMPLES**

67860 None.

67861 **APPLICATION USAGE**

67862 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for  
 67863 programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

67864 Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified  
 67865 whether the signal mask is saved.

67866 **RATIONALE**

67867 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to  
 67868 permit implementors to recognize the name in the compiler and not implement an actual  
 67869 function. These same restrictions apply to the *sigsetjmp()* macro.

67870 There are processors that cannot easily support these calls, but this was not considered a  
 67871 sufficient reason to exclude them.

67872 4.2 BSD and 4.3 BSD provided functions named *\_setjmp()* and *\_longjmp()* that, together with  
 67873 *setjmp()* and *longjmp()*, provided the same functionality as *sigsetjmp()* and *siglongjmp()*. On  
 67874 those systems, *setjmp()* and *longjmp()* saved and restored signal masks, while *\_setjmp()* and  
 67875 *\_longjmp()* did not. On System V Release 3 and in corresponding issues of the SVID, *setjmp()*  
 67876 and *longjmp()* were explicitly defined not to save and restore signal masks. In order to permit  
 67877 existing practice in both cases, the relation of *setjmp()* and *longjmp()* to signal masks is not  
 67878 specified, and a new set of functions is defined instead.

67879 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching  
 67880 *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts  
 67881 saved by other threads would be at best a questionable practice and were not considered worthy  
 67882 of standardization.

67883 **FUTURE DIRECTIONS**

67884 None.

67885 **SEE ALSO**67886 *pthread\_sigmask()*, *siglongjmp()*, *signal()*, *sigsuspend()*67887 XBD **<setjmp.h>**67888 **CHANGE HISTORY**

67889 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67890 **Issue 5**

67891 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

67892 **Issue 6**67893 The DESCRIPTION is reworded in terms of *setjmp()*.67894 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an extension over the ISO C standard.67896 **Issue 8**

67897 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

67898 **NAME**

67899 sigsuspend — wait for a signal

67900 **SYNOPSIS**

```
67901 CX #include <signal.h>
67902 int sigsuspend(const sigset_t *sigmask);
```

67903 **DESCRIPTION**

67904 The *sigsuspend()* function shall atomically both replace the current signal mask of the calling  
 67905 thread with the set of signals pointed to by *sigmask* and suspend the thread until delivery of a  
 67906 signal whose action is either to execute a signal-catching function or to terminate the process.  
 67907 This shall not cause any other signals that may have been pending on the process to become  
 67908 pending on the thread.

67909 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to  
 67910 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching  
 67911 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*  
 67912 call.

67913 It is not possible to block signals that cannot be ignored. This is enforced by the system without  
 67914 causing an error to be indicated.

67915 **RETURN VALUE**

67916 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion  
 67917 return value. If a return occurs,  $-1$  shall be returned and *errno* set to indicate the error.

67918 **ERRORS**

67919 The *sigsuspend()* function shall fail if:

67920 [EINTR] A signal is caught by the calling process and control is returned from the  
 67921 signal-catching function.

67922 **EXAMPLES**

67923 None.

67924 **APPLICATION USAGE**

67925 Normally, at the beginning of a critical code section, a specified set of signals is blocked using  
 67926 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait  
 67927 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was  
 67928 returned by the *sigprocmask()* call.

67929 **RATIONALE**

67930 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers  
 67931 can install an additional cancellation handler which resets the signal mask to the expected value.

```
67932 void cleanup(void *arg)
67933 {
67934     sigset_t *ss = (sigset_t *) arg;
67935     pthread_sigmask(SIG_SETMASK, ss, NULL);
67936 }
67937 int call_sigsuspend(const sigset_t *mask)
67938 {
67939     sigset_t oldmask;
67940     int result;
67941     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
67942     pthread_cleanup_push(cleanup, &oldmask);
```

```
67943         result = sigsuspend(mask);
67944         pthread_cleanup_pop(0);
67945         return result;
67946     }
```

**67947 FUTURE DIRECTIONS**

67948 None.

**67949 SEE ALSO**

67950 [Section 2.4](#) (on page 513), [pause\(\)](#), [sigaction\(\)](#), [sigaddset\(\)](#), [sigdelset\(\)](#), [sigemptyset\(\)](#), [sigfillset\(\)](#)

67951 XBD [<signal.h>](#)

**67952 CHANGE HISTORY**

67953 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

**67954 Issue 5**

67955 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

**67956 Issue 6**

67957 The text in the RETURN VALUE section has been changed from “suspends process execution”  
67958 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

67959 Text in the APPLICATION USAGE section has been replaced.

67960 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an  
67961 extension over the ISO C standard.

**67962 Issue 7**

67963 SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

**67964 Issue 8**

67965 Austin Group Defect 1201 is applied, clarifying the atomicity requirements for [sigsuspend\(\)](#).

67966 Austin Group Defect 1223 is applied, changing the example code in the RATIONALE.



67967 **NAME**

67968 sigtimedwait, sigwaitinfo — wait for queued signals

67969 **SYNOPSIS**

```
67970 CX #include <signal.h>
67971 int sigtimedwait(const sigset_t *restrict set,
67972 siginfo_t *restrict info,
67973 const struct timespec *restrict timeout);
67974 int sigwaitinfo(const sigset_t *restrict set,
67975 siginfo_t *restrict info);
```

67976 **DESCRIPTION**

67977 The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals  
67978 specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the  
67979 **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-  
67980 valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall return  
67981 immediately with an error. If *timeout* is the null pointer, the behavior is unspecified. The  
67982 CLOCK\_MONOTONIC clock shall be used to measure the time interval specified by the *timeout*  
67983 argument.

67984 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of  
67985 multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the  
67986 lowest numbered one. The selection order between realtime and non-realtime signals, or  
67987 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at  
67988 the time of the call, the calling thread shall be suspended until one or more signals in *set* become  
67989 pending or until it is interrupted by an unblocked, caught signal.

67990 The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function, except that the return  
67991 value and the error reporting method are different (see RETURN VALUE), and that if the *info*  
67992 argument is non-NULL, the selected signal number shall be stored in the *si\_signo* member, and  
67993 the cause of the signal shall be stored in the *si\_code* member. If any value is queued to the  
67994 selected signal, the first such queued value shall be dequeued and, if the *info* argument is non-  
67995 NULL, the value shall be stored in the *si\_value* member of *info*. The system resource used to  
67996 queue the signal shall be released and returned to the system for other use. If no value is  
67997 queued, the content of the *si\_value* member is undefined. If no further signals are queued for the  
67998 selected signal, the pending indication for that signal shall be reset.

67999 **RETURN VALUE**

68000 Upon successful completion (that is, one of the signals specified by *set* is pending or is  
68001 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,  
68002 the function shall return a value of  $-1$  and set *errno* to indicate the error.

68003 **ERRORS**

68004 The *sigtimedwait()* function shall fail if:

68005 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.

68006 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

68007 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be  
68008 documented in system documentation whether this error causes these  
68009 functions to fail.

68010 The *sigtimedwait()* function may also fail if:  
 68011 [EINVAL] The *timeout* argument specified a *tv\_nsec* value less than zero or greater than  
 68012 or equal to 1 000 million.  
 68013 An implementation should only check for this error if no signal is pending in *set* and it is  
 68014 necessary to wait.

#### 68015 EXAMPLES

68016 None.

#### 68017 APPLICATION USAGE

68018 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application developers  
 68019 should note that this is inconsistent with other functions such as *pthread\_cond\_timedwait()* that  
 68020 return [ETIMEDOUT].

68021 Note that in order to ensure that generated signals are queued and signal values passed to  
 68022 *sigqueue()* are available in *si\_value*, applications which use *sigwaitinfo()* or *sigtimedwait()* need to  
 68023 set the SA\_SIGINFO flag for each signal in the set (see Section 2.4, on page 513). This means  
 68024 setting each signal to be handled by a three-argument signal-catching function, even if the  
 68025 handler will never be called. It is not possible (portably) to set a signal handler to SIG\_DFL  
 68026 while setting the SA\_SIGINFO flag, because assigning to the *sa\_handler* member of **struct**  
 68027 **sigaction** instead of the *sa\_sigaction* member would result in undefined behavior, and SIG\_DFL  
 68028 need not be assignment-compatible with *sa\_sigaction*. Even if an assignment of SIG\_DFL to  
 68029 *sa\_sigaction* is accepted by the compiler, the implementation need not treat this value as special—  
 68030 it could just be taken as the address of a signal-catching function.

#### 68031 RATIONALE

68032 Existing programming practice on realtime systems uses the ability to pause waiting for a  
 68033 selected set of events and handle the first event that occurs in-line instead of in a signal-handling  
 68034 function. This allows applications to be written in an event-directed style similar to a state  
 68035 machine. This style of programming is useful for largescale transaction processing in which the  
 68036 overall throughput of an application and the ability to clearly track states are more important  
 68037 than the ability to minimize the response time of individual event handling.

68038 It is possible to construct a signal-waiting macro function out of the realtime signal function  
 68039 mechanism defined in this volume of POSIX.1-2024. However, such a macro has to include the  
 68040 definition of a generalized handler for all signals to be waited on. A significant portion of the  
 68041 overhead of handler processing can be avoided if the signal-waiting function is provided by the  
 68042 kernel. This volume of POSIX.1-2024 therefore provides two signal-waiting functions—one that  
 68043 waits indefinitely and one with a timeout—as part of the overall realtime signal function  
 68044 specification.

68045 The specification of a function with a timeout allows an application to be written that can be  
 68046 broken out of a wait after a set period of time if no event has occurred. It was argued that setting  
 68047 a timer event before the wait and recognizing the timer event in the wait would also implement  
 68048 the same functionality, but at a lower performance level. Because of the performance  
 68049 degradation associated with the user-level specification of a timer event and the subsequent  
 68050 cancellation of that timer event after the wait completes for a valid event, and the complexity  
 68051 associated with handling potential race conditions associated with the user-level method, the  
 68052 separate function has been included.

68053 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*  
 68054 function defined by this volume of POSIX.1-2024. The only difference is that *sigwaitinfo()* returns  
 68055 the queued signal value in the *value* argument. The return of the queued value is required so that  
 68056 applications can differentiate between multiple events queued to the same signal number.

68057 The two distinct functions are being maintained because some implementations may choose to  
 68058 implement the POSIX Threads Extension functions and not implement the queued signals  
 68059 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*  
 68060 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a  
 68061 macro on *sigwaitinfo()*.

68062 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns  
 68063 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed  
 68064 wait, and immediate return, and concerns regarding consistency with other functions where the  
 68065 conditional and timed waits were separate functions from the pure blocking function. The  
 68066 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a macro  
 68067 with a null pointer for *timeout*.

68068 The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-  
 68069 generated signals. One important question was how many threads that are suspended in a call  
 68070 to a *sigwait()* function for a signal should return from the call when the signal is sent. Four  
 68071 choices were considered:

- 68072 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 68073 2. One or more threads return.
- 68074 3. All waiting threads return.
- 68075 4. Exactly one thread returns.

68076 Prohibiting multiple calls to *sigwait()* for the same signal was felt to be overly restrictive. The  
 68077 “one or more” behavior made implementation of conforming packages easy at the expense of  
 68078 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait()* in  
 68079 application code in order to achieve predictable behavior. There was concern that the “all  
 68080 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU  
 68081 resources by replicating the signals in the general case. Furthermore, no convincing examples  
 68082 could be presented that delivery to all was either simpler or more powerful than delivery to one.

68083 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*  
 68084 function for a signal should return when that signal occurs. This is not an onerous restriction as:

- 68085 • A multi-way signal wait can be built from the single-way wait.
- 68086 • Signals should only be handled by application-level code, as library routines cannot guess  
 68087 what the application wants to do with signals generated for the entire process.
- 68088 • Applications can thus arrange for a single thread to wait for any given signal and call any  
 68089 needed routines upon its arrival.

68090 In an application that is using signals for interprocess communication, signal processing is  
 68091 typically done in one place. Alternatively, if the signal is being caught so that process cleanup  
 68092 can be done, the signal handler thread can call separate process cleanup routines for each  
 68093 portion of the application. Since the application main line started each portion of the application,  
 68094 it is at the right abstraction level to tell each portion of the application to clean up.

68095 Certainly, there exist programming styles where it is logical to consider waiting for a single  
 68096 signal in multiple threads. A simple *sigwait\_multiple()* routine can be constructed to achieve this  
 68097 goal. A possible implementation would be to have each *sigwait\_multiple()* caller registered as  
 68098 having expressed interest in a set of signals. The caller then waits on a thread-specific condition  
 68099 variable. A single server thread calls a *sigwait()* function on the union of all registered signals.  
 68100 When the *sigwait()* function returns, the appropriate state is set and condition variables are  
 68101 broadcast. New *sigwait\_multiple()* callers may cause the pending *sigwait()* call to be canceled

68102 and reissued in order to update the set of signals being waited for.

68103 **FUTURE DIRECTIONS**

68104 None.

68105 **SEE ALSO**

68106 [Section 2.4](#) (on page 513), [Section 2.8.1](#) (on page 528), [pause\(\)](#), [pthread\\_sigmask\(\)](#), [sigaction\(\)](#),  
68107 [sigpending\(\)](#), [sigsuspend\(\)](#), [sigwait\(\)](#)

68108 XBD [<signal.h>](#), [<time.h>](#)

68109 **CHANGE HISTORY**

68110 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
68111 POSIX Threads Extension.

68112 **Issue 6**

68113 These functions are marked as part of the Realtime Signals Extension option.

68114 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the `sigwaitinfo()` function  
68115 has been corrected so that the second argument is of type `siginfo_t *`.

68116 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
68117 implementation does not support the Realtime Signals Extension option.

68118 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the  
68119 CLOCK\_MONOTONIC clock, if supported, is used to measure timeout intervals.

68120 The `restrict` keyword is added to the `sigtimedwait()` and `sigwaitinfo()` prototypes for alignment  
68121 with the ISO/IEC 9899:1999 standard.

68122 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/130 is applied, restoring wording in the  
68123 RETURN VALUE section to that in the original base document (“An implementation should  
68124 only check for this error if no signal is pending in *set* and it is necessary to wait”).

68125 **Issue 7**

68126 The `sigtimedwait()` and `sigwaitinfo()` functions are moved from the Realtime Signals Extension  
68127 option to the Base.

68128 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0583 [392] is applied.

68129 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0333 [815] is applied.

68130 **Issue 8**

68131 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

68132 **NAME**

68133 sigwait — wait for queued signals

68134 **SYNOPSIS**

```
68135 CX #include <signal.h>
68136 int sigwait(const sigset_t *restrict set, int *restrict sig);
```

68137 **DESCRIPTION**

68138 The *sigwait()* function shall select a pending signal from *set*, atomically clear it from the system's  
 68139 set of pending signals, and return that signal number in the location referenced by *sig*. If prior to  
 68140 the call to *sigwait()* there are multiple pending instances of a single signal number, it is  
 68141 implementation-defined whether upon successful return there are any remaining pending  
 68142 signals for that signal number. If the implementation supports queued signals and there are  
 68143 multiple signals queued for the signal number selected, the first such queued signal shall cause a  
 68144 return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the  
 68145 time of the call, the thread shall be suspended until one or more becomes pending. The signals  
 68146 defined by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior  
 68147 is undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

68148 If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these  
 68149 threads shall return from *sigwait()* with the signal number. If more than a single thread is  
 68150 blocked in *sigwait()* for a signal when that signal is generated for the process, it is unspecified  
 68151 which of the waiting threads returns from *sigwait()*. If the signal is generated for a specific  
 68152 thread, as by *pthread\_kill()*, only that thread shall return.

68153 Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it  
 68154 shall be the lowest numbered one. The selection order between realtime and non-realtime  
 68155 signals, or between multiple pending non-realtime signals, is unspecified.

68156 **RETURN VALUE**

68157 Upon successful completion, *sigwait()* shall store the signal number of the received signal at the  
 68158 location referenced by *sig* and return zero. Otherwise, an error number shall be returned to  
 68159 indicate the error.

68160 **ERRORS**

68161 The *sigwait()* function may fail if:

68162 [EINVAL] The *set* argument contains an invalid or unsupported signal number.

68163 **EXAMPLES**

68164 None.

68165 **APPLICATION USAGE**

68166 None.

68167 **RATIONALE**

68168 To provide a convenient way for a thread to wait for a signal, this volume of POSIX.1-2024  
 68169 provides the *sigwait()* function. For most cases where a thread has to wait for a signal, the  
 68170 *sigwait()* function should be quite convenient, efficient, and adequate.

68171 However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that  
 68172 could be used by threads. After some consideration, threads were allowed to use semaphores  
 68173 and *sem\_post()* was defined to be async-signal-safe.

68174 In summary, when it is necessary for code run in response to an asynchronous signal to notify a  
 68175 thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation  
 68176 provides semaphores, they also can be used, either following *sigwait()* or from within a signal

68177 handling routine previously registered with *sigaction()*.

68178 **FUTURE DIRECTIONS**

68179 None.

68180 **SEE ALSO**

68181 [Section 2.4](#) (on page 513), [Section 2.8.1](#) (on page 528), [pause\(\)](#), [pthread\\_sigmask\(\)](#), [sigaction\(\)](#),  
68182 [sigpending\(\)](#), [sigsuspend\(\)](#), [sigtimedwait\(\)](#)

68183 XBD [<signal.h>](#), [<time.h>](#)

68184 **CHANGE HISTORY**

68185 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the  
68186 POSIX Threads Extension.

68187 **Issue 6**

68188 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the  
68189 ISO/IEC 9899:1999 standard.

68190 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/131 is applied, updating the  
68191 DESCRIPTION to state that if more than a single thread is blocked in *sigwait()*, it is unspecified  
68192 which of the waiting threads returns, and that if a signal is generated for a specific thread only  
68193 that thread shall return.

68194 **Issue 7**

68195 Functionality relating to the Realtime Signals Extension option is moved to the Base.

68196 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0584 [76] is applied.

68197 **NAME**

68198 sigwaitinfo — wait for queued signals

68199 **SYNOPSIS**68200 

```
#include <signal.h>
```

68201 

```
int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);
```

68202 **DESCRIPTION**68203 Refer to *sigtimedwait()*.

68204 **NAME**

68205 sin, sinf, sinl — sine function

68206 **SYNOPSIS**

```
68207 #include <math.h>
68208 double sin(double x);
68209 float sinf(float x);
68210 long double sinl(long double x);
```

68211 **DESCRIPTION**

68212 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 68213 conflict between the requirements described here and the ISO C standard is unintentional. This  
 68214 volume of POSIX.1-2024 defers to the ISO C standard.

68215 These functions shall compute the sine of their argument  $x$ , measured in radians.

68216 An application wishing to check for error situations should set *errno* to zero and call  
 68217 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 68218 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 68219 zero, an error has occurred.

68220 **RETURN VALUE**

68221 Upon successful completion, these functions shall return the sine of  $x$ .

68222 MX If  $x$  is NaN, a NaN shall be returned.

68223 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

68224 If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and a NaN shall be returned.

68225 MXX If  $x$  is subnormal,  $x$  should be returned.

68226 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *sin()*, *sinf()*, and *sinl()* shall  
 68227 return an implementation-defined value no greater in magnitude than DBL\_MIN, FLT\_MIN,  
 68228 and LDBL\_MIN, respectively.

68229 **ERRORS**

68230 These functions shall fail if:

68231 MX **Domain Error** The  $x$  argument is  $\pm\text{Inf}$ .

68232 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 68233 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 68234 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 68235 shall be raised.

68236 These functions may fail if:

68237 MX **Range Error** The value of  $x$  is subnormal

68238 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 68239 then *errno* shall be set to [ERANGE]. If the integer expression  
 68240 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 68241 floating-point exception shall be raised.



68242 **EXAMPLES**68243 **Taking the Sine of a 45-Degree Angle**

```

68244 #include <math.h>
68245 ...
68246 double radians = 45.0 * M_PI / 180;
68247 double result;
68248 ...
68249 result = sin(radians);

```

68250 **APPLICATION USAGE**

68251 These functions may lose accuracy when their argument is near a multiple of  $\pi$  or is far from 0.0.

68252 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
68253 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

68254 **RATIONALE**

68255 None.

68256 **FUTURE DIRECTIONS**

68257 None.

68258 **SEE ALSO**

68259 [asin\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

68260 XBD Section 4.23 (on page 109), [<math.h>](#)

68261 **CHANGE HISTORY**

68262 First released in Issue 1. Derived from Issue 1 of the SVID.

68263 **Issue 5**

68264 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
68265 in previous issues.

68266 **Issue 6**

68267 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

68268 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
68269 revised to align with the ISO/IEC 9899:1999 standard.

68270 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
68271 marked.

68272 **Issue 7**

68273 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0585 [68] and XSH/TC1-2008/0586  
68274 [320] are applied.

68275 **Issue 8**

68276 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is  
68277 subnormal to avoid the need for two shading changes.

68278 **NAME**

68279           sinh, sinhf, sinh1 — hyperbolic sine functions

68280 **SYNOPSIS**

```
68281       #include <math.h>
68282       double sinh(double x);
68283       float  sinhf(float x);
68284       long double sinh1(long double x);
```

68285 **DESCRIPTION**

68286 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
68287 conflict between the requirements described here and the ISO C standard is unintentional. This  
68288 volume of POSIX.1-2024 defers to the ISO C standard.

68289       These functions shall compute the hyperbolic sine of their argument *x*.

68290       An application wishing to check for error situations should set *errno* to zero and call  
68291 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
68292 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
68293 zero, an error has occurred.

68294 **RETURN VALUE**

68295       Upon successful completion, these functions shall return the hyperbolic sine of *x*.

68296       If the result would cause an overflow, a range error shall occur and  $\pm$ HUGE\_VAL,  
68297  $\pm$ HUGE\_VALF, and  $\pm$ HUGE\_VALL (with the same sign as *x*) shall be returned as appropriate for  
68298 the type of the function.

68299 MX       If *x* is NaN, a NaN shall be returned.

68300       If *x* is  $\pm 0$  or  $\pm$ Inf, *x* shall be returned.

68301 MXX     If *x* is subnormal, *x* should be returned.

68302 MX       If *x* is subnormal, a range error may occur and, if *x* is not returned, *sinh*(*x*), *sinhf*(*x*), and *sinh1*(*x*)  
68303 shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
68304 FLT\_MIN, and LDBL\_MIN, respectively.

68305 **ERRORS**

68306       These functions shall fail if:

68307       Range Error       The result would cause an overflow.

68308               If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
68309 then *errno* shall be set to [ERANGE]. If the integer expression  
68310 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
68311 floating-point exception shall be raised.

68312       These functions may fail if:

68313 MX       Range Error       The value *x* is subnormal.

68314               If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
68315 then *errno* shall be set to [ERANGE]. If the integer expression  
68316 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
68317 floating-point exception shall be raised.

**68318 EXAMPLES**

68319 None.

**68320 APPLICATION USAGE**

68321 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
68322 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

**68323 RATIONALE**

68324 None.

**68325 FUTURE DIRECTIONS**

68326 None.

**68327 SEE ALSO**

68328 [asinh\(\)](#), [cosh\(\)](#), [fclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [tanh\(\)](#)

68329 XBD Section 4.23 (on page 109), [<math.h>](#)

**68330 CHANGE HISTORY**

68331 First released in Issue 1. Derived from Issue 1 of the SVID.

**68332 Issue 5**

68333 The DESCRIPTION is updated to indicate how an application should check for an error. This  
68334 text was previously published in the APPLICATION USAGE section.

**68335 Issue 6**

68336 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

68337 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
68338 revised to align with the ISO/IEC 9899:1999 standard.

68339 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
68340 marked.

**68341 Issue 7**

68342 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0587 [68] is applied.

**68343 Issue 8**

68344 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is  
68345 subnormal to avoid the need for two shading changes.

68346 **NAME**

68347       sinl — sine function

68348 **SYNOPSIS**

68349       #include <math.h>

68350       long double sinl(long double *x*);

68351 **DESCRIPTION**

68352       Refer to *sin()*.

68353 **NAME**

68354 sleep — suspend execution for an interval of time

68355 **SYNOPSIS**

68356 #include &lt;unistd.h&gt;

68357 unsigned sleep(unsigned *seconds*);68358 **DESCRIPTION**

68359 The *sleep()* function shall cause the calling thread to be suspended from execution until either  
 68360 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is  
 68361 delivered to the calling thread and its action is to invoke a signal-catching function or to  
 68362 terminate the process. The suspension time may be longer than requested due to the scheduling  
 68363 of other activity by the system.

68364 In single-threaded programs, *sleep()* may make use of SIGALRM. In multi-threaded programs,  
 68365 *sleep()* shall not make use of SIGALRM and the remainder of this DESCRIPTION does not apply.

68366 If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the  
 68367 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()*  
 68368 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also  
 68369 unspecified whether it remains pending after *sleep()* returns or it is discarded.

68370 If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a  
 68371 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from  
 68372 delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

68373 If a signal-catching function interrupts *sleep()* and examines or changes either the time a  
 68374 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or  
 68375 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

68376 If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an  
 68377 environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and  
 68378 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also  
 68379 unspecified whether the SIGALRM signal is blocked, unless the signal mask of the process is  
 68380 restored as part of the environment.

68381 **RETURN VALUE**

68382 If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()*  
 68383 returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested  
 68384 time minus the time actually slept) in seconds.

68385 **ERRORS**

68386 No errors are defined.

68387 **EXAMPLES**

68388 None.

68389 **APPLICATION USAGE**

68390 None.

68391 **RATIONALE**

68392 There are two general approaches to the implementation of the *sleep()* function. One is to use the  
 68393 *alarm()* function to schedule a SIGALRM signal and then suspend the calling thread waiting for  
 68394 that signal. The other is to implement an independent facility. This volume of POSIX.1-2024  
 68395 permits either approach in single-threaded programs, but the simple alarm/suspend  
 68396 implementation is not appropriate for multi-threaded programs.

68397 In order to comply with the requirement that no primitive shall change a process attribute unless

68398 explicitly described by this volume of POSIX.1-2024, an implementation using SIGALRM must  
 68399 carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the action  
 68400 previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has  
 68401 been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened to  
 68402 that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-  
 68403 catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()*  
 68404 would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The  
 68405 action and blocking for SIGALRM must be saved and restored.

68406 Historical implementations often implement the SIGALRM-based version using *alarm()* and  
 68407 *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*.  
 68408 Another such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid  
 68409 that window. That implementation introduces a different problem: when the SIGALRM signal  
 68410 interrupts a signal-catching function installed by the user to catch a different signal, the  
 68411 *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*,  
 68412 *alarm()*, and *sigsuspend()* can avoid these problems.

68413 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,  
 68414 differences between the two types of implementations. These are the cases mentioned in this  
 68415 volume of POSIX.1-2024 where some other activity relating to SIGALRM takes place, and the  
 68416 results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of  
 68417 concern to most applications.

68418 See also the discussion of the term *realtime* in *alarm()*.

68419 Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early  
 68420 under *alarm()* applies to *sleep()* as well.

68421 Application developers should note that the type of the argument *seconds* and the return value of  
 68422 *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application  
 68423 cannot pass a value greater than the minimum guaranteed value for {UINT\_MAX}, which the  
 68424 ISO C standard sets as 65 535, and any application passing a larger value is restricting its  
 68425 portability. A different type was considered, but historical implementations, including those  
 68426 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

68427 Scheduling delays may cause the process to return from the *sleep()* function significantly after  
 68428 the requested time. In such cases, the return value should be set to zero, since the formula  
 68429 (requested time minus the time actually spent) yields a negative number and *sleep()* returns an  
 68430 **unsigned**.

#### 68431 FUTURE DIRECTIONS

68432 A future version of this standard may require that *sleep()* does not make use of SIGALRM in all  
 68433 programs, not just multi-threaded programs.

#### 68434 SEE ALSO

68435 *alarm()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*, *timer\_create()*

68436 XBD <**unistd.h**>

#### 68437 CHANGE HISTORY

68438 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 68439 Issue 5

68440 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

68441 **Issue 6**

68442 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/132 is applied, making a correction in the  
68443 RATIONALE section.

68444 **Issue 7**

68445 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0334 [625] is applied.

68446 **Issue 8**

68447 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

68448 **NAME**

68449            snprintf — print formatted output

68450 **SYNOPSIS**

68451            #include <stdio.h>

68452            int snprintf(char \*restrict *s*, size\_t *n*,  
68453                const char \*restrict *format*, ...);

68454 **DESCRIPTION**

68455            Refer to *fprintf()*.



68456 **NAME**

68457 socketmark — determine whether a socket is at the out-of-band mark

68458 **SYNOPSIS**

68459 #include &lt;sys/socket.h&gt;

68460 int socketmark(int s);

68461 **DESCRIPTION**

68462 The *socketmark()* function shall determine whether the socket specified by the descriptor *s* is at  
 68463 the out-of-band data mark (see [Section 2.10.12](#), on page 552). If the protocol for the socket  
 68464 supports out-of-band data by marking the stream with an out-of-band data mark, the  
 68465 *socketmark()* function shall return 1 when all data preceding the mark has been read and the out-  
 68466 of-band data mark is the first element in the receive queue. The *socketmark()* function shall not  
 68467 remove the mark from the stream.

68468 **RETURN VALUE**

68469 Upon successful completion, the *socketmark()* function shall return a value indicating whether  
 68470 the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data  
 68471 preceding the mark has been read, the return value shall be 1; if there is no mark, or if data  
 68472 precedes the mark in the receive queue, the *socketmark()* function shall return 0. Otherwise, it  
 68473 shall return a value of -1 and set *errno* to indicate the error.

68474 **ERRORS**68475 The *socketmark()* function shall fail if:68476 [EBADF] The *s* argument is not a valid file descriptor.68477 [ENOTTY] The file associated with the *s* argument is not a socket.68478 **EXAMPLES**

68479 None.

68480 **APPLICATION USAGE**

68481 The use of this function between receive operations allows an application to determine which  
 68482 received data precedes the out-of-band data and which follows the out-of-band data.

68483 There is an inherent race condition in the use of this function. On an empty receive queue, the  
 68484 current read of the location might well be at the “mark”, but the system has no way of knowing  
 68485 that the next data segment that will arrive from the network will carry the mark, and  
 68486 *socketmark()* will return false, and the next read operation will silently consume the mark.

68487 Hence, this function can only be used reliably when the application already knows that the out-  
 68488 of-band data has been seen by the system or that it is known that there is data waiting to be read  
 68489 at the socket (via SIGURG or *select()*). See [Section 2.10.11](#) (on page 552), [Section 2.10.12](#) (on page  
 68490 552), [Section 2.10.14](#) (on page 553), and *pselect()* for details.

68491 **RATIONALE**

68492 The *socketmark()* function replaces the historical SIOCATMARK command to *ioctl()* which  
 68493 implemented the same functionality on many implementations. Using a wrapper function  
 68494 follows the adopted conventions to avoid specifying commands to the *ioctl()* function. The  
 68495 *socketmark()* function could be implemented as follows:

```
68496 #include <sys/ioctl.h>
68497 int socketmark(int s)
68498 {
68499     int val;
68500     if (ioctl(s, SIOCATMARK, &val) == -1)
```

```
68501         return (-1);
68502     return (val);
68503 }
```

68504 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of  
68505 SIOCATMARK.

68506 **FUTURE DIRECTIONS**

68507 None.

68508 **SEE ALSO**

68509 [Section 2.10.12](#) (on page 552), [pselect\(\)](#), [recv\(\)](#), [recvmsg\(\)](#)

68510 XBD [<sys/socket.h>](#)

68511 **CHANGE HISTORY**

68512 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

68513 **Issue 7**

68514 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

68515 **Issue 8**

68516 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

68517 **NAME**

68518 socket — create an endpoint for communication

68519 **SYNOPSIS**

68520 #include &lt;sys/socket.h&gt;

68521 int socket(int *domain*, int *type*, int *protocol*);68522 **DESCRIPTION**

68523 The *socket()* function shall create an unbound socket in a communications domain, and return a  
 68524 file descriptor that can be used in later function calls that operate on sockets. The file descriptor  
 68525 shall be allocated as described in [Section 2.6](#) (on page 525).

68526 The *socket()* function takes the following arguments:

68527 *domain* Specifies the communications domain in which a socket is to be created.

68528 *type* Specifies the type of socket to be created.

68529 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*  
 68530 of 0 causes *socket()* to use an unspecified default protocol appropriate for the  
 68531 requested socket type.

68532 The *domain* argument specifies the address family used in the communications domain. The  
 68533 address families supported by the system are implementation-defined.

68534 Symbolic constants that can be used for the domain argument are defined in the <sys/socket.h>  
 68535 header.

68536 The *type* argument specifies the socket type, which determines the semantics of communication  
 68537 over the socket. The following socket types are defined; implementations may specify additional  
 68538 socket types:

68539 SOCK\_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte streams,  
 68540 and may provide a transmission mechanism for out-of-band data.

68541 SOCK\_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages of  
 68542 fixed maximum length.

## 68543 SOCK\_SEQPACKET

68544 Provides sequenced, reliable, bidirectional, connection-mode transmission  
 68545 paths for records. A record can be sent using one or more output operations  
 68546 and received using one or more input operations, but a single operation never  
 68547 transfers part of more than one record. Record boundaries are visible to the  
 68548 receiver via the MSG\_EOR flag.

68549 Additionally, the *type* argument can contain the bitwise-inclusive OR of flags from the following  
 68550 list:

68551 SOCK\_CLOEXEC Atomically set the FD\_CLOEXEC flag on the new file descriptor.

68552 SOCK\_CLOFORK Atomically set the FD\_CLOFORK flag on the new file descriptor.

68553 SOCK\_NONBLOCK Set the O\_NONBLOCK file status flag on the new file description.

68554 Implementations may define additional flags.

68555 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
 68556 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
 68557 be used. The protocols supported by the system are implementation-defined.

68558 The process may need to have appropriate privileges to use the *socket()* function or to create

68559 some sockets.

#### 68560 RETURN VALUE

68561 Upon successful completion, *socket()* shall return a non-negative integer, the socket file  
68562 descriptor. Otherwise, a value of  $-1$  shall be returned and *errno* set to indicate the error.

#### 68563 ERRORS

68564 The *socket()* function shall fail if:

68565 [EAFNOSUPPORT]

68566 The implementation does not support the specified address family.

68567 [EMFILE] All file descriptors available to the process are currently open.

68568 [ENFILE] No more file descriptors are available for the system.

68569 [EPROTONOSUPPORT]

68570 The value of *protocol* is non-zero and either the protocol is not supported by  
68571 the address family or the protocol is not supported by the implementation.

68572 [EPROTOTYPE] The value of *protocol* is non-zero and the socket type is not supported by the  
68573 protocol.

68574 OB [ESOCKTNOSUPPORT] or [EPROTONOSUPPORT] or [EPROTOTYPE]

68575 The socket type is not supported by the address family, or the socket type is  
68576 not supported by the implementation.

68577 The *socket()* function may fail if:

68578 [EACCES] The process does not have appropriate privileges.

68579 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

68580 [ENOMEM] Insufficient memory was available to fulfill the request.

#### 68581 EXAMPLES

68582 None.

#### 68583 APPLICATION USAGE

68584 The documentation for specific address families specifies which protocols each address family  
68585 supports. The documentation for specific protocols specifies which socket types each protocol  
68586 supports.

68587 The application can determine whether an address family is supported by trying to create a  
68588 socket with *domain* set to the protocol in question.

#### 68589 RATIONALE

68590 The use of the SOCK\_CLOEXEC and SOCK\_CLOFORK flags in the *type* argument of *socket()* is  
68591 necessary to avoid a data race in multi-threaded applications. Without SOCK\_CLOFORK, a file  
68592 descriptor is leaked into a child process created by one thread in the window between another  
68593 thread calling *socket()* and using *fcntl()* to set the FD\_CLOFORK flag. Without  
68594 SOCK\_CLOEXEC, a file descriptor intentionally inherited by child processes is similarly leaked  
68595 into an executed program if FD\_CLOEXEC is not set atomically. The SOCK\_NONBLOCK flag is  
68596 for convenience in avoiding additional *fcntl()* calls.

68597 Historically the standard did not specify the *errno* value to be used when the socket type is not  
68598 supported, and there were differences between implementations. Some reused the existing  
68599 standard [EPROTONOSUPPORT] or [EPROTOTYPE] values while others set *errno* to a (then)  
68600 non-standard value of [ESOCKTNOSUPPORT]. All three values are permitted in this version of  
68601 the standard, but the use of [EPROTONOSUPPORT] or [EPROTOTYPE] is considered to be  
68602 misleading when no protocol is specified (that is, the value of *protocol* is zero) and consequently

68603 those alternatives have been marked obsolescent. If *protocol* is non-zero, since there is no  
68604 precedence between error conditions, all three values will still be permitted even after the  
68605 obsolescent alternatives for the [ESOCKTNOSUPPORT] condition have been removed.

#### 68606 FUTURE DIRECTIONS

68607 A future version of this standard may disallow setting *errno* to [EPROTONOSUPPORT] or  
68608 [EPROTOTYPE] when the socket type is not supported and *protocol* is zero.

#### 68609 SEE ALSO

68610 Section 2.6 (on page 525), *accept()*, *bind()*, *close()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*,  
68611 *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*

68612 XBD <[netinet/in.h](#)>, <[sys/socket.h](#)>

#### 68613 CHANGE HISTORY

68614 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

#### 68615 Issue 7

68616 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0335 [835] is applied.

#### 68617 Issue 8

68618 Austin Group Defects 411 and 1318 are applied, adding SOCK\_CLOEXEC, SOCK\_CLOFORK,  
68619 and SOCK\_NONBLOCK.

68620 Austin Group Defect 1067 is applied, adding the [ESOCKTNOSUPPORT] error and changing the  
68621 [EPROTONOSUPPORT] and [EPROTOTYPE] errors.

68622 Austin Group Defect 1475 is applied, adding *close()* to the SEE ALSO section.

68623 **NAME**

68624 socketpair — create a pair of connected sockets

68625 **SYNOPSIS**

```
68626 #include <sys/socket.h>
68627 int socketpair(int domain, int type, int protocol,
68628               int socket_vector[2]);
```

68629 **DESCRIPTION**

68630 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,  
 68631 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two  
 68632 sockets shall be identical. The file descriptors used in referencing the created sockets shall be  
 68633 returned in *socket\_vector*[0] and *socket\_vector*[1]. The file descriptors shall be allocated as  
 68634 described in [Section 2.6](#) (on page 525).

68635 The *socketpair()* function takes the following arguments:

68636	<i>domain</i>	Specifies the communications domain in which the sockets are to be created.
68637	<i>type</i>	Specifies the type of sockets to be created.
68638	<i>protocol</i>	Specifies a particular protocol to be used with the sockets. Specifying a 68639 <i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol 68640 appropriate for the requested socket type.
68641	<i>socket_vector</i>	Specifies a 2-integer array to hold the file descriptors of the created socket pair.

68642 The *type* argument specifies the socket type, which determines the semantics of communications  
 68643 over the socket. The following socket types are defined; implementations may specify additional  
 68644 socket types:

68645	SOCK_STREAM	Provides sequenced, reliable, bidirectional, connection-mode byte 68646 streams, and may provide a transmission mechanism for out-of-band 68647 data.
68648	SOCK_DGRAM	Provides datagrams, which are connectionless-mode, unreliable messages 68649 of fixed maximum length.
68650	SOCK_SEQPACKET	Provides sequenced, reliable, bidirectional, connection-mode transmission 68651 paths for records. A record can be sent using one or more output 68652 operations and received using one or more input operations, but a single 68653 operation never transfers part of more than one record. Record 68654 boundaries are visible to the receiver via the MSG_EOR flag.

68655 Additionally, the *type* argument can contain the bitwise-inclusive OR of flags from the following  
 68656 list:

68657	SOCK_CLOEXEC	Atomically set the FD_CLOEXEC flag on the new file descriptors.
68658	SOCK_CLOFORK	Atomically set the FD_CLOFORK flag on the new file descriptors.
68659	SOCK_NONBLOCK	Set the O_NONBLOCK file status flag on the new file descriptions.

68660 Implementations may define additional flags.

68661 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address  
 68662 family. If the *protocol* argument is zero, the default protocol for this address family and type shall  
 68663 be used. The protocols supported by the system are implementation-defined.

68664 The process may need to have appropriate privileges to use the *socketpair()* function or to create  
 68665 some sockets.

68666 **RETURN VALUE**

68667 Upon successful completion, this function shall return 0; otherwise, `-1` shall be returned and  
 68668 *errno* set to indicate the error, no file descriptors shall be allocated, and the contents of  
 68669 *socket\_vector* shall be left unmodified.

68670 **ERRORS**

68671 The *socketpair()* function shall fail if:

68672 [EAFNOSUPPORT]

68673 The implementation does not support the specified address family.

68674 [EMFILE]

68675 All, or all but one, of the file descriptors available to the process are currently open.

68676 [ENFILE]

No more file descriptors are available for the system.

68677 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.

68678 [EPROTONOSUPPORT]

68679 The protocol is not supported by the address family, or the protocol is not  
 68680 supported by the implementation.

68681 [EPROTOTYPE] The socket type is not supported by the protocol.

68682 The *socketpair()* function may fail if:

68683 [EACCES]

The process does not have appropriate privileges.

68684 [ENOBUFS]

Insufficient resources were available in the system to perform the operation.

68685 [ENOMEM]

Insufficient memory was available to fulfill the request.

68686 **EXAMPLES**

68687 None.

68688 **APPLICATION USAGE**

68689 The documentation for specific address families specifies which protocols each address family  
 68690 supports. The documentation for specific protocols specifies which socket types each protocol  
 68691 supports.

68692 The *socketpair()* function is used primarily with UNIX domain sockets and need not be  
 68693 supported for other domains.

68694 **RATIONALE**

68695 The use of the `SOCK_CLOEXEC` and `SOCK_CLOFORK` flags in the *type* argument of *socketpair()*  
 68696 is necessary to avoid a data race in multi-threaded applications. Without `SOCK_CLOFORK`, a  
 68697 file descriptor is leaked into a child process created by one thread in the window between  
 68698 another using *socketpair()* and using *fcntl()* to set the `FD_CLOFORK` flag. Without  
 68699 `SOCK_CLOEXEC`, a file descriptor intentionally inherited by child processes is similarly leaked  
 68700 into an executed program if `FD_CLOEXEC` is not set atomically.

68701 Since socket pairs are often used for communication between a parent and child process,  
 68702 `SOCK_CLOFORK` has to be used with care in order for the pair to be usable. If the parent will be  
 68703 writing and the child will be reading, `SOCK_CLOFORK` should be used when creating the pair,  
 68704 and then *fcntl()* should be used to clear `FD_CLOFORK` for the read side of the pair. This  
 68705 prevents the write side from leaking into other children, ensuring the child will get end-of-file  
 68706 when the parent closes the write side (although the read side can still be leaked). If the parent  
 68707 will be reading and the child will be writing, or if the socket pair will be used bidirectionally,  
 68708 there is no way to prevent the write side(s) being leaked (short of preventing other threads from  
 68709 creating child processes) in order to ensure the parent gets end-of-file when the child closes its

68710 side, and so the two processes should use an alternative method of indicating the end of  
68711 communications, for example using *shutdown()*.

68712 Arranging for FD\_CLOEXEC to be set appropriately is more straightforward. The parent should  
68713 use SOCK\_CLOEXEC when creating the socket pair and the child should clear FD\_CLOEXEC  
68714 on the side to be passed to the new program before calling an *exec* family function to execute it.

68715 The SOCK\_NONBLOCK flag is for convenience in avoiding additional *fcntl()* calls.

#### 68716 FUTURE DIRECTIONS

68717 None.

#### 68718 SEE ALSO

68719 [Section 2.6](#) (on page 525), *socket()*

68720 XBD [<sys/socket.h>](#)

#### 68721 CHANGE HISTORY

68722 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

#### 68723 Issue 7

68724 The description of the [EMFILE] error condition is aligned with the *pipe()* function.

68725 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0336 [835] and XSH/TC2-2008/0337  
68726 [483,835] are applied.

#### 68727 Issue 8

68728 Austin Group Defects 411 and 1318 are applied, adding SOCK\_CLOEXEC, SOCK\_CLOFORK,  
68729 and SOCK\_NONBLOCK.



68730 **NAME**

68731        sprintf — print formatted output

68732 **SYNOPSIS**

68733        #include &lt;stdio.h&gt;

68734        int sprintf(char \*restrict *s*, const char \*restrict *format*, ...);68735 **DESCRIPTION**68736        Refer to *fprintf()*.

68737 **NAME**

68738 sqrt, sqrtf, sqrtl — square root function

68739 **SYNOPSIS**

```
68740 #include <math.h>
68741 double sqrt(double x);
68742 float sqrtf(float x);
68743 long double sqrtl(long double x);
```

68744 **DESCRIPTION**

68745 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 68746 conflict between the requirements described here and the ISO C standard is unintentional. This  
 68747 volume of POSIX.1-2024 defers to the ISO C standard.

68748 These functions shall compute the square root of their argument  $x$ ,  $\sqrt{x}$ .

68749 An application wishing to check for error situations should set *errno* to zero and call  
 68750 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 68751 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 68752 zero, an error has occurred.

68753 **RETURN VALUE**

68754 Upon successful completion, these functions shall return the square root of  $x$ .

68755 MX The returned value shall be dependent on the current rounding direction mode.

68756 MX For finite values of  $x < -0$ , a domain error shall occur, and either a NaN (if supported), or an  
 68757 implementation-defined value shall be returned.

68758 MX If  $x$  is NaN, a NaN shall be returned.

68759 If  $x$  is  $\pm 0$  or  $+\text{Inf}$ ,  $x$  shall be returned.

68760 If  $x$  is  $-\text{Inf}$ , a domain error shall occur, and a NaN shall be returned.

68761 **ERRORS**

68762 These functions shall fail if:

68763 MX Domain Error The finite value of  $x$  is  $< -0$ , or  $x$  is  $-\text{Inf}$ .

68764 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 68765 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 68766 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 68767 shall be raised.

68768 **EXAMPLES**68769 **Taking the Square Root of 9.0**

```
68770 #include <math.h>
68771 ...
68772 double x = 9.0;
68773 double result;
68774 ...
68775 result = sqrt(x);
```

68776 **APPLICATION USAGE**

68777 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
68778 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

68779 **RATIONALE**

68780 None.

68781 **FUTURE DIRECTIONS**

68782 None.

68783 **SEE ALSO**

68784 *feclearexcept()*, *fetetestexcept()*, *isnan()*

68785 XBD Section 4.23 (on page 109), <math.h>, <stdio.h>

68786 **CHANGE HISTORY**

68787 First released in Issue 1. Derived from Issue 1 of the SVID.

68788 **Issue 5**

68789 The DESCRIPTION is updated to indicate how an application should check for an error. This  
68790 text was previously published in the APPLICATION USAGE section.

68791 **Issue 6**

68792 The *sqrtf()* and *sqrtl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

68793 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
68794 revised to align with the ISO/IEC 9899:1999 standard.

68795 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
68796 marked.

68797 **Issue 7**

68798 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0588 [320] is applied.

68799 **Issue 8**

68800 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
68801 standard.

68802 **NAME**

68803           srand — pseudo-random number generator

68804 **SYNOPSIS**

68805           #include <stdlib.h>

68806           void srand(unsigned seed);

68807 **DESCRIPTION**

68808           Refer to *rand()*.

68809 **NAME**

68810           srand48 — seed the uniformly distributed double-precision pseudo-random number generator

68811 **SYNOPSIS**

```
68812 XSI       #include <stdlib.h>  
68813       void srand48(long seedval);
```

68814 **DESCRIPTION**68815       Refer to *drand48()*.

68816 **NAME**

68817           srandom — seed pseudo-random number generator

68818 **SYNOPSIS**

68819 XSI        #include <stdlib.h>

68820           void srandom(unsigned *seed*);

68821 **DESCRIPTION**

68822           Refer to *initstate()*.

68823 **NAME**

68824        scanf — convert formatted input

68825 **SYNOPSIS**

68826        #include &lt;stdio.h&gt;

68827        int sscanf(const char \*restrict *s*, const char \*restrict *format*, ...);68828 **DESCRIPTION**68829        Refer to *fscanf()*.

68830 **NAME**

68831           stat — get file status

68832 **SYNOPSIS**

68833           #include &lt;sys/stat.h&gt;

68834           int stat(const char \*restrict *path*, struct stat \*restrict *buf*);68835 **DESCRIPTION**68836           Refer to *fstatat()*.



68837 **NAME**

68838           statvfs — get file system information

68839 **SYNOPSIS**

68840           #include &lt;sys/statvfs.h&gt;

68841           int statvfs(const char \*restrict *path*, struct statvfs \*restrict *buf*);68842 **DESCRIPTION**68843           Refer to [fstatvfs\(\)](#).

68844 **NAME**

68845            stderr, stdin, stdout — standard I/O streams

68846 **SYNOPSIS**

68847            #include &lt;stdio.h&gt;

68848            extern FILE \*stderr, \*stdin, \*stdout;

68849 **DESCRIPTION**

68850 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
 68851            conflict between the requirements described here and the ISO C standard is unintentional. This  
 68852            volume of POSIX.1-2024 defers to the ISO C standard.

68853            A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type  
 68854            **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer  
 68855            to designate the stream in all further transactions. Normally, there are three open streams with  
 68856            constant pointers declared in the <stdio.h> header and associated with the standard open files.

68857            At program start-up, three streams shall be predefined and already open: *stdin* (standard input,  
 68858            for conventional input) for reading, *stdout* (standard output, for conventional output) for  
 68859            writing, and *stderr* (standard error, for diagnostic output) for writing. When opened, *stderr* shall  
 68860 CX        not be fully buffered; *stdin* and *stdout* shall be fully buffered if and only if the file descriptor  
 68861            associated with the stream is determined not to be associated with an interactive device.

68862 CX        The following symbolic values in <unistd.h> define the file descriptors that shall be associated  
 68863            with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

68864            STDIN\_FILENO        Standard input value, *stdin*. Its value is 0.68865            STDOUT\_FILENO       Standard output value, *stdout*. Its value is 1.68866            STDERR\_FILENO       Standard error value, *stderr*. Its value is 2.

68867            These file descriptors are often all associated with a single open file description which has access  
 68868            mode O\_RDWR (e.g., in the case of a terminal device for a login shell). However, the *stderr*, *stdin*,  
 68869            and *stdout* streams need not be opened for both reading and writing at program start-up in this  
 68870            case.

68871 **RETURN VALUE**

68872            None.

68873 **ERRORS**

68874            No errors are defined.

68875 **EXAMPLES**

68876            None.

68877 **APPLICATION USAGE**

68878            None.

68879 **RATIONALE**

68880            None.

68881 **FUTURE DIRECTIONS**

68882            None.

68883 **SEE ALSO**68884            *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fprintf()*, *fread()*, *fscanf()*, *fseek()*, *getc()*, *isatty()*, *popen()*,  
 68885            *putc()*, *puts()*, *read()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vfprintf()*

68886            XBD &lt;stdio.h&gt;, &lt;unistd.h&gt;

68887 **CHANGE HISTORY**

68888 First released in Issue 1.

68889 **Issue 6**

68890 Extensions beyond the ISO C standard are marked.

68891 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.68892 **Issue 8**68893 Austin Group Defect 1347 is applied, clarifying the requirements for how *stderr*, *stdin*, and *stdout*  
68894 are opened at program start-up.

68895 **NAME**

68896 stpcpy — copy a string and return a pointer to the end of the result

68897 **SYNOPSIS**

```
68898 CX #include <string.h>  
68899 char *stpcpy(char *restrict s1, const char *restrict s2);
```

68900 **DESCRIPTION**

68901 Refer to *strcpy()*.

68902 **NAME**

68903 stpncpy — copy fixed length string, returning a pointer to the array end

68904 **SYNOPSIS**

```
68905 CX #include <string.h>  
68906 char *stpncpy(char *restrict s1, const char *restrict s2, size_t size);
```

68907 **DESCRIPTION**68908 Refer to *strncpy()*.

68909 **NAME**

68910 str2sig — translate between signal names and numbers

68911 **SYNOPSIS**

68912 CX `#include <signal.h>`

68913 `int str2sig(const char *restrict str, int *restrict pnum);`

68914 **DESCRIPTION**

68915 Refer to [sig2str\(\)](#).

68916 **NAME**68917 `strcasecmp`, `strcasecmp_l`, `strncasecmp`, `strncasecmp_l` — case-insensitive string comparisons68918 **SYNOPSIS**

```
68919 #include <strings.h>
68920 int strcasecmp(const char *s1, const char *s2);
68921 int strcasecmp_l(const char *s1, const char *s2,
68922                 locale_t locale);
68923 int strncasecmp(const char *s1, const char *s2, size_t n);
68924 int strncasecmp_l(const char *s1, const char *s2,
68925                  size_t n, locale_t locale);
```

68926 **DESCRIPTION**

68927 The `strcasecmp()` and `strcasecmp_l()` functions shall compare, while ignoring differences in case,  
68928 the string pointed to by `s1` to the string pointed to by `s2`. The `strncasecmp()` and `strncasecmp_l()`  
68929 functions shall compare, while ignoring differences in case, not more than `n` bytes from the  
68930 string pointed to by `s1` to the string pointed to by `s2`.

68931 The `strcasecmp()` and `strncasecmp()` functions use the current locale to determine the case of the  
68932 characters.

68933 The `strcasecmp_l()` and `strncasecmp_l()` functions use the locale represented by `locale` to determine  
68934 the case of the characters.

68935 When the `LC_CTYPE` category of the locale being used is from the POSIX locale, these functions  
68936 shall behave as if the strings had been converted to lowercase and then a byte comparison  
68937 performed, and `errno` shall not be changed on valid input. Otherwise, the results are unspecified.

68938 The behavior is undefined if the `locale` argument to `strcasecmp_l()` or `strncasecmp_l()` is the special  
68939 locale object `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

68940 **RETURN VALUE**

68941 Upon completion, `strcasecmp()` and `strcasecmp_l()` shall return an integer greater than, equal to,  
68942 or less than 0, if the string pointed to by `s1` is, ignoring case, greater than, equal to, or less than  
68943 the string pointed to by `s2`, respectively.

68944 Upon successful completion, `strncasecmp()` and `strncasecmp_l()` shall return an integer greater  
68945 than, equal to, or less than 0, if the possibly null-terminated array pointed to by `s1` is, ignoring  
68946 case, greater than, equal to, or less than the possibly null-terminated array pointed to by `s2`,  
68947 respectively.

68948 **ERRORS**

68949 No errors are defined.

68950 **EXAMPLES**

68951 None.

68952 **APPLICATION USAGE**

68953 None.

68954 **RATIONALE**

68955 None.

68956 **FUTURE DIRECTIONS**

68957 None.

68958 **SEE ALSO**68959 [wcscasecmp\(\)](#)68960 XBD <[strings.h](#)>68961 **CHANGE HISTORY**

68962 First released in Issue 4, Version 2.

68963 **Issue 5**

68964 Moved from X/OPEN UNIX extension to BASE.

68965 **Issue 7**68966 The *strcasecmp()* and *strncasecmp()* functions are moved from the XSI option to the Base.68967 The *strcasecmp\_l()* and *strncasecmp\_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.68969 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0589 [302], XSH/TC1-2008/0590 [294],  
68970 XSH/TC1-2008/0591 [283], and XSH/TC1-2008/0592 [283] are applied.68971 **Issue 8**68972 Austin Group Defect 448 is applied, adding a requirement that *errno* is not changed on valid  
68973 input.



68974 **NAME**

68975        strcat — concatenate two strings

68976 **SYNOPSIS**

68977        #include &lt;string.h&gt;

68978        char \*strcat(char \*restrict s1, const char \*restrict s2);

68979 **DESCRIPTION**

68980 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
68981        conflict between the requirements described here and the ISO C standard is unintentional. This  
68982        volume of POSIX.1-2024 defers to the ISO C standard.

68983        The *strcat()* function shall append a copy of the string pointed to by *s2* (including the  
68984        terminating NUL character) to the end of the string pointed to by *s1*. The initial byte of *s2*  
68985        overwrites the NUL character at the end of *s1*. If copying takes place between objects that  
68986        overlap, the behavior is undefined.

68987 CX        The *strcat()* function shall not change the setting of *errno* on valid input.

68988 **RETURN VALUE**68989        The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.68990 **ERRORS**

68991        No errors are defined.

68992 **EXAMPLES**

68993        None.

68994 **APPLICATION USAGE**

68995        This version is aligned with the ISO C standard; this does not affect compatibility with XPG3  
68996        applications. Reliable error detection by this function was never guaranteed.

68997 **RATIONALE**

68998        None.

68999 **FUTURE DIRECTIONS**

69000        None.

69001 **SEE ALSO**69002        [strncat\(\)](#)69003        XBD [<string.h>](#)69004 **CHANGE HISTORY**

69005        First released in Issue 1. Derived from Issue 1 of the SVID.

69006 **Issue 6**69007        The *strcat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.69008 **Issue 8**

69009        Austin Group Defect 448 is applied, adding a requirement that *strcat()* does not change the  
69010        setting of *errno* on valid input.

69011 **NAME**

69012           strchr — string scanning operation

69013 **SYNOPSIS**

69014           #include &lt;string.h&gt;

69015           char \*strchr(const char \*s, int c);

69016 **DESCRIPTION**

69017 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
69018       conflict between the requirements described here and the ISO C standard is unintentional. This  
69019       volume of POSIX.1-2024 defers to the ISO C standard.

69020       The *strchr()* function shall locate the first occurrence of *c* (converted to a **char**) in the string  
69021       pointed to by *s*. The terminating NUL character is considered to be part of the string.

69022 CX       The *strchr()* function shall not change the setting of *errno* on valid input.

69023 **RETURN VALUE**

69024       Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not  
69025       found.

69026 **ERRORS**

69027       No errors are defined.

69028 **EXAMPLES**

69029       None.

69030 **APPLICATION USAGE**

69031       None.

69032 **RATIONALE**

69033       None.

69034 **FUTURE DIRECTIONS**

69035       None.

69036 **SEE ALSO**69037       [strrchr\(\)](#)69038       XBD [<string.h>](#)69039 **CHANGE HISTORY**

69040       First released in Issue 1. Derived from Issue 1 of the SVID.

69041 **Issue 6**

69042       Extensions beyond the ISO C standard are marked.

69043 **Issue 8**

69044       Austin Group Defect 448 is applied, adding a requirement that *strchr()* does not change the  
69045       setting of *errno* on valid input.

69046 **NAME**

69047 strcmp — compare two strings

69048 **SYNOPSIS**

69049 #include &lt;string.h&gt;

69050 int strcmp(const char \*s1, const char \*s2);

69051 **DESCRIPTION**

69052 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 69053 conflict between the requirements described here and the ISO C standard is unintentional. This  
 69054 volume of POSIX.1-2024 defers to the ISO C standard.

69055 The *strcmp()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

69056 The sign of a non-zero return value shall be determined by the sign of the difference between the  
 69057 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
 69058 being compared.

69059 CX The *strcmp()* function shall not change the setting of *errno* on valid input.

69060 **RETURN VALUE**

69061 Upon completion, *strcmp()* shall return an integer greater than, equal to, or less than 0, if the  
 69062 string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*,  
 69063 respectively.

69064 **ERRORS**

69065 No errors are defined.

69066 **EXAMPLES**69067 **Checking a Password Entry**

69068 The following example compares the information read from standard input to the value of the  
 69069 name of the user entry. If the *strcmp()* function returns 0 (indicating a match), a further check  
 69070 will be made to see if the user entered the proper old password. The *crypt()* function shall  
 69071 encrypt the old password entered by the user, using the value of the encrypted password in the  
 69072 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the  
 69073 structure, the entered password *oldpasswd* is the correct user's password. Finally, the program  
 69074 encrypts the new password so that it can store the information in the **passwd** structure.

```

69075 #include <string.h>
69076 #include <unistd.h>
69077 #include <stdio.h>
69078 ...
69079 int valid_change;
69080 struct passwd *p;
69081 char user[100];
69082 char oldpasswd[100];
69083 char newpasswd[100];
69084 char savepasswd[100];
69085 ...
69086 if (strcmp(p->pw_name, user) == 0) {
69087     if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
69088         strcpy(savepasswd, crypt(newpasswd, user));
69089         p->pw_passwd = savepasswd;
69090         valid_change = 1;
69091     }

```

```
69092         else {
69093             fprintf(stderr, "Old password is not valid\n");
69094         }
69095     }
69096     ...
```

**69097 APPLICATION USAGE**

69098 None.

**69099 RATIONALE**

69100 None.

**69101 FUTURE DIRECTIONS**

69102 None.

**69103 SEE ALSO**

69104 [strncmp\(\)](#)

69105 XBD <[string.h](#)>

**69106 CHANGE HISTORY**

69107 First released in Issue 1. Derived from Issue 1 of the SVID.

**69108 Issue 6**

69109 Extensions beyond the ISO C standard are marked.

**69110 Issue 8**

69111 Austin Group Defect 448 is applied, adding a requirement that *strcmp()* does not change the  
69112 setting of *errno* on valid input.

69113 **NAME**

69114 strcoll, strcoll\_l — string comparison using collating information

69115 **SYNOPSIS**

```
69116 #include <string.h>
69117 int strcoll(const char *s1, const char *s2);
69118 CX int strcoll_l(const char *s1, const char *s2,
69119 locale_t locale);
```

69120 **DESCRIPTION**

69121 CX For `strcoll()`: The functionality described on this reference page is aligned with the ISO C  
 69122 standard. Any conflict between the requirements described here and the ISO C standard is  
 69123 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

69124 CX The `strcoll()` and `strcoll_l()` functions shall compare the string pointed to by `s1` to the string  
 69125 pointed to by `s2`, both interpreted as appropriate to the `LC_COLLATE` category of the current  
 69126 CX locale, or of the locale represented by `locale`, respectively.

69127 CX The `strcoll()` and `strcoll_l()` functions shall not change the setting of `errno` if successful.

69128 Since no return value is reserved to indicate an error, an application wishing to check for error  
 69129 CX situations should set `errno` to 0, then call `strcoll()`, or `strcoll_l()` then check `errno`.

69130 CX The behavior is undefined if the `locale` argument to `strcoll_l()` is the special locale object  
 69131 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

69132 **RETURN VALUE**

69133 Upon successful completion, `strcoll()` shall return an integer greater than, equal to, or less than 0,  
 69134 according to whether the string pointed to by `s1` is greater than, equal to, or less than the string  
 69135 CX pointed to by `s2` when both are interpreted as appropriate to the current locale. On error,  
 69136 `strcoll()` may set `errno`, but no return value is reserved to indicate an error.

69137 Upon successful completion, `strcoll_l()` shall return an integer greater than, equal to, or less than  
 69138 0, according to whether the string pointed to by `s1` is greater than, equal to, or less than the  
 69139 string pointed to by `s2` when both are interpreted as appropriate to the locale represented by  
 69140 `locale`. On error, `strcoll_l()` may set `errno`, but no return value is reserved to indicate an error.

69141 **ERRORS**

69142 These functions may fail if:

69143 CX `[EINVAL]` The `s1` or `s2` arguments contain characters outside the domain of the collating  
 69144 sequence.

69145 **EXAMPLES**69146 **Comparing Nodes**

69147 The following example uses an application-defined function, `node_compare()`, to compare two  
 69148 nodes based on an alphabetical ordering of the `string` field.

```
69149 #include <string.h>
69150 ...
69151 struct node { /* These are stored in the table. */
69152     char *string;
69153     int length;
69154 };
69155 ...
```

```
69156     int node_compare(const void *node1, const void *node2)
69157     {
69158         return strcoll(((const struct node *)node1)->string,
69159                       ((const struct node *)node2)->string);
69160     }
69161     ...
```

**69162 APPLICATION USAGE**

69163 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

**69164 RATIONALE**

69165 None.

**69166 FUTURE DIRECTIONS**

69167 None.

**69168 SEE ALSO**

69169 [\*alphasort\(\)\*](#), [\*strcmp\(\)\*](#), [\*strxfrm\(\)\*](#)

69170 XBD [\*\*<string.h>\*\*](#)

**69171 CHANGE HISTORY**

69172 First released in Issue 3.

**69173 Issue 5**

69174 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

**69175 Issue 6**

69176 Extensions beyond the ISO C standard are marked.

69177 The following new requirements on POSIX implementations derive from alignment with the  
69178 Single UNIX Specification:

- 69179 • The [EINVAL] optional error condition is added.

69180 An example is added.

**69181 Issue 7**

69182 The *strcoll\_1()* function is added from The Open Group Technical Standard, 2006, Extended API  
69183 Set Part 4.

69184 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0593 [283] and XSH/TC1-2008/0594  
69185 [283] are applied.

69186 **NAME**

69187 stpcpy, strcpy — copy a string

69188 **SYNOPSIS**

69189 #include &lt;string.h&gt;

69190 CX char \*stpcpy(char \*restrict s1, const char \*restrict s2);

69191 char \*strcpy(char \*restrict s1, const char \*restrict s2);

69192 **DESCRIPTION**

69193 CX For `strcpy()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

69196 CX The `stpcpy()` and `strcpy()` functions shall copy the string pointed to by `s2` (including the terminating NUL character) into the array pointed to by `s1`.

69198 If copying takes place between objects that overlap, the behavior is undefined.

69199 CX The `strcpy()` and `stpcpy()` functions shall not change the setting of `errno` on valid input.

69200 **RETURN VALUE**

69201 CX The `stpcpy()` function shall return a pointer to the terminating NUL character copied into the `s1` buffer.

69203 The `strcpy()` function shall return `s1`.

69204 No return values are reserved to indicate an error.

69205 **ERRORS**

69206 No errors are defined.

69207 **EXAMPLES**69208 **Construction of a Multi-Part Message in a Single Buffer**

69209 #include &lt;string.h&gt;

69210 #include &lt;stdio.h&gt;

69211 int

69212 main (void)

69213 {

69214 char buffer [10];

69215 char \*name = buffer;

69216 name = stpcpy (stpcpy (stpcpy (name, "ice"), "-"), "cream");

69217 puts (buffer);

69218 return 0;

69219 }

69220 **Initializing a String**

69221 The following example copies the string "-----" into the `permstring` variable.

69222 #include &lt;string.h&gt;

69223 ...

69224 static char permstring[11];

69225 ...

69226 strcpy (permstring, "-----");

69227 ...

69228 **Storing a Key and Data**

69229 The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the  
 69230 key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there.  
 69231 (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct**  
 69232 **element** \*.)

```

69233 #include <string.h>
69234 #include <stdlib.h>
69235 #include <stdio.h>
69236 ...
69237 /* Structure used to read data and store it. */
69238 struct element {
69239     char *key;
69240     char *data;
69241 };
69242 struct element *tbl, *curtbl;
69243 char *key, *data;
69244 int count;
69245 ...
69246 void dbfree(struct element *, int);
69247 ...
69248 if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
69249     perror("malloc"); dbfree(tbl, count); return NULL;
69250 }
69251 strcpy(curtbl->key, key);
69252 if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
69253     perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
69254 }
69255 strcpy(curtbl->data, data);
69256 ...

```

69257 **APPLICATION USAGE**

69258 Character movement is performed differently in different implementations. Thus, overlapping  
 69259 moves may yield surprises.

69260 This version is aligned with the ISO C standard; this does not affect compatibility with XPG3  
 69261 applications. Reliable error detection by this function was never guaranteed.

69262 **RATIONALE**

69263 None.

69264 **FUTURE DIRECTIONS**

69265 None.

69266 **SEE ALSO**

69267 *strncpy()*, *wscpy()*

69268 XBD <[string.h](#)>

69269 **CHANGE HISTORY**

69270 First released in Issue 1. Derived from Issue 1 of the SVID.



69271 **Issue 6**

69272 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

69273 **Issue 7**

69274 The *strcpy()* function is added from The Open Group Technical Standard, 2006, Extended API  
69275 Set Part 1.

69276 **Issue 8**

69277 Austin Group Defect 448 is applied, adding a requirement that *strcpy()* and *strcpy()* do not  
69278 change the setting of *errno* on valid input.

69279 Austin Group Defect 1787 is applied, changing the NAME section.

69280 **NAME**

69281 strcspn — get the length of a complementary substring

69282 **SYNOPSIS**

69283 #include &lt;string.h&gt;

69284 size\_t strcspn(const char \*s1, const char \*s2);

69285 **DESCRIPTION**69286 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
69287 conflict between the requirements described here and the ISO C standard is unintentional. This  
69288 volume of POSIX.1-2024 defers to the ISO C standard.69289 The *strcspn()* function shall compute the length (in bytes) of the maximum initial segment of the  
69290 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.69291 CX The *strcspn()* function shall not change the setting of *errno* on valid input.69292 **RETURN VALUE**69293 The *strcspn()* function shall return the length of the computed segment of the string pointed to  
69294 by *s1*; no return value is reserved to indicate an error.69295 **ERRORS**

69296 No errors are defined.

69297 **EXAMPLES**

69298 None.

69299 **APPLICATION USAGE**

69300 None.

69301 **RATIONALE**

69302 None.

69303 **FUTURE DIRECTIONS**

69304 None.

69305 **SEE ALSO**69306 [strspn\(\)](#)69307 XBD <[string.h](#)>69308 **CHANGE HISTORY**

69309 First released in Issue 1. Derived from Issue 1 of the SVID.

69310 **Issue 5**69311 The RETURN VALUE section is updated to indicate that *strcspn()* returns the length of *s1*, and  
69312 not *s1* itself as was previously stated.69313 **Issue 6**69314 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is  
69315 updated to indicate that the computed segment length is returned, not the *s1* length.69316 **Issue 8**69317 Austin Group Defect 448 is applied, adding a requirement that *strcspn()* does not change the  
69318 setting of *errno* on valid input.

69319 **NAME**69320 `strdup`, `strndup` — duplicate a specific number of bytes from a string69321 **SYNOPSIS**

```
69322 CX #include <string.h>
69323 char *strdup(const char *s);
69324 char *strndup(const char *s, size_t size);
```

69325 **DESCRIPTION**

69326 The `strdup()` function shall return a pointer to a new string, which is a duplicate of the string  
69327 pointed to by `s`. The returned pointer can be passed to `free()`. A null pointer is returned if the  
69328 new string cannot be created.

69329 The `strndup()` function shall be equivalent to the `strdup()` function, duplicating the provided `s` in  
69330 a new block of memory allocated as if by using `malloc()`, with the exception being that `strndup()`  
69331 copies at most `size` bytes from the array `s` into the newly allocated memory, terminating the new  
69332 string with a null byte. If `s` contains a null terminator within the first `size` bytes, all bytes in `s` up  
69333 to and including the null terminator shall be copied into the new memory buffer. The `strndup()`  
69334 function shall not examine more than `size` bytes of the array pointed to by `s`. The newly created  
69335 string shall always be properly terminated.

69336 **RETURN VALUE**

69337 The `strdup()` function shall return a pointer to a new string on success. Otherwise, it shall return  
69338 a null pointer and set `errno` to indicate the error.

69339 Upon successful completion, the `strndup()` function shall return a pointer to the newly allocated  
69340 memory containing the duplicated string. Otherwise, it shall return a null pointer and set `errno`  
69341 to indicate the error.

69342 **ERRORS**

69343 These functions shall fail if:

69344 [ENOMEM] Storage space available is insufficient.

69345 **EXAMPLES**

69346 None.

69347 **APPLICATION USAGE**

69348 For functions that allocate memory as if by `malloc()`, the application should release such memory  
69349 when it is no longer required by a call to `free()`. For `strdup()` and `strndup()`, this is the return  
69350 value.

69351 Implementations are free to `malloc()` a buffer containing either  $(size + 1)$  bytes or  $(strlen(s, size) + 1)$  bytes. Applications should not assume that `strndup()` will allocate  $(size + 1)$  bytes when `strlen(s)` is smaller than `size`.

69354 **RATIONALE**

69355 None.

69356 **FUTURE DIRECTIONS**

69357 None.

69358 **SEE ALSO**

69359 [free\(\)](#), [wcsdup\(\)](#)

69360 XBD [<string.h>](#)

69361 **CHANGE HISTORY**

69362 First released in Issue 4, Version 2.

69363 **Issue 5**

69364 Moved from X/OPEN UNIX extension to BASE.

69365 **Issue 7**69366 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the ``may fail'' [ENOMEM]  
69367 error to become a ``shall fail'' error.69368 The *strdup()* function is moved from the XSI option to the Base.69369 The *strndup()* function is added from The Open Group Technical Standard, 2006, Extended API  
69370 Set Part 1.69371 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by  
69372 *malloc()*.

69373 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0338 [738] is applied.

69374 **Issue 8**69375 Austin Group Defect 1019 is applied, clarifying that the *strndup()* argument *s* need not point to a  
69376 null-terminated string.

69377 **NAME**

69378            strerror, strerror\_l, strerror\_r — get error message string

69379 **SYNOPSIS**

69380            #include &lt;string.h&gt;

69381            char \*strerror(int *errnum*);69382 CX         char \*strerror\_l(int *errnum*, locale\_t *locale*);69383            int strerror\_r(int *errnum*, char \**strerrbuf*, size\_t *buflen*);69384 **DESCRIPTION**69385 CX         For *strerror()*: The functionality described on this reference page is aligned with the ISO C  
69386 standard. Any conflict between the requirements described here and the ISO C standard is  
69387 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.69388            The *strerror()* function shall map the error number in *errnum* to a locale-dependent error  
69389 message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*,  
69390 but *strerror()* shall map any value of type **int** to a message.69391 CX         The application shall not modify the string returned. The returned string pointer might be  
69392 CX         invalidated or the string content might be overwritten by a subsequent call to *strerror()*, or by a  
69393 subsequent call to *strerror\_l()* in the same thread. The returned pointer and the string content  
69394 might also be invalidated if the calling thread is terminated.69395 CX         The string may be overwritten by a subsequent call to *strerror\_l()* in the same thread.69396            The contents of the error message strings returned by *strerror()* should be determined by the  
69397 setting of the *LC\_MESSAGES* category in the current locale.69398            The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
69399 *strerror()*.69400 CX         The *strerror()* and *strerror\_l()* functions shall not change the setting of *errno* if successful.69401            Since no return value is reserved to indicate an error of *strerror()*, an application wishing to  
69402 check for error situations should set *errno* to 0, then call *strerror()*, then check *errno*. Similarly,  
69403 since *strerror\_l()* is required to return a string for some errors, an application wishing to check  
69404 for all error situations should set *errno* to 0, then call *strerror\_l()*, then check *errno*.69405            The *strerror()* function need not be thread-safe; however, *strerror()* shall avoid data races with all  
69406 other functions.69407 CX         The *strerror\_l()* function shall map the error number in *errnum* to a locale-dependent error  
69408 message string in the locale represented by *locale* and shall return a pointer to it.69409            The *strerror\_r()* function shall map the error number in *errnum* to a locale-dependent error  
69410 message string and shall return the string in the buffer pointed to by *strerrbuf*, with length  
69411 *buflen*.69412 CX         If the value of *errnum* is a valid error number, the message string shall indicate what error  
69413 occurred; if the value of *errnum* is zero, the message string shall either be an empty string or  
69414 indicate that no error occurred; otherwise, if these functions complete successfully, the message  
69415 string shall indicate that an unknown error occurred.69416 CX         The behavior is undefined if the *locale* argument to *strerror\_l()* is the special locale object  
69417 *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.

69418 **RETURN VALUE**

69419 Upon completion, whether successful or not, *strerror()* shall return a pointer to the generated  
 69420 CX message string. On error *errno* may be set, but no return value is reserved to indicate an error.

69421 Upon successful completion, *strerror\_l()* shall return a pointer to the generated message string. If  
 69422 *errnum* is not a valid error number, *errno* may be set to [EINVAL], but a pointer to a message  
 69423 string shall still be returned. If any other error occurs, *errno* shall be set to indicate the error and  
 69424 a null pointer shall be returned.

69425 Upon successful completion, *strerror\_r()* shall return 0. Otherwise, an error number shall be  
 69426 returned to indicate the error.

69427 **ERRORS**

69428 These functions may fail if:

69429 CX [EINVAL] The value of *errnum* is neither a valid error number nor zero.

69430 The *strerror\_r()* function shall fail if:

69431 CX [ERANGE] Insufficient storage was supplied via *strerrbuf* and *buflen* to contain the  
 69432 generated message string.

69433 **EXAMPLES**

69434 None.

69435 **APPLICATION USAGE**

69436 Historically in some implementations, calls to *perror()* would overwrite the string that the  
 69437 pointer returned by *strerror()* points to. Such implementations did not conform to the ISO C  
 69438 standard; however, application developers should be aware of this behavior if they wish their  
 69439 applications to be portable to such implementations.

69440 Applications should use *strerror\_l()* rather than *strerror()* or *strerror\_r()* to avoid thread safety  
 69441 and possible alternative (non-conforming) versions of these functions in some implementations.

69442 **RATIONALE**

69443 The *strerror\_l()* function is required to be thread-safe, thereby eliminating the need for an  
 69444 equivalent to the *strerror\_r()* function.

69445 Earlier versions of this standard did not explicitly require that the error message strings returned  
 69446 by *strerror()* and *strerror\_r()* provide any information about the error. This version of the  
 69447 standard requires a meaningful message for any successful completion.

69448 Since no return value is reserved to indicate a *strerror()* error, but all calls (whether successful or  
 69449 not) must return a pointer to a message string, on error *strerror()* can return a pointer to an  
 69450 empty string or a pointer to a meaningful string that can be printed.

69451 Note that the [EINVAL] error condition is a may fail error. If an invalid error number is supplied  
 69452 as the value of *errnum*, applications should be prepared to handle any of the following:

- 69453 1. Error (with no meaningful message): *errno* is set to [EINVAL], the return value is a pointer  
 69454 to an empty string.
- 69455 2. Successful completion: *errno* is unchanged and the return value points to a string like  
 69456 "unknown error" or "error number xxx" (where xxx is the value of *errnum*).
- 69457 3. Combination of #1 and #2: *errno* is set to [EINVAL] and the return value points to a string  
 69458 like "unknown error" or "error number xxx" (where xxx is the value of *errnum*).  
 69459 Since applications frequently use the return value of *strerror()* as an argument to  
 69460 functions like *fprintf()* (without checking the return value) and since applications have no  
 69461 way to parse an error message string to determine whether *errnum* represents a valid

69462 error number, implementations are encouraged to implement #3. Similarly,  
 69463 implementations are encouraged to have *strerror\_r()* return [EINVAL] and put a string  
 69464 like "unknown error" or "error number xxx" in the buffer pointed to by *strrdbuf*  
 69465 when the value of *errnum* is not a valid error number.

69466 Additionally, implementations are encouraged to null terminate *strrdbuf* when failing with  
 69467 [ERANGE] for any size other than *buflen* of zero.

69468 Some applications rely on being able to set *errno* to 0 before calling a function with no reserved  
 69469 value to indicate an error, then call *strerror(errno)* afterwards to detect whether an error occurred  
 69470 (because *errno* changed) or to indicate success (because *errno* remained zero). This usage pattern  
 69471 requires that *strerror(0)* succeed with useful results. Previous versions of the standard did not  
 69472 specify the behavior when *errnum* is zero.

#### 69473 FUTURE DIRECTIONS

69474 None.

#### 69475 SEE ALSO

69476 *perror()*

69477 XBD <*string.h*>

#### 69478 CHANGE HISTORY

69479 First released in Issue 3.

#### 69480 Issue 5

69481 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

69482 A note indicating that the *strerror()* function need not be reentrant is added to the  
 69483 DESCRIPTION.

#### 69484 Issue 6

69485 Extensions beyond the ISO C standard are marked.

69486 The following new requirements on POSIX implementations derive from alignment with the  
 69487 Single UNIX Specification:

- 69488 • In the RETURN VALUE section, the fact that *errno* may be set is added.
- 69489 • The [EINVAL] optional error condition is added.

69490 The normative text is updated to avoid use of the term “must” for application requirements.

69491 The *strerror\_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

69492 The *strerror\_r()* function is marked as part of the Thread-Safe Functions option.

#### 69493 Issue 7

69494 Austin Group Interpretation 1003.1-2001 #072 is applied, updating the ERRORS section.

69495 Austin Group Interpretation 1003.1-2001 #156 is applied.

69496 Austin Group Interpretation 1003.1-2001 #187 is applied, clarifying the behavior when the  
 69497 generated error message is an empty string.

69498 SD5-XSH-ERN-191 is applied, updating the APPLICATION USAGE section.

69499 The *strerror\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
 69500 Set Part 4.

69501 The *strerror\_r()* function is moved from the Thread-Safe Functions option to the Base.

69502 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0595 [75], XSH/TC1-2008/0596 [447],

- 69503 XSH/TC1-2008/0597 [382,428], XSH/TC1-2008/0598 [283], XSH/TC1-2008/0599 [382,428],  
69504 XSH/TC1-2008/0600 [283], and XSH/TC1-2008/0601 [382,428] are applied.
- 69505 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0339 [656] is applied.
- 69506 **Issue 8**
- 69507 Austin Group Defect 398 is applied, changing the [ERANGE] error from “may fail” to “shall  
69508 fail”.
- 69509 Austin Group Defect 655 is applied, changing the APPLICATION USAGE section.
- 69510 Austin Group Defect 1302 is applied, aligning the *strerror()* function with the  
69511 ISO/IEC 9899:2018 standard.



69512 **NAME**

69513 strfmon, strfmon\_l — convert monetary value to a string

69514 **SYNOPSIS**

```
69515 #include <monetary.h>
69516 ssize_t strfmon(char *restrict s, size_t maxsize,
69517     const char *restrict format, ...);
69518 ssize_t strfmon_l(char *restrict s, size_t maxsize,
69519     locale_t locale, const char *restrict format, ...);
```

69520 **DESCRIPTION**

69521 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the  
 69522 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

69523 The format is a character string, beginning and ending in its initial state, if any, that contains two  
 69524 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*  
 69525 *specifications*, each of which shall result in the fetching of zero or more arguments which are  
 69526 converted and formatted. The results are undefined if there are insufficient arguments for the  
 69527 format. If the format is exhausted while arguments remain, the excess arguments are simply  
 69528 ignored.

69529 The application shall ensure that a conversion specification consists of the following sequence:

- 69530 • A '%' character
- 69531 • Optional flags
- 69532 • Optional field width
- 69533 • Optional left precision
- 69534 • Optional right precision
- 69535 • A required conversion specifier character that determines the conversion to be performed

69536 The *strfmon\_l()* function shall be equivalent to the *strfmon()* function, except that the locale data  
 69537 used is from the locale represented by *locale*.

69538 **Flags**

69539 One or more of the following optional flags can be specified to control the conversion:

69540 =*f* An '=' followed by a single character *f* which is used as the numeric fill character. In  
 69541 order to work with precision or width counts, the fill character shall be a single byte  
 69542 character; if not, the behavior is undefined. The default numeric fill character is the  
 69543 <space>. This flag does not affect field width filling which always uses the <space>.  
 69544 This flag is ignored unless a left precision (see below) is specified.

69545 ^ Do not format the currency amount with grouping characters. The default is to insert  
 69546 the grouping characters if defined for the current locale.

69547 + or ( Specify the style of representing positive and negative currency amounts. Only one of  
 69548 '+' or '(' may be specified.

69549 If '+' is specified, the locale's **positive\_sign** and **negative\_sign** values shall be used  
 69550 (for example, in many locales, the empty string if positive and '-' if negative).  
 69551 However, if both values would be returned by *localeconv()* as empty strings, *strfmon()*  
 69552 shall fail. The placement of the signs (if not empty) shall depend on the locale settings:

69553 • For the *n* conversion specifier, the placement specified by the locale's  
69554 **p\_sign\_posn** and **n\_sign\_posn** values shall be used.

69555 • For the *i* conversion specifier, the placement specified by the locale's  
69556 **int\_p\_sign\_posn** and **int\_n\_sign\_posn** values shall be used.

69557 If a sign's placement cannot be determined from these locale values because a value  
69558 that needs to be used would be returned by *localeconv()* as 0 or {CHAR\_MAX}, the sign  
69559 shall be placed as if the relevant value was 1.

69560 If ' (' is specified, negative amounts shall be enclosed within parentheses and the  
69561 locale's **positive\_sign** and **negative\_sign** values shall not be used.

69562 If neither flag is specified, the style used shall depend on the locale settings:

69563 • For the *n* conversion specifier, the style specified by the locale's **p\_sign\_posn** and  
69564 **n\_sign\_posn** values shall be used.

69565 • For the *i* conversion specifier, the style specified by the locale's **int\_p\_sign\_posn**  
69566 and **int\_n\_sign\_posn** values shall be used.

69567 If the style cannot be determined from these locale values because a value that needs to  
69568 be used would be returned by *localeconv()* as {CHAR\_MAX}, the style used shall be that  
69569 specified for the '+' flag; if this would cause *strfmon()* to fail because the locale's  
69570 **positive\_sign** and **negative\_sign** values would both be returned by *localeconv()* as  
69571 empty strings, *strfmon()* shall behave as if the **negative\_sign** value was the string "-".

69572 ! Suppress the currency symbol from the output conversion.

69573 - Specify the alignment. If this flag is present the result of the conversion is left-justified  
69574 (padded to the right) rather than right-justified. This flag shall be ignored unless a field  
69575 width (see below) is specified.

## 69576 Field Width

69577 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result  
69578 of the conversion is right-justified (or left-justified if the flag '-' is specified). The  
69579 default is 0.

## 69580 Left Precision

69581 *#n* A '#' followed by a decimal digit string *n* specifying a maximum number of digits  
69582 expected to be formatted to the left of the radix character. This option can be used to  
69583 keep the formatted output from multiple calls to the *strfmon()* function aligned in the  
69584 same columns. It can also be used to fill unused positions with a special character as in  
69585 "\$\*\*\*123.45". This option causes an amount to be formatted as if it has the number  
69586 of digits specified by *n*. If more than *n* digit positions are required, this conversion  
69587 specification is ignored. Digit positions in excess of those actually required are filled  
69588 with the numeric fill character (see the *=f* flag above).

69589 If grouping has not been suppressed with the '^' flag, and it is defined for the current  
69590 locale, grouping separators are inserted before the fill characters (if any) are added.  
69591 Grouping separators are not applied to fill characters even if the fill character is a digit.

69592 To ensure alignment, any characters appearing before or after the number in the  
69593 formatted output such as currency or sign symbols are padded as necessary with  
69594 <space> characters to make their positive and negative formats an equal length.

69595 **Right Precision**

69596 .*p* A <period> followed by a decimal digit string *p* specifying the number of digits after  
 69597 the radix character. If the value of the right precision *p* is 0, no radix character appears.  
 69598 If a right precision is not included, a default specified by the current locale is used. The  
 69599 amount being formatted is rounded to the specified number of digits prior to  
 69600 formatting.

69601 **Conversion Specifier Characters**

69602 The conversion specifier characters and their meanings are:

69603 *i* The **double** argument is formatted according to the locale's international currency  
 69604 format (for example, in the US: USD 1,234.56). If the argument is  $\pm\text{Inf}$  or NaN, the result  
 69605 of the conversion is unspecified.

69606 *n* The **double** argument is formatted according to the locale's national currency format  
 69607 (for example, in the US: \$1,234.56). If the argument is  $\pm\text{Inf}$  or NaN, the result of the  
 69608 conversion is unspecified.

69609  $\%$  Convert to a ' $\%$ '; no argument is converted. The entire conversion specification shall  
 69610 be  $\%\%$ .

69611 **Locale Information**

69612 The *LC\_MONETARY* category of the current locale affects the behavior of this function including  
 69613 the monetary radix character (which may be different from the numeric radix character affected  
 69614 by the *LC\_NUMERIC* category), the grouping separator, the currency symbols, and formats. The  
 69615 international currency symbol should be conformant with the ISO 4217:2015 standard.

69616 If the value of *maxsize* is greater than {SSIZE\_MAX}, the result is implementation-defined.

69617 The behavior is undefined if the *locale* argument to *strfmon\_l()* is the special locale object  
 69618 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.

69619 **RETURN VALUE**

69620 If all conversions are successful and the total number of resulting bytes including the  
 69621 terminating null byte is not more than *maxsize*, these functions shall return the number of bytes  
 69622 placed into the array pointed to by *s*, not including the terminating NUL character. Otherwise,  
 69623 -1 shall be returned, the contents of the array are unspecified, and *errno* shall be set to indicate  
 69624 the error.

69625 **ERRORS**

69626 These functions shall fail if:

69627 [E2BIG] Conversion stopped due to lack of space in the buffer.

69628 [EINVAL] The '+' flag was included in a conversion specification and the locale's  
 69629 **positive\_sign** and **negative\_sign** values would both be returned by  
 69630 *localeconv()* as empty strings.

69631 **EXAMPLES**

69632 Given a locale for the US and the values 123.45, -123.45, and 3456.781, the following output  
69633 might be produced. Square brackets (" [ ] ") are used in this example to delimit the output.

69634	%n	[\$123.45]	Default formatting
69635		[-\$123.45]	
69636		[\$3,456.78]	
69637	%11n	[ \$123.45]	Right align within an 11-character field
69638		[-\$123.45]	
69639		[ \$3,456.78]	
69640	%#5n	[ \$ 123.45]	Aligned columns for values up to 99999
69641		[-\$ 123.45]	
69642		[ \$ 3,456.78]	
69643	%=*#5n	[ \$***123.45]	Specify a fill character
69644		[-\$***123.45]	
69645		[ \$*3,456.78]	
69646	%=0#5n	[ \$000123.45]	Fill characters do not use grouping
69647		[-\$000123.45]	even if the fill character is a digit
69648		[ \$03,456.78]	
69649	%^#5n	[ \$ 123.45]	Disable the grouping separator
69650		[-\$ 123.45]	
69651		[ \$ 3456.78]	
69652	%^#5.0n	[ \$ 123]	Round off to whole units
69653		[-\$ 123]	
69654		[ \$ 3457]	
69655	%^#5.4n	[ \$ 123.4500]	Increase the precision
69656		[-\$ 123.4500]	
69657		[ \$ 3456.7810]	
69658	%(#5n	[ \$ 123.45 ]	Use an alternative pos/neg style
69659		[ (\$ 123.45 )]	
69660		[ \$ 3,456.78 ]	
69661	%!(#5n	[ 123.45 ]	Disable the currency symbol
69662		[ ( 123.45 )]	
69663		[ 3,456.78 ]	
69664	%-14#5.4n	[ \$ 123.4500 ]	Left-justify the output
69665		[-\$ 123.4500 ]	
69666		[ \$ 3,456.7810 ]	
69667	%14#5.4n	[ \$ 123.4500]	Corresponding right-justified output
69668		[ -\$ 123.4500]	
69669		[ \$ 3,456.7810]	

69670 See also the EXAMPLES section in *fprintf()*.

69671 **APPLICATION USAGE**

69672 The '+' flag should be used with care, because if the locale's **positive\_sign** and **negative\_sign**  
69673 values are both empty strings, there is no way to distinguish negative from positive values with  
69674 signs and therefore *strfmon()* fails. If the application has a preference for signs but parentheses  
69675 are acceptable, it should try *strfmon()* with the '+' flag first, and if it fails with [EINVAL] then

69676 repeat the call without the '+' flag.

#### 69677 RATIONALE

69678 The [EINVAL] error condition applies only when the '+' flag is used because this flag indicates  
69679 that the application requires the use of signs, and if there are no signs in the locale data then this  
69680 requirement cannot be satisfied. When neither '+' nor '(' is used, the application is requesting  
69681 whatever formatting is appropriate for the locale, and so *strfmon()* has a fallback of using a '-'  
69682 sign for negative values in cases where the locale data does not indicate parentheses should be  
69683 used and has no signs.

#### 69684 FUTURE DIRECTIONS

69685 Lowercase conversion characters are reserved for future standards use and uppercase for  
69686 implementation-defined use.

#### 69687 SEE ALSO

69688 *fprintf()*, *localeconv()*

69689 XBD <monetary.h>

#### 69690 CHANGE HISTORY

69691 First released in Issue 4.

#### 69692 Issue 5

69693 Moved from ENHANCED I18N to BASE.

69694 The [ENOSYS] error is removed.

69695 Text is added to the DESCRIPTION warning about values of *maxsize* that are greater than  
69696 {SSIZE\_MAX}.

#### 69697 Issue 6

69698 The normative text is updated to avoid use of the term “must” for application requirements.

69699 The **restrict** keyword is added to the *strfmon()* prototype for alignment with the  
69700 ISO/IEC 9899:1999 standard.

69701 The EXAMPLES section is reworked, clarifying the output format.

#### 69702 Issue 7

69703 SD5-XSH-ERN-29 is applied, updating the examples for % (#5n and %! (#5n.

69704 SD5-XSH-ERN-233 is applied, changing the definition of the '+' or '(' flags to refer to  
69705 multiple locales.

69706 The *strfmon()* function is moved from the XSI option to the Base.

69707 The *strfmon\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
69708 Set Part 4.

69709 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0602 [302], XSH/TC1-2008/0603 [283],  
69710 and XSH/TC1-2008/0604 [283] are applied.

#### 69711 Issue 8

69712 Austin Group Defect 1199 is applied, changing the requirements for the '+' and '(' flags and  
69713 adding the [EINVAL] error.

69714 **NAME**

69715 strptime, strptime\_l — convert date and time to a string

69716 **SYNOPSIS**

69717 #include &lt;time.h&gt;

```
69718 size_t strptime(char *restrict s, size_t maxsize,
69719               const char *restrict format, const struct tm *restrict timeptr);
```

```
69720 CX size_t strptime_l(char *restrict s, size_t maxsize,
69721                  const char *restrict format, const struct tm *restrict timeptr,
69722                  locale_t locale);
```

69723 **DESCRIPTION**

69724 CX For *strptime()*: The functionality described on this reference page is aligned with the ISO C  
 69725 standard. Any conflict between the requirements described here and the ISO C standard is  
 69726 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

69727 The *strptime()* function shall place bytes into the array pointed to by *s* as controlled by the string  
 69728 pointed to by *format*. The application shall ensure that the format is a character string, beginning  
 69729 and ending in its initial shift state, if any. The format string consists of zero or more conversion  
 69730 specifications and ordinary characters.

69731 Each conversion specification is introduced by the '%' character after which the following  
 69732 appear in sequence:

- 69733 CX
- An optional flag:
    - 69734 0 The zero character ('0'), which specifies that the character used as the padding  
 69735 character is '0',
    - 69736 + The <plus-sign> character ('+'), which specifies that the character used as the  
 69737 padding character is '0', and that if and only if the field being produced consumes  
 69738 more than four bytes to represent a year (for %F, %G, or %Y) or more than two bytes to  
 69739 represent the year divided by 100 (for %C) then a leading <plus-sign> character shall  
 69740 be included if the year being processed is greater than or equal to zero or a leading  
 69741 <hyphen-minus> character ('-') shall be included if the year is less than zero.
  - 69742 The default padding character is unspecified.
  - An optional minimum field width. If the converted value, including any leading '+' or  
 69743 '-' sign, has fewer bytes than the minimum field width and the padding character is not  
 69744 the NUL character, the output shall be padded on the left (after any leading '+' or '-'  
 69745 sign) with the padding character.
  - An optional E or O modifier.
  - A terminating conversion specifier character that indicates the type of conversion to be  
 69746 applied.

69750 CX The results are unspecified if more than one flag character is specified, a flag character is  
 69751 specified without a minimum field width; a minimum field width is specified without a flag  
 69752 character; a modifier is specified with a flag or with a minimum field width; or if a minimum  
 69753 field width is specified for any conversion specifier other than C, F, G, or Y.

69754 All ordinary characters (including the terminating NUL character) are copied unchanged into  
 69755 the array. If copying takes place between objects that overlap, the behavior is undefined. No  
 69756 more than *maxsize* bytes are placed into the array. Each conversion specifier is replaced by  
 69757 appropriate characters as described in the following list. The appropriate characters are

- 69758 determined using the *LC\_TIME* category of the current locale and by the values of zero or more  
 69759 members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the  
 69760 description. If any of the specified values are outside the normal range, the characters stored are  
 69761 unspecified.
- 69762 CX The *strptime\_l()* function shall be equivalent to the *strptime()* function, except that the locale data  
 69763 used is from the locale represented by *locale*.
- 69764 Local timezone information shall be set as though *strptime()* called *tzset()*.
- 69765 The following conversion specifiers shall be supported:
- 69766 a Replaced by the locale's abbreviated weekday name. [*tm\_wday*]
- 69767 A Replaced by the locale's full weekday name. [*tm\_wday*]
- 69768 b Replaced by the locale's abbreviated month name. [*tm\_mon*]
- 69769 B Replaced by the locale's full month name. [*tm\_mon*]
- 69770 c Replaced by the locale's appropriate date and time representation. (See the Base  
 69771 Definitions volume of POSIX.1-2024, <**time.h**>.)
- 69772 C Replaced by the year divided by 100 and truncated to an integer, as a decimal number.  
 69773 [*tm\_year*]
- 69774 If a minimum field width is not specified:
- 69775 • If the year is between 0 and 9999 inclusive, two characters shall be placed into the  
 69776 array pointed to by *s*, including a leading '0' if there would otherwise be only a  
 69777 single digit.
  - 69778 CX • If the year is less than 0 or greater than 9999, the number of characters placed into  
 69779 the array pointed to by *s* shall be the number of digits and leading sign characters  
 69780 (if any) in the result of dividing the year by 100 and truncating, or two, whichever  
 69781 is greater.
- 69782 CX If a minimum field width is specified, the number of characters placed into the array  
 69783 pointed to by *s* shall be the number of digits and leading sign characters (if any) in the  
 69784 result of dividing the year by 100 and truncating, or the minimum field width,  
 69785 whichever is greater.
- 69786 d Replaced by the day of the month as a decimal number [01,31]. [*tm\_mday*]
- 69787 D Equivalent to *%m/%d/%y*. [*tm\_mon, tm\_mday, tm\_year*]
- 69788 e Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded  
 69789 by a space. [*tm\_mday*]
- 69790 F Equivalent to *%Y-%m-%d* if no flag and no minimum field width are specified. (For  
 69791 years between 1000 and 9999 inclusive this provides the ISO 8601-1:2019 standard  
 69792 complete representation, extended format date representation of a specific day.)  
 69793 [*tm\_year, tm\_mon, tm\_mday*]
- 69794 CX If a minimum field width of *x* is specified, the year shall be output as if by the *Y*  
 69795 specifier (described below) with whatever flag was given and a minimum field width  
 69796 of *x*−6. If *x* is less than 6, the behavior shall be as if *x* equalled 6.
- 69797 If the minimum field width is specified to be 10, and the year is four digits long, then  
 69798 the output string produced shall match the ISO 8601-1:2019 standard subclause 4.1.2.2  
 69799 complete representation, extended format date representation of a specific day. If a +

69800		flag is specified, a minimum field width of $x$ is specified, and $x-7$ bytes are sufficient to hold the digits of the year (not including any needed sign character), then the output shall match the ISO 8601-1:2019 standard subclause 4.1.2.4 complete representation, expanded format date representation of a specific day.
69801		
69802		
69803		
69804	g	Replaced by the last 2 digits of the week-based year (see below) as a decimal number [00,99]. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
69805		
69806	G	Replaced by the week-based year (see below) as a decimal number (for example, 1977). [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
69807		
69808	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by $s$ shall be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.
69809		
69810		
69811	h	Equivalent to %b. [ <i>tm_mon</i> ]
69812	H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [ <i>tm_hour</i> ]
69813	I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [ <i>tm_hour</i> ]
69814	j	Replaced by the day of the year as a decimal number [001,366]. [ <i>tm_yday</i> ]
69815	m	Replaced by the month as a decimal number [01,12]. [ <i>tm_mon</i> ]
69816	M	Replaced by the minute as a decimal number [00,59]. [ <i>tm_min</i> ]
69817	n	Replaced by a <newline>.
69818	p	Replaced by the locale's equivalent of either a.m. or p.m. [ <i>tm_hour</i> ]
69819	CX	r
69820		Replaced by the time in 12-hour clock notation; if the 12-hour format is not supported in the locale, this shall be either an empty string or the time in a 24-hour clock notation. In the POSIX locale this shall be equivalent to %I:%M:%S %p. [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]
69821		
69822	R	Replaced by the time in 24-hour notation (%H:%M). [ <i>tm_hour</i> , <i>tm_min</i> ]
69823	CX	s
69824		Replaced by the number of seconds since the Epoch as a decimal number, calculated as described for <i>mktime()</i> . [ <i>tm_year</i> , <i>tm_mon</i> , <i>tm_mday</i> , <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> , <i>tm_isdst</i> ]
69825	S	Replaced by the second as a decimal number [00,60]. [ <i>tm_sec</i> ]
69826	t	Replaced by a <tab>.
69827	T	Replaced by the time (%H:%M:%S). [ <i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i> ]
69828	u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [ <i>tm_wday</i> ]
69829		
69830	U	Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
69831		
69832		
69833	V	Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
69834		
69835		
69836		
69837		
69838	w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [ <i>tm_wday</i> ]
69839		



69840	W	Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [ <i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i> ]
69841		
69842		
69843	x	Replaced by the locale's appropriate date representation. (See the Base Definitions volume of POSIX.1-2024, < <b>time.h</b> >.)
69844		
69845	X	Replaced by the locale's appropriate time representation. (See the Base Definitions volume of POSIX.1-2024, < <b>time.h</b> >.)
69846		
69847	y	Replaced by the last two digits of the year as a decimal number [00,99]. [ <i>tm_year</i> ]
69848	Y	Replaced by the year as a decimal number (for example, 1997). [ <i>tm_year</i> ]
69849	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by <i>s</i> shall be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.
69850		
69851		
69852	z	Replaced by the offset from UTC in the ISO 8601-1:2019 standard format (+hhmm or -hhmm), or by no characters if no timezone is determinable. For example, "-0430" means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight saving time offset is used. If <i>tm_isdst</i> is negative, no characters are returned.
69853		
69854	CX	[ <i>tm_isdst</i> , <i>tm_gmtoff</i> ]
69855		
69856		
69857	CX	
69858	Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [ <i>tm_isdst</i> , <i>tm_zone</i> ]
69859	CX	
69860	%	Replaced by %.
69861		If a conversion specification does not correspond to any of the above, the behavior is undefined.
69862	CX	If a <b>struct tm</b> broken-down time structure is created by <i>localtime()</i> or <i>localtime_r()</i> , or modified by <i>mktime()</i> , and the value of <i>TZ</i> is subsequently modified, the results of the %Z and %z <i>strftime()</i> conversion specifiers are undefined, when <i>strftime()</i> is called with such a broken-down time structure.
69863		
69864		
69865		
69866		If a <b>struct tm</b> broken-down time structure is created or modified by <i>gmtime()</i> or <i>gmtime_r()</i> , it is unspecified whether the result of the %Z and %z conversion specifiers shall refer to UTC or the current local timezone, when <i>strftime()</i> is called with such a broken-down time structure.
69867		
69868		

### 69869 Modified Conversion Specifiers

69870		Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in XBD Section 7.3.5, on page 152), the behavior shall be as if the unmodified conversion specification were used.
69871		
69872		
69873		
69874		
69875	%EC	Replaced by the locale's alternative appropriate date and time representation.
69876	%EC	Replaced by the name of the base year (period) in the locale's alternative representation.
69877		
69878	%Ex	Replaced by the locale's alternative date representation.
69879	%EX	Replaced by the locale's alternative time representation.

69880		%Ey	Replaced by the offset from %EC (year only) in the locale's alternative representation.
69881		%EY	Replaced by the full alternative year representation.
69882	CX	%Ob	Replaced by the locale's abbreviated alternative month name.
69883	CX	%OB	Replaced by the locale's alternative appropriate full month name.
69884		%Od	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading <space> characters.
69885			
69886			
69887		%Oe	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading <space> characters.
69888			
69889		%OH	Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.
69890		%OI	Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.
69891		%Om	Replaced by the month using the locale's alternative numeric symbols.
69892		%OM	Replaced by the minutes using the locale's alternative numeric symbols.
69893		%OS	Replaced by the seconds using the locale's alternative numeric symbols.
69894		%Ou	Replaced by the weekday as a number in the locale's alternative representation (Monday=1).
69895			
69896		%OU	Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.
69897			
69898		%OV	Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.
69899			
69900		%Ow	Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
69901			
69902		%OW	Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
69903			
69904		%Oy	Replaced by the year (offset from %C) using the locale's alternative numeric symbols.
69905			%g, %G, and %v give values according to the ISO 8601-1:2019 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, %G is replaced by 1998 and %v is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, %G is replaced by 1998 and %v is replaced by 01.
69906			
69907			
69908			
69909			
69910			
69911			
69912			
69913			If a conversion specifier is not one of the above, the behavior is undefined.
69914	CX		The behavior is undefined if the <i>locale</i> argument to <i>strptime_l()</i> is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.
69915			
69916		<b>RETURN VALUE</b>	
69917			If successful, these functions shall return the number of bytes placed into the array pointed to by <i>s</i> , not including the terminating NUL character. If successful, <i>errno</i> shall not be changed.
69918	CX		Otherwise, 0 shall be returned, <i>errno</i> shall be set to indicate the error, and the contents of the array are unspecified.
69919	CX		
69920			

69921 **ERRORS**

69922	CX	These functions shall fail if:
69923	[ERANGE]	The total number of resulting bytes including the terminating NUL character is more than <i>maxsize</i> .
69924		
69925		These functions may fail if:
69926	[EINVAL]	The <i>format</i> string includes a %s conversion and the number of seconds since the Epoch would be negative.
69927		
69928	[EOVERFLOW]	The <i>format</i> string includes a %s conversion and the number of seconds since the Epoch cannot be represented in a <b>time_t</b> .
69929		

69930 **EXAMPLES**69931 **Getting a Localized Date String**

69932 The following example first sets the locale to the user's default. The locale information will be  
 69933 used in the *nl\_langinfo()* and *strptime()* functions. The *nl\_langinfo()* function returns the localized  
 69934 date string which specifies how the date is laid out. The *strptime()* function takes this  
 69935 information and, using the **tm** structure for values, places the date and time information into  
 69936 *datestring*.

```
69937 #include <time.h>
69938 #include <locale.h>
69939 #include <langinfo.h>
69940 ...
69941 struct tm *tm;
69942 char datestring[256];
69943 ...
69944 setlocale (LC_ALL, "");
69945 ...
69946 strptime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
69947 ...
```

69948 **APPLICATION USAGE**

69949 A return value of 0 may indicate either success or failure if a format is the empty string or  
 69950 consists of conversion specifications such as %p or %Z that are replaced by no characters in the  
 69951 current locale or because of the current setting of *tzname[]*, respectively. To distinguish between  
 69952 success and failure when *strptime()* returns 0, an application can set *errno* to 0 before calling  
 69953 *strptime()* and test whether *errno* is 0 afterwards.

69954 The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

69955 Some of the conversion specifications are duplicates of others. They are included for  
 69956 compatibility with *nl\_cxtime()* and *nl\_ascxtime()*, which were published in Issue 2.

69957 The %C, %F, %G, and %Y format specifiers in *strptime()* always print full values, but the *strptime()*  
 69958 %C, %F, and %Y format specifiers only scan two digits (assumed to be the first two digits of a  
 69959 four-digit year) for %C and four digits (assumed to be the entire (four-digit) year) for %F and %Y.  
 69960 This mimics the behavior of *printf()* and *scanf()*; that is:

```
69961 printf("%2d", x = 1000);
69962 prints "1000", but:
69963 scanf("%2d", &x);
```

69964 when given "1000" as input will only store 10 in *x*). Applications using extended ranges of  
 69965 years must be sure that the number of digits specified for scanning years with *strptime()* matches  
 69966 the number of digits that will actually be present in the input stream. Historic implementations  
 69967 of the *%Y* conversion specification (with no flags and no minimum field width) produced  
 69968 different output formats. Some always produced at least four digits (with 0 fill for years from 0  
 69969 through 999) while others only produced the number of digits present in the year (with no fill  
 69970 and no padding). These two forms can be produced with the '0' flag and a minimum field  
 69971 width options using the conversions specifications *%04Y* and *%01Y*, respectively. Similarly,  
 69972 because *%Y* is part of *%F*, field widths of 10 and 7 (*%010F*, *%07F*), respectively, produce the same  
 69973 effect in the year portion of the *%F* conversion result.

69974 In the past, the C and POSIX standards specified that *%F* produced an ISO 8601-1:2019 standard  
 69975 date format, but didn't specify which one. For years in the range [1000,9999], POSIX.1-2024  
 69976 requires that the output produced match the ISO 8601-1:2019 standard complete representation  
 69977 extended format (YYYY-MM-DD) and for years greater than 9999 produce output that matches  
 69978 the ISO 8601-1:2019 standard expanded representation extended format *±YYYYY-MM-DD*). For  
 69979 years less than 1000, *%F* is not required to produce an ISO 8601-1:2019 standard format when  
 69980 used without specifying at least a minimum field width. As stated above, some implementations  
 69981 pad *%Y* conversions with zeros to four digits, in which case *%F* produces an ISO 8601-1:2019  
 69982 standard format; other implementations do not pad *%Y* with zeros, in which case *%F* does not  
 69983 produce an ISO 8601-1:2019 standard format. To fully meet ISO 8601-1:2019 standard  
 69984 requirements, the producer and consumer must agree on a date format that has a specific  
 69985 number of bytes reserved to hold the characters used to represent the years that is sufficiently  
 69986 large to hold all values that will be shared. For example, the  *%+13F* conversion specification will  
 69987 produce output matching the format "*±YYYYYY-MM-DD*" (a leading '+' or '-' sign; a six-digit,  
 69988 0-filled year; a '-' ; a two-digit, leading 0-filled month; another '-' ; and the two-digit, leading  
 69989 0-filled day within the month).

69990 Note that if the year being printed is greater than 9999, the resulting string from the unadorned  
 69991 *%F* conversion specifications will not conform to the ISO 8601-1:2019 standard extended format,  
 69992 complete representation for a date and will instead be an extended format, expanded  
 69993 representation (presumably without the required agreement between the date's producer and  
 69994 consumer).

69995 In the C or POSIX locale, the *E* and *O* modifiers are ignored and the replacement strings for the  
 69996 following specifiers are:

69997	<i>%a</i>	The first three characters of <i>%A</i> .
69998	<i>%A</i>	One of Sunday, Monday, ..., Saturday.
69999	<i>%b</i>	The first three characters of <i>%B</i> .
70000	<i>%B</i>	One of January, February, ..., December.
70001	<i>%c</i>	Equivalent to <i>%a %b %e %T %Y</i> .
70002	<i>%p</i>	One of AM or PM.
70003	<i>%r</i>	Equivalent to <i>%I:%M:%S %p</i> .
70004	<i>%x</i>	Equivalent to <i>%m/%d/%y</i> .
70005	<i>%X</i>	Equivalent to <i>%T</i> .
70006	<i>%Z</i>	Implementation-defined.

## 70007 RATIONALE

70008 The %Y conversion specification to *strptime()* was frequently assumed to be a four-digit year, but  
70009 the ISO C standard does not specify that %Y is restricted to any subset of allowed values from the  
70010 *tm\_year* field. Similarly, the %C conversion specification was assumed to be a two-digit field and  
70011 the first part of the output from the %F conversion specification was assumed to be a four-digit  
70012 field. Since *tm\_year* is a signed **int** with a width of at least 32 bits and **time\_t** is required to have a  
70013 width of at least 64 bits (in conforming programming environments), these assumptions no  
70014 longer hold.

70015 POSIX.1-2024 now allows the format specifications %0xC, %0xF, %0xG, and %0xY (where 'x' is  
70016 a string of decimal digits used to specify printing and scanning of a string of *x* decimal digits)  
70017 with leading zero fill characters. Allowing applications to set the field width enables them to  
70018 agree on the number of digits to be printed and scanned in the ISO 8601-1:2019 standard  
70019 expanded representation of a year (for %F, %G, and %Y) or all but the last two digits of the year  
70020 (for %C). This is based on a feature in some versions of GNU **libc**'s *strptime()*. The GNU version  
70021 allows specifying space, zero, or no-fill characters in *strptime()* format strings, but does not allow  
70022 any flags to be specified in *strptime()* format strings. These implementations also allow these  
70023 flags to be specified for any numeric field. POSIX.1-2024 only requires the zero fill flag ('0') and  
70024 only requires that it be recognized when processing %C, %F, %G, and %Y specifications when a  
70025 minimum field width is also specified. The '0' flag is the only flag needed to produce and scan  
70026 the ISO 8601-1:2019 standard year fields using the extended format forms. POSIX.1-2024 also  
70027 allows applications to specify the same flag and field width specifiers to be used in both  
70028 *strptime()* and *strptime()* format strings for symmetry. Systems may provide other flag characters  
70029 and may accept flags in conjunction with conversion specifiers other than %C, %F, %G, and %Y;  
70030 but portable applications cannot depend on such extensions.

70031 POSIX.1-2024 now also allows the format specifications %+xC, %+xF, %+xG, and %+xY (where  
70032 'x' is a string of decimal digits used to specify printing and scanning of a string of 'x' decimal  
70033 digits) with leading zero fill characters and a leading '+' sign character if the year being  
70034 converted is more than four digits or a minimum field width is specified that allows room for  
70035 more than four digits for the year. This allows date providers and consumers to agree on a  
70036 specific number of digits to represent a year as required by the ISO 8601-1:2019 standard  
70037 expanded representation formats. The expanded representation formats all require the year to  
70038 begin with a leading '+' or '-' sign. (All of these specifiers can also provide a leading '-'  
70039 sign for negative years. Since negative years and the year 0 don't fit well with the Gregorian or  
70040 Julian calendars, the normal ranges of dates start with year 1. The ISO C standard allows *tm\_year*  
70041 to assume values corresponding to years before year 1, but the use of such years provided  
70042 unspecified results.)

70043 Some earlier version of this standard specified that applications wanting to use *strptime()* to scan  
70044 dates and times printed by *strptime()* should provide non-digit characters between fields to  
70045 separate years from months and days. It also supported %F to print and scan the  
70046 ISO 8601-1:2019 standard extended format, complete representation date for years 1 through  
70047 9999 (i.e., YYYY-MM-DD). However, many applications were written to print (using *strptime()*)  
70048 and scan (using *strptime()*) dates written using the basic format complete representation (four-  
70049 digit years) and truncated representation (two-digit years) specified by the ISO 8601-1:2019  
70050 standard representation of dates and times which do not have any separation characters  
70051 between fields. The ISO 8601-1:2019 standard also specifies basic format expanded  
70052 representation where the creator and consumer of these fields agree beforehand to represent  
70053 years as leading zero-filled strings of an agreed length of more than four digits to represent a  
70054 year (again with no separation characters when year, month, and day are all displayed).  
70055 Applications producing and consuming expanded representations are encouraged to use the  
70056 '+' flag and an appropriate maximum field width to scan the year including the leading sign.

70057 Note that even without the '+' flag, years less than zero may be represented with a leading  
70058 <hyphen-minus> for %F, %G, and %Y conversion specifications. Using negative years results in  
70059 unspecified behavior.

70060 If a format specification %+xF with the field width *x* greater than 11 is specified and the width is  
70061 large enough to display the full year, the output string produced will match the ISO 8601-1:2019  
70062 standard subclause 4.1.2.4 expanded representation, extended format date representation for a  
70063 specific day. (For years in the range [1,99999], %+12F is sufficient for an agreed five-digit year  
70064 with a leading sign using the ISO 8601-1:2019 standard expanded representation, extended  
70065 format for a specific day "+YYYY-MM-DD".) Note also that years less than 0 may produce a  
70066 leading <hyphen-minus> character ('-') when using %Y or %C whether or not the '0' or '+'  
70067 flags are used.

70068 The difference between the '0' flag and the '+' flag is whether the leading '+' character will  
70069 be provided for years >9999 as required for the ISO 8601-1:2019 standard extended  
70070 representation format containing a year. For example:

	Year	Conversion Specification	strptime() Output	strptime() Scan Back
70071	1970	%Y	1970	1970
70072	1970	%+4Y	1970	1970
70073	27	%Y	27 or 0027	27
70074	270	%Y	270 or 0270	270
70075	270	%+4Y	0270	270
70076	17	%C%y	0017	17
70077	270	%C%y	0270	270
70078	12345	%Y	12345	1234*
70079	12345	%+4Y	+12345	123*
70080	12345	%05Y	12345	12345
70081	270	%+5Y or %+3C%y	+0270	270
70082	12345	%+5Y or %+3C%y	+12345	1234*
70083	12345	%06Y or %04C%y	012345	12345
70084	12345	%+6Y or %+4C%y	+12345	12345
70085	123456	%08Y or %06C%y	00123456	123456
70086	123456	%+8Y or %+6C%y	+0123456	123456

70089 In the cases above marked with a \* in the *strptime()* scan back field, the implied or specified  
70090 number of characters scanned by *strptime()* was less than the number of characters output by  
70091 *strptime()* using the same format; so the remaining digits of the year were dropped when the  
70092 output date produced by *strptime()* was scanned back in by *strptime()*.

#### 70093 FUTURE DIRECTIONS

70094 None.

#### 70095 SEE ALSO

70096 *asctime()*, *clock()*, *ctime()*, *difftime()*, *futimens()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*,  
70097 *strptime()*, *time()*, *tzset()*, *uselocale()*

70098 XBD Section 7.3.5 (on page 152), <[time.h](#)>

70099 **CHANGE HISTORY**

70100 First released in Issue 3.

70101 **Issue 5**70102 The description of %OV is changed to be consistent with %V and defines Monday as the first day  
70103 of the week.

70104 The description of %Oy is clarified.

70105 **Issue 6**

70106 Extensions beyond the ISO C standard are marked.

70107 The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from  
70108 ``Otherwise, it is week 53 of the previous year, and the next week is week 1'' to ``Otherwise, it is  
70109 the last week of the previous year, and the next week is week 1''.70110 The following new requirements on POSIX implementations derive from alignment with the  
70111 Single UNIX Specification:

- 70112
- The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.
  - The modified conversion specifiers are added for consistency with the ISO POSIX-2  
70113 standard *date* utility.
- 70114

70115 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 70116
- The *strptime()* prototype is updated.
  - The DESCRIPTION is extensively revised.
  - The %z conversion specifier is added.
- 70117
- 
- 70118

70119 An example is added.

70120 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied.

70121 **Issue 7**

70122 Austin Group Interpretation 1003.1-2001 #163 is applied.

70123 The *strptime\_1()* function is added from The Open Group Technical Standard, 2006, Extended API  
70124 Set Part 4.70125 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0605 [283], XSH/TC1-2008/0606 [283],  
70126 XSH/TC1-2008/0607 [193], and XSH/TC1-2008/0608 [193] are applied.70127 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0340 [584], XSH/TC2-2008/0341 [796],  
70128 XSH/TC2-2008/0342 [584], and XSH/TC2-2008/0343 [584] are applied.70129 **Issue 8**70130 Austin Group Defects 169, 1386, and 1612 are applied, adding the s conversion and requiring  
70131 *errno* to be unchanged on success and set on error.

70132 Austin Group Defects 258 and 1166 are applied, adding the OB and Ob modified conversions.

70133 Austin Group Defect 472 is applied, changing the description of the C conversion.

70134 Austin Group Defect 739 is applied, changing the %F conversion to match the ISO C standard  
70135 when no flag and no minimum field width are specified.70136 Austin Group Defect 1125 is applied, changing ``Local timezone information is used'' to ``Local  
70137 timezone information shall be set''.

70138 Austin Group Defect 1253 is applied, changing ``daylight savings'' to ``daylight saving''.

- 70139 Austin Group Defect 1307 is applied, changing the `r` conversion in relation to locales that do not  
70140 support the 12-hour clock format.
- 70141 Austin Group Defects 1313 and 1354 are applied, changing text relating to the ISO 8601-1:2019  
70142 standard in the APPLICATION USAGE section.
- 70143 Austin Group Defect 1462 is applied, changing the RATIONALE section.
- 70144 Austin Group Defect 1533 is applied, adding `tm_gmtoff` and `tm_zone` to the `tm` structure.
- 70145 Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to  
70146 ensure that the format is a character string, beginning and ending in its initial shift state, if any.



70147 **NAME**

70148 strlcat, strlcpy — size-bounded string concatenation and copying

70149 **SYNOPSIS**

```
70150 CX #include <string.h>
70151     size_t strlcat(char *restrict dst, const char *restrict src,
70152                  size_t dstsize);
70153     size_t strlcpy(char *restrict dst, const char *restrict src,
70154                  size_t dstsize);
```

70155 **DESCRIPTION**

70156 The *strlcpy()* and *strlcat()* functions copy and concatenate strings, stopping when either a NUL  
 70157 terminator in the source string is encountered or the specified full size of the destination buffer is  
 70158 reached. They NUL terminate the result if there is room. The application should ensure that  
 70159 room for the NUL terminator is included in *dstsize*.

70160 The *strlcpy()* function shall copy not more than *dstsize* – 1 bytes from the string pointed to by *src*  
 70161 to the array pointed to by *dst*; a NUL byte in *src* and bytes that follow it shall not be copied. A  
 70162 terminating NUL byte shall be appended to the result, unless *dstsize* is 0. If copying takes place  
 70163 between objects that overlap, the behavior is undefined.

70164 The *strlcat()* function shall append not more than *dstsize* – *strlen(dst)* – 1 bytes from the string  
 70165 pointed to by *src* to the end of the string pointed to by *dst*; a NUL byte in *src* and bytes that  
 70166 follow it shall not be appended. The initial byte of *src* shall overwrite the NUL byte at the end of  
 70167 *dst*. A terminating NUL byte shall be appended to the result, unless its location would be at or  
 70168 beyond *dst* + *dstsize*. If copying takes place between objects that overlap, the behavior is  
 70169 undefined.

70170 The *strlcpy()* and *strlcat()* functions shall not change the setting of *errno* on valid input.

70171 **RETURN VALUE**

70172 Upon successful completion, the *strlcpy()* function shall return the length of the string pointed to  
 70173 by *src*; that is, the number of bytes in the string, not including the terminating NUL byte.

70174 Upon successful completion, the *strlcat()* function shall return the initial length of the string (if  
 70175 any) pointed to by *dst*, as limited by *dstsize*, plus the length of the string pointed to by *src*; that is,  
 70176 the value that would be returned by *strlen(dst, dstsize)* + *strlen(src)* before the *strlcat()* call.

70177 No return values are reserved to indicate an error.

70178 **ERRORS**

70179 No errors are defined.

70180 **EXAMPLES**

70181 The following example detects truncation while combining a path prefix (including trailing  
 70182 <slash>) and a filename to produce a portable pathname:

```
70183 char *prefix, *filenam, pathnam[_POSIX_PATH_MAX];
70184
70185 if (strlcpy(pathnam, prefix, sizeof pathnam) >= sizeof pathnam ||
70186     strlcat(pathnam, filenam, sizeof pathnam) >= sizeof pathnam)
70187 {
70188     // truncation occurred
70189     ...
70190 }
```

70190 This code ensures there is room for the NUL terminator by:

- 70191 • Calling *strncpy()* with a non-zero *dstsize* argument.
- 70192 • Only calling *strlcat()* if the return value of *strncpy()* indicated that truncation did not occur.

**70193 APPLICATION USAGE**

70194 The return value of the *strncpy()* and *strlcat()* functions follows the same convention as  
70195 *snprintf()*; that is, they return the total length of the string they tried to create. If the return value  
70196 is greater than or equal to *dstsize*, the output string has been truncated.

**70197 RATIONALE**

70198 None.

**70199 FUTURE DIRECTIONS**

70200 None.

**70201 SEE ALSO**

70202 *fprintf()*, *strlen()*, *strncat()*, *strncpy()*, *wcslcat()*

70203 XBD <**string.h**>

**70204 CHANGE HISTORY**

70205 First released in Issue 8.

70206 **NAME**

70207            strlen, strlen — get length of fixed size string

70208 **SYNOPSIS**

70209            #include &lt;string.h&gt;

70210            size\_t strlen(const char \*s);

70211 CX         size\_t strnlen(const char \*s, size\_t maxlen);

70212 **DESCRIPTION**70213 CX         For *strlen()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.70216         The *strlen()* function shall compute the number of bytes in the string to which *s* points, not including the terminating NUL character.70218 CX         The *strnlen()* function shall compute the smaller of the number of bytes in the array to which *s* points, not including any terminating NUL character, or the value of the *maxlen* argument. The *strnlen()* function shall never examine more than *maxlen* bytes of the array pointed to by *s*.70221 CX         The *strlen()* and *strnlen()* functions shall not change the setting of *errno* on valid input.70222 **RETURN VALUE**70223         The *strlen()* function shall return the length of *s*; no return value shall be reserved to indicate an error.70225 CX         The *strnlen()* function shall return the number of bytes preceding the first null byte in the array to which *s* points, if *s* contains a null byte within the first *maxlen* bytes; otherwise, it shall return *maxlen*.70228 **ERRORS**

70229         No errors are defined.

70230 **EXAMPLES**70231            **Getting String Lengths**70232         The following example sets the maximum length of *key* and *data* by using *strlen()* to get the lengths of those strings.

```

70234         #include <string.h>
70235         ...
70236         struct element {
70237             char *key;
70238             char *data;
70239         };
70240         ...
70241         char *key, *data;
70242         int len;

70243         *keylength = *datalength = 0;
70244         ...
70245         if ((len = strlen(key)) > *keylength)
70246             *keylength = len;
70247         if ((len = strlen(data)) > *datalength)
70248             *datalength = len;
70249         ...
```

70250 **APPLICATION USAGE**

70251 None.

70252 **RATIONALE**

70253 None.

70254 **FUTURE DIRECTIONS**

70255 None.

70256 **SEE ALSO**70257 *strlcat()*, *wcslen()*70258 XBD <**string.h**>70259 **CHANGE HISTORY**

70260 First released in Issue 1. Derived from Issue 1 of the SVID.

70261 **Issue 5**70262 The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not  
70263 *s* itself as was previously stated.70264 **Issue 7**70265 The *strlen()* function is added from The Open Group Technical Standard, 2006, Extended API  
70266 Set Part 1.

70267 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0344 [560] is applied.

70268 **Issue 8**70269 Austin Group Defect 448 is applied, adding a requirement that *strlen()* and *strlen()* do not  
70270 change the setting of *errno* on valid input.70271 Austin Group Defect 986 is applied, adding *strlcat()* to the SEE ALSO section.

70272 **NAME**

70273 strncasecmp, strncasecmp\_l — case-insensitive string comparisons

70274 **SYNOPSIS**

70275 #include &lt;strings.h&gt;

70276 int strncasecmp(const char \*s1, const char \*s2, size\_t n);

70277 int strncasecmp\_l(const char \*s1, const char \*s2,

70278 size\_t n, locale\_t locale);

70279 **DESCRIPTION**70280 Refer to *strcasecmp()*.

70281 **NAME**

70282 strncat — concatenate a string with part of another

70283 **SYNOPSIS**

70284 #include &lt;string.h&gt;

70285 char \*strncat(char \*restrict s1, const char \*restrict s2, size\_t n);

70286 **DESCRIPTION**70287 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
70288 conflict between the requirements described here and the ISO C standard is unintentional. This  
70289 volume of POSIX.1-2024 defers to the ISO C standard.70290 The *strncat()* function shall append not more than *n* bytes (a NUL character and bytes that  
70291 follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by  
70292 *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. A terminating NUL  
70293 character is always appended to the result. If copying takes place between objects that overlap,  
70294 the behavior is undefined.70295 CX The *strncat()* function shall not change the setting of *errno* on valid input.70296 **RETURN VALUE**70297 The *strncat()* function shall return *s1*; no return value shall be reserved to indicate an error.70298 **ERRORS**

70299 No errors are defined.

70300 **EXAMPLES**

70301 None.

70302 **APPLICATION USAGE**

70303 None.

70304 **RATIONALE**

70305 None.

70306 **FUTURE DIRECTIONS**

70307 None.

70308 **SEE ALSO**70309 [strcat\(\)](#), [strlcat\(\)](#)70310 XBD [<string.h>](#)70311 **CHANGE HISTORY**

70312 First released in Issue 1. Derived from Issue 1 of the SVID.

70313 **Issue 6**70314 The *strncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.70315 **Issue 8**70316 Austin Group Defect 448 is applied, adding a requirement that *strncat()* does not change the  
70317 setting of *errno* on valid input.70318 Austin Group Defect 986 is applied, adding *strlcat()* to the SEE ALSO section.

70319 **NAME**

70320 strncmp — compare part of two strings

70321 **SYNOPSIS**

70322 #include &lt;string.h&gt;

70323 int strncmp(const char \*s1, const char \*s2, size\_t n);

70324 **DESCRIPTION**

70325 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
70326 conflict between the requirements described here and the ISO C standard is unintentional. This  
70327 volume of POSIX.1-2024 defers to the ISO C standard.

70328 The *strncmp()* function shall compare not more than *n* bytes (bytes that follow a NUL character  
70329 are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

70330 The sign of a non-zero return value is determined by the sign of the difference between the  
70331 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings  
70332 being compared.

70333 CX The *strncmp()* function shall not change the setting of *errno* on valid input.

70334 **RETURN VALUE**

70335 Upon successful completion, *strncmp()* shall return an integer greater than, equal to, or less than  
70336 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the  
70337 possibly null-terminated array pointed to by *s2* respectively.

70338 **ERRORS**

70339 No errors are defined.

70340 **EXAMPLES**

70341 None.

70342 **APPLICATION USAGE**

70343 None.

70344 **RATIONALE**

70345 None.

70346 **FUTURE DIRECTIONS**

70347 None.

70348 **SEE ALSO**70349 [strcmp\(\)](#)70350 XBD [<string.h>](#)70351 **CHANGE HISTORY**

70352 First released in Issue 1. Derived from Issue 1 of the SVID.

70353 **Issue 6**

70354 Extensions beyond the ISO C standard are marked.

70355 **Issue 8**70356 Austin Group Defect 448 is applied, adding a requirement that *strncmp()* does not change the  
70357 setting of *errno* on valid input.

70358 **NAME**

70359           stpnncpy, strncpy — copy fixed length string, returning a pointer to the array end

70360 **SYNOPSIS**

70361           #include &lt;string.h&gt;

70362 CX       char \*stpnncpy(char \*restrict s1, const char \*restrict s2, size\_t n);

70363       char \*strncpy(char \*restrict s1, const char \*restrict s2, size\_t n);

70364 **DESCRIPTION**70365 CX       For *strncpy()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.70366 CX       The *stpnncpy()* and *strncpy()* functions shall copy not more than *n* bytes (bytes that follow a NUL character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.70367       If the array pointed to by *s2* is a string that is shorter than *n* bytes, NUL characters shall be appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

70368       If copying takes place between objects that overlap, the behavior is undefined.

70369 CX       The *strncpy()* and *stpnncpy()* functions shall not change the setting of *errno* on valid input.70374 **RETURN VALUE**70375 CX       If a NUL character is written to the destination, the *stpnncpy()* function shall return the address of the first such NUL character. Otherwise, it shall return *&s1[n]*.70376       The *strncpy()* function shall return *s1*.

70377       No return values are reserved to indicate an error.

70379 **ERRORS**

70380       No errors are defined.

70381 **EXAMPLES**

70382       None.

70383 **APPLICATION USAGE**70384       Applications must provide the space in *s1* for the *n* bytes to be transferred, as well as ensure that the *s2* and *s1* arrays do not overlap.

70385       Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

70386       If there is no NUL character byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-terminated.70387 **RATIONALE**

70388       None.

70389 **FUTURE DIRECTIONS**

70390       None.

70391 **SEE ALSO**70392       [strcpy\(\)](#), [strlcat\(\)](#), [wcsncpy\(\)](#)70393       XBD [<string.h>](#)



70397 **CHANGE HISTORY**

70398 First released in Issue 1. Derived from Issue 1 of the SVID.

70399 **Issue 6**

70400 The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

70401 **Issue 7**

70402 The *stpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

70404 **Issue 8**

70405 Austin Group Defect 448 is applied, adding a requirement that *strncpy()* and *stpncpy()* do not change the setting of *errno* on valid input.

70407 Austin Group Defect 986 is applied, adding *strlcat()* to the SEE ALSO section.

70408 **NAME**

70409           strndup — duplicate a specific number of bytes from a string

70410 **SYNOPSIS**

70411 CX       #include <string.h>

70412       char \*strndup(const char \*s, size\_t size);

70413 **DESCRIPTION**

70414       Refer to *strdup()*.

70415 **NAME**

70416 strnlen — get length of fixed size string

70417 **SYNOPSIS**

```
70418 CX #include <string.h>  
70419 size_t strnlen(const char *s, size_t maxlen);
```

70420 **DESCRIPTION**70421 Refer to *strlen()*.

70422 **NAME**

70423 strpbrk — scan a string for a byte

70424 **SYNOPSIS**

70425 #include &lt;string.h&gt;

70426 char \*strpbrk(const char \*s1, const char \*s2);

70427 **DESCRIPTION**

70428 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
70429 conflict between the requirements described here and the ISO C standard is unintentional. This  
70430 volume of POSIX.1-2024 defers to the ISO C standard.

70431 The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte  
70432 from the string pointed to by *s2*.

70433 CX The *strpbrk()* function shall not change the setting of *errno* on valid input.

70434 **RETURN VALUE**

70435 Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no  
70436 byte from *s2* occurs in *s1*.

70437 **ERRORS**

70438 No errors are defined.

70439 **EXAMPLES**

70440 None.

70441 **APPLICATION USAGE**

70442 None.

70443 **RATIONALE**

70444 None.

70445 **FUTURE DIRECTIONS**

70446 None.

70447 **SEE ALSO**70448 [strchr\(\)](#), [strrchr\(\)](#)70449 XBD [<string.h>](#)70450 **CHANGE HISTORY**

70451 First released in Issue 1. Derived from Issue 1 of the SVID.

70452 **Issue 8**

70453 Austin Group Defect 448 is applied, adding a requirement that *strpbrk()* does not change the  
70454 setting of *errno* on valid input.

70455 **NAME**

70456 strptime — date and time conversion

70457 **SYNOPSIS**

```
70458 XSI      #include <time.h>
70459      char *strptime(const char *restrict buf, const char *restrict format,
70460                    struct tm *restrict tm);
```

70461 **DESCRIPTION**

70462 The *strptime()* function shall convert the character string pointed to by *buf* to values which are  
 70463 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

70464 The application shall ensure that the format is a character string, beginning and ending in its  
 70465 initial shift state, if any. The format is composed of zero or more directives. Each directive is  
 70466 composed of one of the following: one or more white-space bytes; an ordinary character (neither  
 70467 '%' nor a white-space byte); or a conversion specification.

70468 Each conversion specification is introduced by the '%' character after which the following  
 70469 appear in sequence:

- 70470 • An optional flag, the zero character ('0') or the <plus-sign> character ('+'), which is  
 70471 ignored.
- 70472 • An optional field width. If a field width is specified, it shall be interpreted as a string of  
 70473 decimal digits that determine the maximum number of bytes converted for the conversion  
 70474 rather than the number of bytes specified below in the description of the conversion  
 70475 specifiers.
- 70476 • An optional E or O modifier.
- 70477 • A terminating conversion specifier character that indicates the type of conversion to be  
 70478 applied.

70479 The conversions are determined using the *LC\_TIME* category of the current locale. The  
 70480 application shall ensure that there are white-space bytes or other non-alphanumeric bytes  
 70481 between any two conversion specifications unless all of the adjacent conversion specifications  
 70482 convert a known, fixed number of characters. In the following list, the maximum number of  
 70483 characters scanned (excluding the one matching the next directive) is as follows:

- 70484 • If a maximum field width is specified, then that number
- 70485 • Otherwise, the pattern "{x}" indicates that the maximum is *x*
- 70486 • Otherwise, the pattern "[x,y]" indicates that the value shall fall within the range given  
 70487 (both bounds being inclusive), and the maximum number of characters scanned shall be  
 70488 the maximum required to represent any value in the range without leading zeros and  
 70489 without a leading <plus-sign>

70490 The following conversion specifiers are supported.

70491 The results are unspecified if a modifier is specified with a flag or with a minimum field width,  
 70492 or if a field width is specified for any conversion specifier other than C, F, G, Y, or Z.

- 70493 a The day of the week, using the locale's weekday names; either the abbreviated or full  
 70494 name can be specified. The *tm\_wday* member of the **tm** structure pointed to by *tm* shall  
 70495 be set to the corresponding day of the week number (Sunday=0).

70496	A	Equivalent to %a.
70497	b	The month, using the locale's month names; either the abbreviated or full version of either the default or the alternative month name can be specified. The <i>tm_mon</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to the corresponding month number.
70498		
70499		
70500		
70501	c	Replaced by the locale's appropriate date and time representation. The members of the <b>tm</b> structure pointed to by <i>tm</i> shall be set as specified for the conversions present in the locale's <b>d_t_fmt</b> value.
70502		
70503		
70504	C	All but the last two digits of the year {2}; leading zeros shall be permitted but shall not be required. A leading '+' or '-' character shall be permitted before any leading zeros but shall not be required. The <i>tm_year</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to the number formed by appending the last two digits of the year to these digits, minus 1900. If a <i>y</i> conversion is also performed, the last two digits of the year shall be those processed by the <i>y</i> conversion; otherwise, they shall be 00.
70505		
70506		
70507		
70508		
70509		
70510	d	The day of the month [01,31]; leading zeros shall be permitted but shall not be required. The <i>tm_mday</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number.
70511		
70512	D	Equivalent to %m/%d/%y.
70513	e	Equivalent to %d.
70514	F	This specifier is similar to %Y-%m-%d where the characters up to the first <hyphen-minus> separator shall be converted as for %Y but with unlimited field width, the characters between the two <hyphen-minus> separators shall be converted as for %m, and the characters after the last <hyphen-minus> separator shall be converted as for %d. If a field width is specified, each of the %Y, %m, and %d conversions shall not convert any characters past the overall %F field width. The members of the <b>tm</b> structure pointed to by <i>tm</i> shall be set as specified for the Y, m, and d conversions.
70515		
70516		
70517		
70518		
70519		
70520		
70521		
70522	g	The last 2 digits of the week-based year (see below) as a decimal number (for example, 77). Leading zeros shall be permitted but shall not be required. A leading '+' or '-' character shall be permitted before any leading zeros but shall not be required. The effect of this year, if any, on the <b>tm</b> structure pointed to by <i>tm</i> is unspecified.
70523		
70524		
70525	G	The week-based year (see below) as a decimal number (for example, 1977). Leading zeros shall be permitted but shall not be required. A leading '+' or '-' character shall be permitted before any leading zeros but shall not be required. The effect of this year, if any, on the <b>tm</b> structure pointed to by <i>tm</i> is unspecified.
70526		
70527		
70528		
70529	h	Equivalent to %b.
70530	H	The hour (24-hour clock) [00,23]; leading zeros shall be permitted but shall not be required. The <i>tm_hour</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number.
70531		
70532		
70533	I	The hour (12-hour clock) [01,12]; leading zeros shall be permitted but shall not be required. If a <i>p</i> conversion is also performed, the <i>tm_hour</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to the hour, by the 24-hour clock, corresponding to the combined results of the <b>I</b> and <i>p</i> conversions. If a <i>p</i> conversion is not also performed, the behavior is unspecified.
70534		
70535		
70536		
70537		
70538	j	The day number of the year [001,366]; leading zeros shall be permitted but shall not be required. The <i>tm_yday</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number minus 1.
70539		
70540		

70541	m	The month number [01,12]; leading zeros shall be permitted but shall not be required.
70542		The <i>tm_mon</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number
70543		minus 1.
70544	M	The minute [00,59]; leading zeros shall be permitted but shall not be required. The
70545		<i>tm_min</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number.
70546	n	Any white-space bytes.
70547	p	The locale's equivalent of a.m. or p.m. If an <b>I</b> conversion is also performed, the <i>tm_hour</i>
70548		member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set as specified for the <b>I</b>
70549		conversion; otherwise, the behavior is unspecified.
70550	r	12-hour clock time, if the 12-hour format is supported in the locale (see XBD <a href="#">Section</a>
70551		<a href="#">7.3.5</a> , on page 152); in the POSIX locale, this shall be equivalent to <code>%I:%M:%S %p</code> . The
70552		members of the <b>tm</b> structure pointed to by <i>tm</i> shall be set as specified for the
70553		conversions present in the locale's <i>t_fmt_ampm</i> value.
70554	R	Equivalent to <code>%H:%M</code> .
70555	s	The number of seconds since the Epoch as a decimal number (see XBD <a href="#">Section 4.19</a> , on
70556		page 107); leading zeros shall be permitted but shall not be required. The effect of this
70557		number, if any, on the <b>tm</b> structure pointed to by <i>tm</i> is unspecified.
70558	S	The seconds [00,60]; leading zeros shall be permitted but shall not be required. The
70559		<i>tm_sec</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number.
70560	t	Any white-space bytes.
70561	T	Equivalent to <code>%H:%M:%S</code> .
70562	u	The weekday as a decimal number [1,7], with 1 representing Monday. The <i>tm_wday</i>
70563		member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number modulo 7.
70564	U	The week number of the year (Sunday as the first day of the week) as a decimal
70565		number [00,53]; leading zeros shall be permitted but shall not be required. The effect of
70566		this week number, if any, on the <b>tm</b> structure pointed to by <i>tm</i> is unspecified.
70567	V	The week number of the week-based year (see below) as a decimal number [01,53].
70568		Leading zeros shall be permitted but shall not be required. The effect of this week
70569		number, if any, on the <b>tm</b> structure pointed to by <i>tm</i> is unspecified.
70570	w	The weekday as a decimal number [0,6], with 0 representing Sunday. The <i>tm_wday</i>
70571		member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this number.
70572	W	The week number of the year (Monday as the first day of the week) as a decimal
70573		number [00,53]; leading zeros shall be permitted but shall not be required. The effect of
70574		this week number, if any, on the <b>tm</b> structure pointed to by <i>tm</i> is unspecified.
70575	x	The date, using the locale's date format. The members of the <b>tm</b> structure pointed to by
70576		<i>tm</i> shall be set as specified for the conversions present in the locale's <i>d_fmt</i> value.
70577	X	The time, using the locale's time format. The members of the <b>tm</b> structure pointed to by
70578		<i>tm</i> shall be set as specified for the conversions present in the locale's <i>t_fmt</i> value.
70579	y	The last two digits of the year; leading zeros shall be permitted but shall not be
70580		required. A leading '+' or '-' character shall be permitted before any leading zeros
70581		but shall not be required. If a <b>C</b> conversion is not also performed, values in the range
70582		[69,99] shall refer to years 1969 to 1999 inclusive and values in the range [00,68] shall
70583		refer to years 2000 to 2068 inclusive. If a <b>C</b> conversion is also performed, the <i>tm_year</i>

70584		member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set as specified for the C
70585		conversion; otherwise, the <i>tm_year</i> member shall be set to the calculated year minus
70586		1900.
70587	<b>Note:</b>	It is expected that in a future version of this standard the default century inferred
70588		from a 2-digit year will change. (This would apply to all commands accepting a
70589		2-digit year as input.)
70590	Y	The full year {4}; leading zeros shall be permitted but shall not be required. A leading
70591		'+' or '-' character shall be permitted before any leading zeros but shall not be
70592		required. The <i>tm_year</i> member of the <b>tm</b> structure pointed to by <i>tm</i> shall be set to this
70593		number minus 1900.
70594	z	The offset from UTC in the ISO 8601-1:2019 standard format (+hhmm or -hhmm). For
70595		example, "-0430" means 4 hours 30 minutes behind UTC (west of Greenwich). The
70596		effect of this offset, if any, on the <b>tm</b> structure pointed to by <i>tm</i> is unspecified.
70597	Z	The timezone name. If this name matches the name pointed to by <i>tzname</i> [1], and the
70598		names pointed to by <i>tzname</i> [0] and <i>tzname</i> [1] differ, then the <i>tm_isdst</i> member of the <b>tm</b>
70599		structure pointed to by <i>tm</i> shall be set to 1. Otherwise, if this name matches the name
70600		pointed to by <i>tzname</i> [0] then the <i>tm_isdst</i> member of the <b>tm</b> structure pointed to by <i>tm</i>
70601		shall be set to 0. The <i>tm_zone</i> and <i>tm_gmtoff</i> members of the structure may also be set in
70602		an unspecified manner. Members other than <i>tm_isdst</i> , <i>tm_zone</i> , and <i>tm_gmtoff</i> may be
70603		affected if an <i>s</i> conversion is also performed but shall otherwise not be affected.
70604	%	Replaced by %.

### Modified Conversion Specifiers

70605		Some conversion specifiers can be modified by the E and O modifier characters to indicate that
70606		an alternative format or specification should be used rather than the one normally used by the
70607		unmodified conversion specifier. If the alternative format or specification does not exist in the
70608		current locale, the behavior shall be as if the unmodified conversion specification were used.
70609		
70610	%Ec	The locale's alternative appropriate date and time representation.
70611	%EC	The name of the base year (period) in the locale's alternative representation.
70612	%Ex	The locale's alternative date representation.
70613	%EX	The locale's alternative time representation.
70614	%Ey	The offset from %EC (year only) in the locale's alternative representation.
70615	%EY	The full alternative year representation.
70616	%Ob	Equivalent to %b.
70617	%OB	Equivalent to %B.
70618	%Od	The day of the month using the locale's alternative numeric symbols; leading zeros
70619		shall be permitted but shall not be required.
70620	%Oe	Equivalent to %Od.
70621	%Oh	Equivalent to %b.
70622	%OH	The hour (24-hour clock) using the locale's alternative numeric symbols.
70623	%OI	The hour (12-hour clock) using the locale's alternative numeric symbols.



70624	%Om	The month using the locale's alternative numeric symbols.
70625	%OM	The minutes using the locale's alternative numeric symbols.
70626	%OS	The seconds using the locale's alternative numeric symbols.
70627	%OU	The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
70628		
70629	%OV	The same as %V but using the locale's alternative numeric symbols.
70630	%Ow	The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
70631		
70632	%OW	The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
70633		
70634	%Oy	The year (offset from %C) using the locale's alternative numeric symbols.
70635	%g, %G, and %V convert values according to the ISO 8601-1:2019 standard week-based year. In this system, weeks begin on a Monday and week 1 of the week-based year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding week-based year (thus, the string "1998 53 6" with format specifier "%G %V %u" represents Saturday 2nd January 1999). If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following week-based year (thus, the string "1998 01 2" with format specifier "%G %V %u" represents Tuesday 30th December 1997).	
70636		
70637		
70638		
70639		
70640		
70641		
70642		
70643		
70644	A conversion specification composed of white-space bytes is executed by scanning input up to the first non-white-space byte (which remains unscanned), or until no more characters can be scanned.	
70645		
70646		
70647	A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unscanned.	
70648		
70649		
70650		
70651	A series of conversion specifications composed of %n, %t, white-space bytes, or any combination is executed by scanning up to the first non-white-space byte (which remains unscanned), or until no more characters can be scanned.	
70652		
70653		
70654	Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the affected <b>tm</b> structure members are set as specified in the description of the conversion specification. Case is ignored when matching items in <i>buf</i> such as month or weekday names. If no match is found, <i>strptime()</i> fails and no more characters are scanned.	
70655		
70656		
70657		
70658		
70659		
70660		
70661	<b>RETURN VALUE</b>	
70662	Upon successful completion, <i>strptime()</i> shall return a pointer to the character following the last character parsed. Otherwise, a null pointer shall be returned.	
70663		
70664	<b>ERRORS</b>	
70665	No errors are defined.	

## 70666 EXAMPLES

70667 **Convert a Date-Plus-Time String to Broken-Down Time and Then into Seconds**

70668 The following example demonstrates the use of *strptime()* to convert a string into broken-down  
70669 time. The broken-down time is then converted into seconds since the Epoch using *mktime()*.

```
70670 #include <time.h>
70671 ...
70672 struct tm tm;
70673 time_t t;
70674 if (strptime("6 Dec 2001 12:33:45", "%d %b %Y %H:%M:%S", &tm) == NULL)
70675     /* Handle error */;
70676 printf("year: %d; month: %d; day: %d;\n",
70677        tm.tm_year, tm.tm_mon, tm.tm_mday);
70678 printf("hour: %d; minute: %d; second: %d\n",
70679        tm.tm_hour, tm.tm_min, tm.tm_sec);
70680 printf("week day: %d; year day: %d\n", tm.tm_wday, tm.tm_yday);
70681 tm.tm_isdst = -1; /* Not set by strptime(); tells mktime()
70682                  to determine whether daylight saving time
70683                  is in effect */
70684 t = mktime(&tm);
70685 if (t == -1)
70686     /* Handle error */;
70687 printf("seconds since the Epoch: %ld\n", (long) t);"
```

## 70688 APPLICATION USAGE

70689 Several “equivalent to” formats and the special processing of white-space characters are  
70690 provided in order to ease the use of identical format strings for *strptime()* and *strptime()*.

70691 It should be noted that dates constructed by the *strptime()* function with the %Y or %C%Y  
70692 conversion specifiers may have values larger than 9999. If the *strptime()* function is used to read  
70693 such values using %C%Y or %Y, the year values will be truncated to four digits. Applications  
70694 should use %+w%Y or %+xY with *w* and *x* set large enough to contain the full value of any years  
70695 that will be printed or scanned.

70696 The effect of the *s* conversion is unspecified because existing implementations differ in behavior.  
70697 Some do a conversion equivalent to *gmtime()*, ignoring all available timezone information; some  
70698 do a conversion equivalent to *localtime()*, using the same timezone it would use and ignoring  
70699 any timezone information provided by a *z* or *Z* conversion. Although none has been observed,  
70700 there may be existing (or future) implementations that use timezone information provided by a  
70701 *z* or *Z* conversion, although using the latter would not be reliable as timezone names are often  
70702 ambiguous. Applications that need to convert a seconds since the Epoch value to a **tm** structure  
70703 should call *gmtime()* or *localtime()* (or their thread-safe equivalents) directly.

70704 The effect of the *z* conversion is unspecified because existing implementations differ in behavior.  
70705 Some just use it to set the *tm\_gmtoff* member of the **tm** structure; some use the value to adjust the  
70706 other field members to represent UTC, convert the resulting time to a seconds since the Epoch  
70707 value, and then convert back to a **tm** structure by the equivalent of *localtime()*. An application  
70708 that needs either of these behaviors should perform the necessary processing explicitly itself.

70709 Although the *Z* conversion might be expected to set the *tm\_zone* member of the **tm** structure, no  
70710 existing implementation has been found that sets it. Applications that need it set should set it

- 70711 explicitly after calling *strptime()*.
- 70712 See also the APPLICATION USAGE section in *strptime()*.
- 70713 It is unspecified whether multiple calls to *strptime()* using the same **tm** structure will update the  
 70714 current contents of the structure or overwrite all contents of the structure. Conforming  
 70715 applications should make a single call to *strptime()* with a format and all data needed to  
 70716 completely specify the date and time being converted.
- 70717 **RATIONALE**
- 70718 See the RATIONALE section for *strptime()*.
- 70719 **FUTURE DIRECTIONS**
- 70720 None.
- 70721 **SEE ALSO**
- 70722 *fprintf()*, *fscanf()*, *strptime()*, *time()*
- 70723 XBD <**time.h**>
- 70724 **CHANGE HISTORY**
- 70725 First released in Issue 4.
- 70726 **Issue 5**
- 70727 Moved from ENHANCED I18N to BASE.
- 70728 The [ENOSYS] error is removed.
- 70729 The exact meaning of the %y and %Oy specifiers is clarified in the DESCRIPTION.
- 70730 **Issue 6**
- 70731 The Open Group Corrigendum U033/5 is applied. The %r specifier description is reworded.
- 70732 The normative text is updated to avoid use of the term “must” for application requirements.
- 70733 The **restrict** keyword is added to the *strptime()* prototype for alignment with the  
 70734 ISO/IEC 9899:1999 standard.
- 70735 The Open Group Corrigendum U047/2 is applied.
- 70736 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
 70737 specification” for consistency with *strptime()*.
- 70738 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/133 is applied, adding the example to the  
 70739 EXAMPLES section.
- 70740 **Issue 7**
- 70741 Austin Group Interpretation 1003.1-2001 #041 is applied, updating the DESCRIPTION and  
 70742 APPLICATION USAGE sections.
- 70743 Austin Group Interpretation 1003.1-2001 #163 is applied.
- 70744 SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression  
 70745 that %Y is 4-digit years.
- 70746 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0345 [920] and XSH/TC2-2008/0346  
 70747 [919] are applied.
- 70748 **Issue 8**
- 70749 Austin Group Defect 169 is applied, adding the s conversion.
- 70750 Austin Group Defects 258 and 1166 are applied, adding the Ob, OB, and Oh modified  
 70751 conversions.

- 70752 Austin Group Defect 879 is applied, adding the F, g, G, u, V, z, and Z conversions and the OV  
70753 modified conversion, and adding a statement to the week number conversions that their effect  
70754 on the **tm** structure pointed to by *tm* is unspecified.
- 70755 Austin Group Defect 1163 is applied, clarifying the handling of white space in the format string.
- 70756 Austin Group Defect 1307 is applied, changing the r conversion in relation to locales that do not  
70757 support the 12-hour clock format.
- 70758 Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to  
70759 ensure that the format is a character string, beginning and ending in its initial shift state, if any.
- 70760 Austin Group Defect 1727 is applied, clarifying which members of the **tm** structure are updated  
70761 by each conversion.

70762 **NAME**

70763 strrchr — string scanning operation

70764 **SYNOPSIS**

70765 #include &lt;string.h&gt;

70766 char \*strrchr(const char \*s, int c);

70767 **DESCRIPTION**

70768 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
70769 conflict between the requirements described here and the ISO C standard is unintentional. This  
70770 volume of POSIX.1-2024 defers to the ISO C standard.

70771 The *strrchr()* function shall locate the last occurrence of *c* (converted to a **char**) in the string  
70772 pointed to by *s*. The terminating NUL character is considered to be part of the string.

70773 CX The *strrchr()* function shall not change the setting of *errno* on valid input.

70774 **RETURN VALUE**

70775 Upon successful completion, *strrchr()* shall return a pointer to the byte or a null pointer if *c* does  
70776 not occur in the string.

70777 **ERRORS**

70778 No errors are defined.

70779 **EXAMPLES**70780 **Finding the Base Name of a File**

70781 The following example uses *strrchr()* to get a pointer to the base name of a file. The *strrchr()*  
70782 function searches backwards through the name of the file to find the last '/' character in *name*.  
70783 This pointer (plus one) will point to the base name of the file.

```
70784 #include <string.h>
70785 ...
70786 const char *name;
70787 char *basename;
70788 ...
70789 basename = strrchr(name, '/') + 1;
70790 ...
```

70791 **APPLICATION USAGE**

70792 None.

70793 **RATIONALE**

70794 None.

70795 **FUTURE DIRECTIONS**

70796 None.

70797 **SEE ALSO**70798 [strchr\(\)](#)70799 XBD [<string.h>](#)70800 **CHANGE HISTORY**

70801 First released in Issue 1. Derived from Issue 1 of the SVID.

70802 **Issue 8**

70803

70804

Austin Group Defect 448 is applied, adding a requirement that *strchr()* does not change the setting of *errno* on valid input.

70805 **NAME**

70806 strsignal — get signal message string

70807 **SYNOPSIS**

```
70808 CX #include <string.h>  
70809 char *strsignal(int signum);
```

70810 **DESCRIPTION**

70811 The *strsignal()* function shall map the signal number in *signum* to an implementation-defined  
70812 string and shall return a pointer to it. It shall use the same set of messages as the *psignal()*  
70813 function.

70814 The application shall not modify the string returned. The returned pointer might be invalidated  
70815 or the string content might be overwritten by a subsequent call to *strsignal()* or *setlocale()*. The  
70816 returned pointer might also be invalidated if the calling thread is terminated.

70817 The contents of the message strings returned by *strsignal()* should be determined by the setting  
70818 of the *LC\_MESSAGES* category in the current locale.

70819 The implementation shall behave as if no function defined in this standard calls *strsignal()*.

70820 Since no return value is reserved to indicate an error, an application wishing to check for error  
70821 situations should set *errno* to 0, then call *strsignal()*, then check *errno*.

70822 The *strsignal()* function need not be thread-safe.

70823 **RETURN VALUE**

70824 Upon successful completion, *strsignal()* shall return a pointer to a string. Otherwise, if *signum* is  
70825 not a valid signal number, the return value is unspecified.

70826 **ERRORS**

70827 No errors are defined.

70828 **EXAMPLES**

70829 None.

70830 **APPLICATION USAGE**

70831 None.

70832 **RATIONALE**

70833 If *signum* is not a valid signal number, some implementations return NULL, while for others the  
70834 *strsignal()* function returns a pointer to a string containing an unspecified message denoting an  
70835 unknown signal. POSIX.1-2024 leaves this return value unspecified.

70836 **FUTURE DIRECTIONS**

70837 None.

70838 **SEE ALSO**

70839 [psiginfo\(\)](#), [setlocale\(\)](#), [sig2str\(\)](#)

70840 XBD [<string.h>](#)

70841 **CHANGE HISTORY**

70842 First released in Issue 7.

70843 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0609 [75] is applied.

70844 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0347 [656] is applied.

70845 **Issue 8**

70846 Austin Group Defect 1138 is applied, adding *sig2str()* to the SEE ALSO section.

70847 Austin Group Defect 1474 is applied, changing the NAME section.



70848 **NAME**

70849 strspn — get length of a substring

70850 **SYNOPSIS**

70851 #include &lt;string.h&gt;

70852 size\_t strspn(const char \*s1, const char \*s2);

70853 **DESCRIPTION**

70854 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
70855 conflict between the requirements described here and the ISO C standard is unintentional. This  
70856 volume of POSIX.1-2024 defers to the ISO C standard.

70857 The *strspn()* function shall compute the length (in bytes) of the maximum initial segment of the  
70858 string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

70859 CX The *strspn()* function shall not change the setting of *errno* on valid input.

70860 **RETURN VALUE**

70861 The *strspn()* function shall return the computed length; no return value is reserved to indicate an  
70862 error.

70863 **ERRORS**

70864 No errors are defined.

70865 **EXAMPLES**

70866 None.

70867 **APPLICATION USAGE**

70868 None.

70869 **RATIONALE**

70870 None.

70871 **FUTURE DIRECTIONS**

70872 None.

70873 **SEE ALSO**70874 [strcspn\(\)](#)70875 XBD [<string.h>](#)70876 **CHANGE HISTORY**

70877 First released in Issue 1. Derived from Issue 1 of the SVID.

70878 **Issue 5**

70879 The RETURN VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not  
70880 *s* itself as was previously stated.

70881 **Issue 7**

70882 SD5-XSH-ERN-182 is applied.

70883 **Issue 8**

70884 Austin Group Defect 448 is applied, adding a requirement that *strspn()* does not change the  
70885 setting of *errno* on valid input.

70886 **NAME**

70887        strstr — find a substring

70888 **SYNOPSIS**

70889        #include &lt;string.h&gt;

70890        char \*strstr(const char \*s1, const char \*s2);

70891 **DESCRIPTION**

70892 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
70893        conflict between the requirements described here and the ISO C standard is unintentional. This  
70894        volume of POSIX.1-2024 defers to the ISO C standard.

70895        The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the  
70896        sequence of bytes (excluding the terminating NUL character) in the string pointed to by *s2*.

70897 CX        The *strstr()* function shall not change the setting of *errno* on valid input.

70898 **RETURN VALUE**

70899        Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer  
70900        if the string is not found.

70901        If *s2* points to a string with zero length, the function shall return *s1*.

70902 **ERRORS**

70903        No errors are defined.

70904 **EXAMPLES**

70905        None.

70906 **APPLICATION USAGE**

70907        None.

70908 **RATIONALE**

70909        None.

70910 **FUTURE DIRECTIONS**

70911        None.

70912 **SEE ALSO**

70913        [memmem\(\)](#), [strchr\(\)](#)

70914        XBD [<string.h>](#)

70915 **CHANGE HISTORY**

70916        First released in Issue 3. Included for alignment with the ANSI C standard.

70917 **Issue 8**

70918        Austin Group Defect 448 is applied, adding a requirement that *strstr()* does not change the  
70919        setting of *errno* on valid input.

70920        Austin Group Defect 1061 is applied, adding *memmem()* to the SEE ALSO section.

70921 **NAME**70922 `strtod, strtodf, strtold` — convert a string to a double-precision number70923 **SYNOPSIS**70924 `#include <stdlib.h>`70925 `double strtod(const char *restrict nptr, char **restrict endptr);`70926 `float strtodf(const char *restrict nptr, char **restrict endptr);`70927 `long double strtold(const char *restrict nptr, char **restrict endptr);`70928 **DESCRIPTION**70929 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
70930 conflict between the requirements described here and the ISO C standard is unintentional. This  
70931 volume of POSIX.1-2024 defers to the ISO C standard.70932 These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**,  
70933 and **long double** representation, respectively. First, they decompose the input string into three  
70934 parts:

- 70935
1. An initial, possibly empty, sequence of white-space bytes
  - 70936 2. A subject sequence interpreted as a floating-point constant or representing infinity or  
70937 NaN
  - 70938 3. A final string of one or more unrecognized characters, including the terminating NUL  
70939 character of the input string

70940 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
70941 the result.70942 The expected form of the subject sequence is an optional '+' or '-' sign, then one of the  
70943 following:

- 70944
- A non-empty sequence of decimal digits optionally containing a radix character; then an  
70945 optional exponent part consisting of the character 'e' or the character 'E', optionally  
70946 followed by a '+' or '-' character, and then followed by one or more decimal digits
  - 70947 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
70948 character; then an optional binary exponent part consisting of the character 'p' or the  
70949 character 'P', optionally followed by a '+' or '-' character, and then followed by one or  
70950 more decimal digits
  - 70951 • One of INF or INFINITY, ignoring case
  - 70952 • One of NAN or NAN(*n-char-sequence*<sub>opt</sub>), ignoring case in the NAN part, where:

70953 `n-char-sequence:`70954 `digit`70955 `nondigit`70956 `n-char-sequence digit`70957 `n-char-sequence nondigit`70958 The subject sequence is defined as the longest initial subsequence of the input string, starting  
70959 with the first non-white-space byte, that is of the expected form. The subject sequence contains  
70960 no characters if the input string is not of the expected form.70961 If the subject sequence has the expected form for a floating-point number, the sequence of  
70962 characters starting with the first digit or the decimal-point character (whichever occurs first)  
70963 shall be interpreted as a floating constant of the C language, except that the radix character shall  
70964 be used in place of a period, and that if neither an exponent part nor a radix character appears in

70965 a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal  
 70966 floating-point number, an exponent part of the appropriate type with value zero is assumed to  
 70967 follow the last digit in the string. If the subject sequence begins with a <hyphen-minus>, the  
 70968 sequence shall be interpreted as negated. A character sequence INF or INFINITY shall be  
 70969 interpreted as an infinity, if representable in the return type, else as if it were a floating constant  
 70970 that is too large for the range of the return type. A character sequence NAN or NAN(*n-char-*  
 70971 *sequence<sub>opt</sub>*) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a  
 70972 subject sequence part that does not have the expected form; the meaning of the *n-char* sequences  
 70973 is implementation-defined. A pointer to the final string is stored in the object pointed to by  
 70974 *endptr*, provided that *endptr* is not a null pointer.

70975 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the value  
 70976 resulting from the conversion is correctly rounded.

70977 CX The radix character is defined in the current locale (category *LC\_NUMERIC*). In the POSIX  
 70978 locale, or in a locale where the radix character is not defined, the radix character shall default to  
 70979 a <period> ('.').

70980 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be  
 70981 accepted.

70982 If the subject sequence is empty or does not have the expected form, no conversion shall be  
 70983 performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is  
 70984 not a null pointer.

70985 These functions shall not change the setting of *errno* if successful.

70986 Since 0 is returned on error and is also a valid return on success, an application wishing to check  
 70987 for error situations should set *errno* to 0, then call *strtod()*, *strtof()*, or *strtold()*, then check *errno*.

#### 70988 RETURN VALUE

70989 Upon successful completion, these functions shall return the converted value. If no conversion  
 70990 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

70991 If the correct value would cause an overflow and default rounding is in effect, ±HUGE\_VAL,  
 70992 ±HUGE\_VALF, or ±HUGE\_VALL shall be returned (according to the sign of the value), and *errno*  
 70993 shall be set to [ERANGE].

70994 If the correct value would cause an underflow, a value whose magnitude is no greater than the  
 70995 CX smallest normalized positive number in the return type shall be returned and *errno* set to  
 70996 [ERANGE].

#### 70997 ERRORS

70998 These functions shall fail if:

70999 [ERANGE] The value to be returned would cause overflow and default rounding is in  
 71000 CX effect or the value to be returned would cause underflow.

71001 These functions may fail if:

71002 CX [EINVAL] No conversion could be performed.

71003 **EXAMPLES**

71004 None.

71005 **APPLICATION USAGE**

71006 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the  
 71007 result is not exactly representable, the result should be one of the two numbers in the  
 71008 appropriate internal format that are adjacent to the hexadecimal floating source value, with the  
 71009 extra stipulation that the error should have a correct sign for the current rounding direction.

71010 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>)  
 71011 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 71012 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 71013 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 71014 values of *L*, *D*, and *U* satisfy  $L \leq D \leq U$ . The result should be one of the (equal or adjacent)  
 71015 values that would be obtained by correctly rounding *L* and *U* according to the current rounding  
 71016 direction, with the extra stipulation that the error with respect to *D* should have a correct sign  
 71017 for the current rounding direction.

71018 The changes to *strtod()* introduced by the ISO/IEC 9899:1999 standard can alter the behavior of  
 71019 well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier  
 71020 versions of this standard. One such example would be:

```

71021 int
71022 what_kind_of_number (char *s)
71023 {
71024     char *endp;
71025     double d;
71026     long l;

71027     d = strtod(s, &endp);
71028     if (s != endp && *endp == '\0')
71029         printf("It's a float with value %g\n", d);
71030     else
71031     {
71032         l = strtol(s, &endp, 0);
71033         if (s != endp && *endp == '\0')
71034             printf("It's an integer with value %ld\n", l);
71035         else
71036             return 1;
71037     }
71038     return 0;
71039 }
```

71040 If the function is called with:

```
71041 what_kind_of_number ("0x10")
```

71042 an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
71043 It's an integer with value 16
```

71044 With the ISO/IEC 9899:1999 standard, the result is:

```
71045 It's a float with value 16
```

71046 The change in behavior is due to the inclusion of floating-point numbers in hexadecimal  
 71047 notation without requiring that either a decimal point or the binary exponent be present.

71048	<b>RATIONALE</b>
71049	None.
71050	<b>FUTURE DIRECTIONS</b>
71051	None.
71052	<b>SEE ALSO</b>
71053	<i>fscanf()</i> , <i>isspace()</i> , <i>localeconv()</i> , <i>setlocale()</i> , <i>strtol()</i>
71054	XBD Chapter 7 (on page 127), <code>&lt;float.h&gt;</code> , <code>&lt;stdlib.h&gt;</code>
71055	<b>CHANGE HISTORY</b>
71056	First released in Issue 1. Derived from Issue 1 of the SVID.
71057	<b>Issue 5</b>
71058	The DESCRIPTION is updated to indicate that <i>errno</i> is not changed if the function is successful.
71059	<b>Issue 6</b>
71060	Extensions beyond the ISO C standard are marked.
71061	The following new requirements on POSIX implementations derive from alignment with the
71062	Single UNIX Specification:
71063	• In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
71064	added if no conversion could be performed.
71065	The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
71066	• The <i>strtod()</i> function is updated.
71067	• The <i>strtof()</i> and <i>strtold()</i> functions are added.
71068	• The DESCRIPTION is extensively revised.
71069	ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.
71070	IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second
71071	paragraph in the RETURN VALUE section. This change clarifies the sign of the return value.
71072	<b>Issue 7</b>
71073	Austin Group Interpretation 1003.1-2001 #015 is applied.
71074	POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0610 [302], XSH/TC1-2008/0611 [94],
71075	and XSH/TC1-2008/0612 [105] are applied.
71076	POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0348 [584] and XSH/TC2-2008/0349
71077	[796] are applied.
71078	<b>Issue 8</b>
71079	Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.
71080	Austin Group Defect 1213 is applied, correcting some typographic errors in the APPLICATION
71081	USAGE section.
71082	Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018
71083	standard.
71084	Austin Group Defect 1686 is applied, adding CX shading to some text in the RETURN VALUE
71085	section.

71086 **NAME**

71087 strtoimax, strtoumax — convert string to integer type

71088 **SYNOPSIS**

71089 #include &lt;inttypes.h&gt;

71090 intmax\_t strtoimax(const char \*restrict *nptr*, char \*\*restrict *endptr*,  
71091 int *base*);71092 uintmax\_t strtoumax(const char \*restrict *nptr*, char \*\*restrict *endptr*,  
71093 int *base*);71094 **DESCRIPTION**71095 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
71096 conflict between the requirements described here and the ISO C standard is unintentional. This  
71097 volume of POSIX.1-2024 defers to the ISO C standard.71098 These functions shall be equivalent to the *strtol()*, *strtoll()*, *strtoul()*, and *strtoull()* functions,  
71099 except that the initial portion of the string shall be converted to **intmax\_t** and **uintmax\_t**  
71100 representation, respectively.71101 **RETURN VALUE**

71102 These functions shall return the converted value, if any.

71103 CX If no conversion could be performed, zero shall be returned and *errno* may be set to [EINVAL].71104 CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].71105 If the correct value is outside the range of representable values, {INTMAX\_MAX},  
71106 {INTMAX\_MIN}, or {UINTMAX\_MAX} shall be returned (according to the return type and sign  
71107 of the value, if any), and *errno* shall be set to [ERANGE].71108 **ERRORS**

71109 These functions shall fail if:

71110 CX [EINVAL] The value of *base* is not supported.

71111 [ERANGE] The value to be returned is not representable.

71112 These functions may fail if:

71113 [EINVAL] No conversion could be performed.

71114 **EXAMPLES**

71115 None.

71116 **APPLICATION USAGE**71117 Since the value of *\*endptr* is unspecified if the value of *base* is not supported, applications should  
71118 either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for  
71119 an [EINVAL] error before examining *\*endptr*.71120 **RATIONALE**

71121 None.

71122 **FUTURE DIRECTIONS**

71123 None.

71124 **SEE ALSO**71125 [strtol\(\)](#), [strtoul\(\)](#)71126 XBD [<inttypes.h>](#)

71127 **CHANGE HISTORY**

71128 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

71129 **Issue 7**

71130 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0613 [453] and XSH/TC1-2008/0614  
71131 [453] are applied.



71132 **NAME**

71133            strtok, strtok\_r — split string into tokens

71134 **SYNOPSIS**

71135            #include &lt;string.h&gt;

71136            char \*strtok(char \*restrict *s*, const char \*restrict *sep*);71137 CX         char \*strtok\_r(char \*restrict *s*, const char \*restrict *sep*,71138            char \*\*restrict *state*);71139 **DESCRIPTION**71140 CX         For *strtok()*: The functionality described on this reference page is aligned with the ISO C  
71141 standard. Any conflict between the requirements described here and the ISO C standard is  
71142 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.71143            A sequence of calls to *strtok()* breaks the string pointed to by *s* into a sequence of tokens, each of  
71144 which is delimited by a byte from the string pointed to by *sep*. The first call in the sequence has *s*  
71145 as its first argument, and is followed by calls with a null pointer as their first argument. The  
71146 separator string pointed to by *sep* may be different from call to call.71147            The first call in the sequence searches the string pointed to by *s* for the first byte that is *not*  
71148 contained in the current separator string pointed to by *sep*. If no such byte is found, then there  
71149 are no tokens in the string pointed to by *s* and *strtok()* shall return a null pointer. If such a byte is  
71150 found, it is the start of the first token.71151            The *strtok()* function then searches from there for a byte that *is* contained in the current separator  
71152 string. If no such byte is found, the current token extends to the end of the string pointed to by *s*,  
71153 and subsequent searches for a token shall return a null pointer. If such a byte is found, it is  
71154 overwritten by a NUL character, which terminates the current token. The *strtok()* function saves  
71155 a pointer to the following byte, from which the next search for a token shall start.71156            Each subsequent call, with a null pointer as the value of the first argument, starts searching from  
71157 the saved pointer and behaves as described above.71158            The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
71159 *strtok()*.71160            The *strtok()* function need not be thread-safe; however, *strtok()* shall avoid data races with all  
71161 other functions.71162 CX         The *strtok\_r()* function shall be equivalent to *strtok()*, except that *strtok\_r()* shall be thread-safe  
71163 and the argument *state* points to a user-provided pointer that allows *strtok\_r()* to maintain state  
71164 between calls which scan the same string. The application shall ensure that the pointer pointed  
71165 to by *state* is unique for each string (*s*) being processed concurrently by *strtok\_r()* calls. The  
71166 application need not initialize the pointer pointed to by *state* to any particular value. The  
71167 implementation shall not update the pointer pointed to by *state* to point (directly or indirectly) to  
71168 resources, other than within the string *s*, that need to be freed or released by the caller.71169 CX         The *strtok()* and *strtok\_r()* functions shall not change the setting of *errno* on valid input.71170 **RETURN VALUE**71171            Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,  
71172 if there is no token, *strtok()* shall return a null pointer.71173 CX         The *strtok\_r()* function shall return a pointer to the token found, or a null pointer when no token  
71174 is found.

71175 **ERRORS**

71176 No errors are defined.

71177 **EXAMPLES**71178 **Searching for Word Separators**

71179 The following example searches for tokens separated by &lt;space&gt; characters.

```

71180 #include <string.h>
71181 ...
71182 char *token;
71183 char line[] = "LINE TO BE SEPARATED";
71184 char *search = " ";

71185 /* Token will point to "LINE". */
71186 token = strtok(line, search);

71187 /* Token will point to "TO". */
71188 token = strtok(NULL, search);

```

71189 **Find First two Fields in a Buffer**

71190 The following example uses *strtok()* to find two character strings (a key and data associated with  
 71191 that key) separated by any combination of <space>, <tab>, or <newline> characters at the start  
 71192 of the array of characters pointed to by *buffer*.

```

71193 #include <string.h>
71194 ...
71195 char *buffer;
71196 ...
71197 struct element {
71198     char *key;
71199     char *data;
71200 } e;
71201 ...
71202 // Load the buffer...
71203 ...
71204 // Get the key and its data...
71205 e.key = strtok(buffer, " \t\n");
71206 e.data = strtok(NULL, " \t\n");
71207 // Process the rest of the contents of the buffer...
71208 ...

```

71209 **APPLICATION USAGE**

71210 Note that if *sep* is the empty string, *strtok()* and *strtok\_r()* return a pointer to the remainder of the  
 71211 string being tokenized.

71212 The *strtok\_r()* function is thread-safe and stores its state in a user-supplied buffer instead of  
 71213 possibly using a static data area that may be overwritten by an unrelated call from another  
 71214 thread.

71215 **RATIONALE**

71216 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to  
 71217 the last substring between separator strings. This function uses static storage to keep track of  
 71218 the current string position between calls. The new function, *strtok\_r()*, takes an additional

- 71219 argument, *state*, to keep track of the current position in the string.
- 71220 **FUTURE DIRECTIONS**
- 71221 None.
- 71222 **SEE ALSO**
- 71223 XBD <[string.h](#)>
- 71224 **CHANGE HISTORY**
- 71225 First released in Issue 1. Derived from Issue 1 of the SVID.
- 71226 **Issue 5**
- 71227 The *strtok\_r()* function is included for alignment with the POSIX Threads Extension.
- 71228 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.
- 71229 **Issue 6**
- 71230 Extensions beyond the ISO C standard are marked.
- 71231 The *strtok\_r()* function is marked as part of the Thread-Safe Functions option.
- 71232 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 71233 The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.
- 71234
- 71235 The **restrict** keyword is added to the *strtok()* and *strtok\_r()* prototypes for alignment with the
- 71236 ISO/IEC 9899:1999 standard.
- 71237 **Issue 7**
- 71238 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 71239 SD5-XSH-ERN-235 is applied, correcting an example.
- 71240 The *strtok\_r()* function is moved from the Thread-Safe Functions option to the Base.
- 71241 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0615 [177] is applied.
- 71242 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0350 [878] is applied.
- 71243 **Issue 8**
- 71244 Austin Group Defect 448 is applied, adding a requirement that *strtok()* and *strtok\_r()* do not
- 71245 change the setting of *errno* on valid input.
- 71246 Austin Group Defect 1302 is applied, aligning the *strtok()* function with the ISO/IEC 9899:2018
- 71247 standard.

71248 **NAME**

71249            strtol, strtoll — convert a string to a long integer

71250 **SYNOPSIS**

71251            #include &lt;stdlib.h&gt;

71252            long strtol(const char \*restrict *nptr*, char \*\*restrict *endptr*, int *base*);71253            long long strtoll(const char \*restrict *nptr*, char \*\*restrict *endptr*,71254                    int *base*)71255 **DESCRIPTION**

71256 CX        The functionality described on this reference page is aligned with the ISO C standard. Any  
 71257 conflict between the requirements described here and the ISO C standard is unintentional. This  
 71258 volume of POSIX.1-2024 defers to the ISO C standard.

71259        These functions shall convert the initial portion of the string pointed to by *nptr* to a type **long**  
 71260 and **long long** representation, respectively. First, they decompose the input string into three  
 71261 parts:

- 71262            1. An initial, possibly empty, sequence of white-space bytes
- 71263            2. A subject sequence interpreted as an integer represented in some radix determined by the  
 71264 value of *base*
- 71265            3. A final string of one or more unrecognized characters, including the terminating NUL  
 71266 character of the input string.

71267        Then they shall attempt to convert the subject sequence to an integer, and return the result.

71268        If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
 71269 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
 71270 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
 71271 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
 71272 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 71273 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

71274        If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 71275 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 71276 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
 71277 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
 71278 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
 71279 digits, following the sign if present.

71280        The subject sequence is defined as the longest initial subsequence of the input string, starting  
 71281 with the first non-white-space byte, that is of the expected form. The subject sequence shall  
 71282 contain no characters if the input string is empty or consists entirely of white-space bytes, or if  
 71283 the first non-white-space byte is other than a sign or a permissible letter or digit.

71284        If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
 71285 characters starting with the first digit shall be interpreted as an integer constant. If the subject  
 71286 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the  
 71287 base for conversion, ascribing to each letter its value as given above. If the subject sequence  
 71288 begins with a <hyphen-minus>, the resulting value shall be the negative of the converted value.  
 71289 A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that  
 71290 *endptr* is not a null pointer.

71291 CX        In other than the C or POSIX locale, additional locale-specific subject sequence forms may be  
 71292 accepted.

71293 If the subject sequence is empty or does not have the expected form, no conversion is performed;  
 71294 the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a  
 71295 null pointer.

71296 These functions shall not change the setting of *errno* if successful.

71297 Since 0, {LONG\_MIN} or {LLONG\_MIN}, and {LONG\_MAX} or {LLONG\_MAX} are returned  
 71298 on error and are also valid returns on success, an application wishing to check for error  
 71299 situations should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

#### 71300 RETURN VALUE

71301 Upon successful completion, these functions shall return the converted value, if any. If no  
 71302 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

71303 CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].

71304 If the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
 71305 {LLONG\_MIN}, or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
 71306 *errno* set to [ERANGE].

#### 71307 ERRORS

71308 These functions shall fail if:

71309 CX [EINVAL] The value of *base* is not supported.

71310 [ERANGE] The value to be returned is not representable.

71311 These functions may fail if:

71312 [EINVAL] No conversion could be performed.

#### 71313 EXAMPLES

71314 None.

#### 71315 APPLICATION USAGE

71316 Since the value of *\*endptr* is unspecified if the value of *base* is not supported, applications should  
 71317 either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for  
 71318 an [EINVAL] error before examining *\*endptr*.

#### 71319 RATIONALE

71320 None.

#### 71321 FUTURE DIRECTIONS

71322 None.

#### 71323 SEE ALSO

71324 *fscanf()*, *isalpha()*, *strtod()*

71325 XBD <stdlib.h>

#### 71326 CHANGE HISTORY

71327 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 71328 Issue 5

71329 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 71330 Issue 6

71331 Extensions beyond the ISO C standard are marked.

71332 The following new requirements on POSIX implementations derive from alignment with the  
 71333 Single UNIX Specification:

- 71334 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
71335 added if no conversion could be performed.

71336 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 71337 • The *strtol()* prototype is updated.  
71338 • The *strtoll()* function is added.

71339 **Issue 7**

71340 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0616 [453], XSH/TC1-2008/0617 [105],  
71341 XSH/TC1-2008/0618 [453], XSH/TC1-2008/0619 [453], and XSH/TC1-2008/0620 [453] are  
71342 applied.

71343 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0351 [892], XSH/TC2-2008/0352 [584],  
71344 XSH/TC2-2008/0353 [796], and XSH/TC2-2008/0354 [892] are applied.

71345 **Issue 8**

71346 Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-  
71347 minus> is converted.

71348 Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

71349 **NAME**

71350 strtold — convert a string to a double-precision number

71351 **SYNOPSIS**

71352 #include &lt;stdlib.h&gt;

71353 long double strtold(const char \*restrict *nptr*, char \*\*restrict *endptr*);71354 **DESCRIPTION**71355 Refer to *strtod()*.

71356 **NAME**

71357            **strtoll** — convert a string to a long integer

71358 **SYNOPSIS**

71359            #include <stdlib.h>

71360            long long strtoll(const char \*restrict *str*, char \*\*restrict *endptr*,  
71361                            int *base*);

71362 **DESCRIPTION**

71363            Refer to *strtol()*.



71364 **NAME**

71365 strtoul, strtoull — convert a string to an unsigned long

71366 **SYNOPSIS**

71367 #include &lt;stdlib.h&gt;

```
71368 unsigned long strtoul(const char *restrict str,
71369 char **restrict endptr, int base);
71370 unsigned long long strtoull(const char *restrict str,
71371 char **restrict endptr, int base);
```

71372 **DESCRIPTION**

71373 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 71374 conflict between the requirements described here and the ISO C standard is unintentional. This  
 71375 volume of POSIX.1-2024 defers to the ISO C standard.

71376 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned  
 71377 long** and **unsigned long long** representation, respectively. First, they decompose the input string  
 71378 into three parts:

- 71379 1. An initial, possibly empty, sequence of white-space bytes
- 71380 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 71381 value of *base*
- 71382 3. A final string of one or more unrecognized characters, including the terminating NUL  
 71383 character of the input string

71384 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the  
 71385 result.

71386 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,  
 71387 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A  
 71388 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An  
 71389 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to  
 71390 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 71391 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

71392 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 71393 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 71394 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the  
 71395 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the  
 71396 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and  
 71397 digits, following the sign if present.

71398 The subject sequence is defined as the longest initial subsequence of the input string, starting  
 71399 with the first non-white-space byte, that is of the expected form. The subject sequence shall  
 71400 contain no characters if the input string is empty or consists entirely of white-space bytes, or if  
 71401 the first non-white-space byte is other than a sign or a permissible letter or digit.

71402 If the subject sequence has the expected form and the value of *base* is 0, the sequence of  
 71403 characters starting with the first digit shall be interpreted as an integer constant. If the subject  
 71404 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the  
 71405 base for conversion, ascribing to each letter its value as given above. If the subject sequence  
 71406 begins with a <hyphen-minus>, the resulting value shall be the negative of the converted value;  
 71407 this action shall be performed in the return type. A pointer to the final string shall be stored in  
 71408 the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

71409 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be  
71410 accepted.

71411 If the subject sequence is empty or does not have the expected form, no conversion shall be  
71412 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*  
71413 is not a null pointer.

71414 These functions shall not change the setting of *errno* if successful.

71415 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and are also valid returns  
71416 on success, an application wishing to check for error situations should set *errno* to 0, then call  
71417 *strtol()* or *strtoll()*, then check *errno*.

#### 71418 RETURN VALUE

71419 Upon successful completion, these functions shall return the converted value, if any. If no  
71420 conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

71421 CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].

71422 If the correct value is outside the range of representable values, {ULONG\_MAX} or  
71423 {ULLONG\_MAX} shall be returned and *errno* set to [ERANGE].

#### 71424 ERRORS

71425 These functions shall fail if:

71426 CX [EINVAL] The value of *base* is not supported.

71427 [ERANGE] The value to be returned is not representable.

71428 These functions may fail if:

71429 CX [EINVAL] No conversion could be performed.

#### 71430 EXAMPLES

71431 None.

#### 71432 APPLICATION USAGE

71433 Since the value of *\*endptr* is unspecified if the value of *base* is not supported, applications should  
71434 either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for  
71435 an [EINVAL] error before examining *\*endptr*.

#### 71436 RATIONALE

71437 None.

#### 71438 FUTURE DIRECTIONS

71439 None.

#### 71440 SEE ALSO

71441 [fscanf\(\)](#), [isalpha\(\)](#), [strtod\(\)](#), [strtol\(\)](#)

71442 XBD <stdlib.h>

#### 71443 CHANGE HISTORY

71444 First released in Issue 4. Derived from the ANSI C standard.

#### 71445 Issue 5

71446 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

71447 **Issue 6**

71448 Extensions beyond the ISO C standard are marked.

71449 The following new requirements on POSIX implementations derive from alignment with the  
71450 Single UNIX Specification:

- 71451 • The [EINVAL] error condition is added for when the value of *base* is not supported.
- 71452 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
71453 added if no conversion could be performed.

71454 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 71455 • The *strtoul()* prototype is updated.
- 71456 • The *strtoull()* function is added.

71457 **Issue 7**

71458 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0621 [105], XSH/TC1-2008/0622 [453],  
71459 and XSH/TC1-2008/0623 [453] are applied.

71460 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0355 [584] and XSH/TC2-2008/0356  
71461 [796] are applied.

71462 **Issue 8**

71463 Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-  
71464 minus> is converted.

71465 Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

71466 **NAME**

71467        strtoumax — convert a string to an integer type

71468 **SYNOPSIS**

71469        #include <inttypes.h>

71470        uintmax\_t strtoumax(const char \*restrict *nptr*, char \*\*restrict *endptr*,  
71471                           int *base*);

71472 **DESCRIPTION**

71473        Refer to *strtoimax()*.

71474 **NAME**

71475 strxfrm, strxfrm\_l — string transformation

71476 **SYNOPSIS**

71477 #include &lt;string.h&gt;

71478 size\_t strxfrm(char \*restrict s1, const char \*restrict s2, size\_t n);

71479 CX size\_t strxfrm\_l(char \*restrict s1, const char \*restrict s2,

71480 size\_t n, locale\_t locale);

71481 **DESCRIPTION**71482 CX For *strxfrm()*: The functionality described on this reference page is aligned with the ISO C  
71483 standard. Any conflict between the requirements described here and the ISO C standard is  
71484 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.71485 CX The *strxfrm()* and *strxfrm\_l()* functions shall transform the string pointed to by *s2* and place the  
71486 resulting string into the array pointed to by *s1*. The transformation is such that if *strcmp()* is  
71487 applied to two transformed strings, it shall return a value greater than, equal to, or less than 0,  
71488 CX corresponding to the result of *strcoll()* or *strcoll\_l()*, respectively, applied to the same two  
71489 CX original strings with the same locale. No more than *n* bytes are placed into the resulting array  
71490 pointed to by *s1*, including the terminating NUL character. If *n* is 0, *s1* is permitted to be a null  
71491 pointer. If copying takes place between objects that overlap, the behavior is undefined.71492 CX The *strxfrm()* and *strxfrm\_l()* functions shall not change the setting of *errno* if successful.71493 Since no return value is reserved to indicate an error, an application wishing to check for error  
71494 CX situations should set *errno* to 0, then call *strxfrm()* or *strxfrm\_l()*, then check *errno*.71495 CX The behavior is undefined if the *locale* argument to *strxfrm\_l()* is the special locale object  
71496 LC\_GLOBAL\_LOCALE or is not a valid locale object handle.71497 **RETURN VALUE**71498 CX Upon successful completion, *strxfrm()* and *strxfrm\_l()* shall return the length of the  
71499 transformed string (not including the terminating NUL character). If the value returned is *n* or  
71500 more, the contents of the array pointed to by *s1* are unspecified.71501 CX On error, *strxfrm()* and *strxfrm\_l()* may set *errno* but no return value is reserved to indicate an  
71502 error.71503 **ERRORS**

71504 These functions may fail if:

71505 CX [EINVAL] The string pointed to by the *s2* argument contains characters outside the  
71506 domain of the collating sequence.71507 **EXAMPLES**

71508 None.

71509 **APPLICATION USAGE**71510 The transformation function is such that two transformed strings can be ordered by *strcmp()* as  
71511 appropriate to collating sequence information in the current locale (category *LC\_COLLATE*).71512 The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the  
71513 *s1* array prior to making the transformation.

71514 **RATIONALE**

71515 None.

71516 **FUTURE DIRECTIONS**

71517 None.

71518 **SEE ALSO**71519 [strcmp\(\)](#), [strcoll\(\)](#)71520 XBD <[string.h](#)>71521 **CHANGE HISTORY**

71522 First released in Issue 3. Included for alignment with the ISO C standard.

71523 **Issue 5**71524 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.71525 **Issue 6**

71526 Extensions beyond the ISO C standard are marked.

71527 The following new requirements on POSIX implementations derive from alignment with the  
71528 Single UNIX Specification:

- 71529
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
71530 added if no conversion could be performed.

71531 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.71532 **Issue 7**71533 The *strxfrm\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
71534 Set Part 4.71535 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0624 [283], XSH/TC1-2008/0625 [283],  
71536 and XSH/TC1-2008/0626 [302] are applied.

71537 **NAME**

71538 swab — swap bytes

71539 **SYNOPSIS**

```
71540 XSI #include <unistd.h>
71541 void swab(const void *restrict src, void *restrict dest,
71542          ssize_t nbytes);
```

71543 **DESCRIPTION**

71544 The *swab()* function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to  
71545 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab()*  
71546 copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If  
71547 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is  
71548 negative, *swab()* does nothing.

71549 **RETURN VALUE**

71550 None.

71551 **ERRORS**

71552 No errors are defined.

71553 **EXAMPLES**

71554 None.

71555 **APPLICATION USAGE**

71556 None.

71557 **RATIONALE**

71558 None.

71559 **FUTURE DIRECTIONS**

71560 None.

71561 **SEE ALSO**71562 XBD <[unistd.h](#)>71563 **CHANGE HISTORY**

71564 First released in Issue 1. Derived from Issue 1 of the SVID.

71565 **Issue 6**

71566 The **restrict** keyword is added to the *swab()* prototype for alignment with the  
71567 ISO/IEC 9899:1999 standard.

71568 **NAME**

71569 swprintf — print formatted wide-character output

71570 **SYNOPSIS**

71571 #include <stdio.h>

71572 #include <wchar.h>

71573 int swprintf(wchar\_t \*restrict *ws*, size\_t *n*,

71574 const wchar\_t \*restrict *format*, ...);

71575 **DESCRIPTION**

71576 Refer to *fwprintf()*.



71577 **NAME**

71578 swscanf — convert formatted wide-character input

71579 **SYNOPSIS**

71580 #include &lt;stdio.h&gt;

71581 #include &lt;wchar.h&gt;

71582 int swscanf(const wchar\_t \*restrict ws,

71583 const wchar\_t \*restrict format, ...);

71584 **DESCRIPTION**71585 Refer to *fwscanf()*.

71586 **NAME**71587 `symlink, symlinkat` — make a symbolic link71588 **SYNOPSIS**71589 `#include <unistd.h>`71590 `int symlink(const char *path1, const char *path2);`71591 OH `#include <fcntl.h>`71592 `int symlinkat(const char *path1, int fd, const char *path2);`71593 **DESCRIPTION**

71594 The `symlink()` function shall create a symbolic link called `path2` that contains the string pointed to  
71595 by `path1` (`path2` is the name of the symbolic link created, `path1` is the string contained in the  
71596 symbolic link).

71597 The string pointed to by `path1` shall be treated only as a string and shall not be validated as a  
71598 pathname.

71599 If the `symlink()` function fails for any reason other than [EIO], any file named by `path2` shall be  
71600 unaffected.

71601 If `path2` names a symbolic link, `symlink()` shall fail and set `errno` to [EEXIST].

71602 The symbolic link's user ID shall be set to the process' effective user ID. The symbolic link's  
71603 group ID shall be set to the group ID of the parent directory or to the effective group ID of the  
71604 process. Implementations shall provide a way to initialize the symbolic link's group ID to the  
71605 group ID of the parent directory. Implementations may, but need not, provide an  
71606 implementation-defined way to initialize the symbolic link's group ID to the effective group ID  
71607 of the calling process.

71608 The values of the file mode bits for the created symbolic link are unspecified. All interfaces  
71609 specified by POSIX.1-2024 shall behave as if the contents of symbolic links can always be read,  
71610 except that the value of the file mode bits returned in the `st_mode` field of the `stat` structure is  
71611 unspecified.

71612 Upon successful completion, `symlink()` shall mark for update the last data access, last data  
71613 modification, and last file status change timestamps of the symbolic link. Also, the last data  
71614 modification and last file status change timestamps of the directory that contains the new entry  
71615 shall be marked for update.

71616 The `symlinkat()` function shall be equivalent to the `symlink()` function except in the case where  
71617 `path2` specifies a relative path. In this case the symbolic link is created relative to the directory  
71618 associated with the file descriptor `fd` instead of the current working directory. If the access mode  
71619 of the open file description associated with the file descriptor is not `O_SEARCH`, the function  
71620 shall check whether directory searches are permitted using the current permissions of the  
71621 directory underlying the file descriptor. If the access mode is `O_SEARCH`, the function shall not  
71622 perform the check.

71623 If `symlinkat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working  
71624 directory shall be used and the behavior shall be identical to a call to `symlink()`.

71625 **RETURN VALUE**

71626 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
71627 return `-1` and set `errno` to indicate the error.

71628 **ERRORS**

71629 These functions shall fail if:

71630 [EACCES] Write permission is denied in the directory where the symbolic link is being  
 71631 created, or search permission is denied for a component of the path prefix of  
 71632 *path2*.

71633 [EEXIST] The *path2* argument names an existing file.

71634 [EILSEQ] The last pathname component of *path2* is not a portable filename, and cannot  
 71635 be created in the target directory.

71636 [EIO] An I/O error occurs while reading from or writing to the file system.

71637 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path2*  
 71638 argument.

71639 [ENAMETOOLONG]

71640 The length of a component of the pathname specified by the *path2* argument is  
 71641 longer than {NAME\_MAX} or the length of the *path1* argument is longer than  
 71642 {SYMLINK\_MAX}.

71643 [ENOENT] A component of the path prefix of *path2* does not name an existing file or *path2*  
 71644 is an empty string.

71645 [ENOENT] or [ENOTDIR]

71646 The *path2* argument contains at least one non-`<slash>` character and ends with  
 71647 one or more trailing `<slash>` characters. If *path2* without the trailing `<slash>`  
 71648 characters would name an existing file, an [ENOENT] error shall not occur.

71649 [ENOSPC] The directory in which the entry for the new symbolic link is being placed  
 71650 cannot be extended because no space is left on the file system containing the  
 71651 directory, or the new symbolic link cannot be created because no space is left  
 71652 on the file system which shall contain the link, or the file system is out of file-  
 71653 allocation resources.

71654 [ENOTDIR] A component of the path prefix of *path2* names an existing file that is neither a  
 71655 directory nor a symbolic link to a directory.

71656 [EROFS] The new symbolic link would reside on a read-only file system.

71657 The *symlinkat()* function shall fail if:

71658 [EACCES] The access mode of the open file description associated with *fd* is not  
 71659 O\_SEARCH and the permissions of the directory underlying *fd* do not permit  
 71660 directory searches.

71661 [EBADF] The *path2* argument does not specify an absolute path and the *fd* argument is  
 71662 neither AT\_FDCWD nor a valid file descriptor open for reading or searching.

71663 [ENOTDIR] The *path2* argument is not an absolute path and *fd* is a file descriptor  
 71664 associated with a non-directory file.

71665 These functions may fail if:

71666 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 71667 resolution of the *path2* argument.

71668 [ENAMETOOLONG]

71669 The length of the *path2* argument exceeds {PATH\_MAX} or pathname  
 71670 resolution of a symbolic link in the *path2* argument produced an intermediate

71671 result with a length that exceeds {PATH\_MAX}.

71672 **EXAMPLES**

71673 None.

71674 **APPLICATION USAGE**

71675 Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a  
71676 hard link guarantees the existence of a file, even after the original name has been removed. A  
71677 symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not  
71678 exist when the link is created. A symbolic link can cross file system boundaries.

71679 Normal permission checks are made on each component of the symbolic link pathname during  
71680 its resolution.

71681 **RATIONALE**

71682 The purpose of the *symlinkat()* function is to create symbolic links in directories other than the  
71683 current working directory without exposure to race conditions. Any part of the path of a file  
71684 could be changed in parallel to a call to *symlink()*, resulting in unspecified behavior. By opening  
71685 a file descriptor for the target directory and using the *symlinkat()* function it can be guaranteed  
71686 that the created symbolic link is located relative to the desired directory.

71687 Implementations are encouraged to have *symlink()* and *symlinkat()* report an [EILSEQ] error if  
71688 the last component of *path2* contains any bytes that have the encoded value of a <newline>  
71689 character.

71690 **FUTURE DIRECTIONS**

71691 None.

71692 **SEE ALSO**

71693 *fdopendir()*, *fstatat()*, *lchown()*, *link()*, *open()*, *readlink()*, *rename()*, *unlink()*

71694 XBD <fcntl.h>, <unistd.h>

71695 **CHANGE HISTORY**

71696 First released in Issue 4, Version 2.

71697 **Issue 5**

71698 Moved from X/OPEN UNIX extension to BASE.

71699 **Issue 6**

71700 The following changes were made to align with the IEEE P1003.1a draft standard:

- 71701 • The DESCRIPTION text is updated.
- 71702 • The [ELOOP] optional error condition is added.

71703 **Issue 7**

71704 Austin Group Interpretation 1003.1-2001 #143 is applied.

71705 The *symlinkat()* function is added from The Open Group Technical Standard, 2006, Extended  
71706 API Set Part 2.

71707 Additions have been made describing how *symlink()* sets the user and group IDs and file mode  
71708 of the symbolic link, and its effect on timestamps.

71709 Changes are made to allow a directory to be opened for searching.

71710 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0627 [146,428], XSH/TC1-2008/0628  
71711 [461], XSH/TC1-2008/0629 [146,428], XSH/TC1-2008/0630 [146,428,436], XSH/TC1-2008/0631  
71712 [324], XSH/TC1-2008/0632 [278], XSH/TC1-2008/0633 [278], and XSH/TC1-2008/0634 [151] are  
71713 applied.

71714 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0357 [873], XSH/TC2-2008/0358 [591],  
71715 XSH/TC2-2008/0359 [641], XSH/TC2-2008/0360 [817], XSH/TC2-2008/0361 [822],  
71716 XSH/TC2-2008/0362 [817], and XSH/TC2-2008/0363 [591] are applied.

71717 **Issue 8**

71718 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
71719 filenames containing any bytes that have the encoded value of a <newline> character.

71720 Austin Group Defect 293 is applied, adding the [EILSEQ] error.

71721 **NAME**

71722 sync — schedule file system updates

71723 **SYNOPSIS**

```
71724 XSI #include <unistd.h>  
71725 void sync(void);
```

71726 **DESCRIPTION**71727 The *sync()* function shall cause all information in memory that updates file systems to be  
71728 scheduled for writing out to all file systems.71729 The writing, although scheduled, is not necessarily complete upon return from *sync()*.71730 **RETURN VALUE**71731 The *sync()* function shall not return a value.71732 **ERRORS**

71733 No errors are defined.

71734 **EXAMPLES**

71735 None.

71736 **APPLICATION USAGE**

71737 None.

71738 **RATIONALE**

71739 None.

71740 **FUTURE DIRECTIONS**

71741 None.

71742 **SEE ALSO**71743 [fsync\(\)](#)71744 XBD [<unistd.h>](#)71745 **CHANGE HISTORY**

71746 First released in Issue 4, Version 2.

71747 **Issue 5**

71748 Moved from X/OPEN UNIX extension to BASE.

71749 **NAME**

71750 sysconf — get configurable system variables

71751 **SYNOPSIS**

71752 #include &lt;unistd.h&gt;

71753 long sysconf(int name);

71754 **DESCRIPTION**

71755 The *sysconf()* function provides a method for the application to determine the current value of a  
 71756 configurable system limit or option (*variable*). The implementation shall support all of the  
 71757 variables listed in the following table and may support others.

71758 The *name* argument represents the system variable to be queried. The following table lists the  
 71759 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,  
 71760 and the symbolic constants defined in <unistd.h> that are the corresponding values used for  
 71761 *name*.

71762	Variable	Value of Name
71763	{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
71764	{AIO_MAX}	_SC_AIO_MAX
71765	{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
71766	{ARG_MAX}	_SC_ARG_MAX
71767	{ATEXIT_MAX}	_SC_ATEXIT_MAX
71768	{BC_BASE_MAX}	_SC_BC_BASE_MAX
71769	{BC_DIM_MAX}	_SC_BC_DIM_MAX
71770	{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
71771	{BC_STRING_MAX}	_SC_BC_STRING_MAX
71772	{CHILD_MAX}	_SC_CHILD_MAX
71773	Clock ticks/second	_SC_CLK_TCK
71774	{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
71775	{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
71776	{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
71777	{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
71778	{IOV_MAX}	_SC_IOV_MAX
71779	{LINE_MAX}	_SC_LINE_MAX
71780	{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
71781	{NGROUPS_MAX}	_SC_NGROUPS_MAX
71782	Initial size of <i>getgrgid_r()</i> and	_SC_GETGR_R_SIZE_MAX
71783	<i>getgrnam_r()</i> data buffers	
71784	Initial size of <i>getpwuid_r()</i> and	_SC_GETPW_R_SIZE_MAX
71785	<i>getpwnam_r()</i> data buffers	
71786	{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
71787	{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
71788	Maximum number of execution units that can	_SC_NPROCESSORS_CONF
71789	be made available to run threads†	
71790	Maximum number of execution units	_SC_NPROCESSORS_ONLN
71791	currently available to run threads†	
71792	Highest supported signal number +1	_SC_NSIG
71793	{OPEN_MAX}	_SC_OPEN_MAX
71794	{PAGE_SIZE}	_SC_PAGE_SIZE
71795	{PAGESIZE}	_SC_PAGESIZE
71796	{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS

	Variable	Value of Name
71797		
71798	{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
71799	{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
71800	{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
71801	{RE_DUP_MAX}	_SC_RE_DUP_MAX
71802	{RTSIG_MAX}	_SC_RTSIG_MAX
71803	{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
71804	{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
71805	{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
71806	{STREAM_MAX}	_SC_STREAM_MAX
71807	{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
71808	{TIMER_MAX}	_SC_TIMER_MAX
71809	{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
71810	{TZNAME_MAX}	_SC_TZNAME_MAX
71811	_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
71812	_POSIX_BARRIERS	_SC_BARRIERS
71813	_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
71814	_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
71815	_POSIX_CPUTIME	_SC_CPUTIME
71816	_POSIX_DEVICE_CONTROL	_SC_DEVICE_CONTROL
71817	_POSIX_FSYNC	_SC_FSYNC
71818	_POSIX_IPV6	_SC_IPV6
71819	_POSIX_JOB_CONTROL	_SC_JOB_CONTROL
71820	_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
71821	_POSIX_MEMLOCK	_SC_MEMLOCK
71822	_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
71823	_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
71824	_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
71825	_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
71826	_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
71827	_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
71828	_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
71829	_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
71830	_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
71831	_POSIX_REGEX	_SC_REGEX
71832	_POSIX_SAVED_IDS	_SC_SAVED_IDS
71833	_POSIX_SEMAPHORES	_SC_SEMAPHORES
71834	_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
71835	_POSIX_SHELL	_SC_SHELL
71836	_POSIX_SPAWN	_SC_SPAWN
71837	_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
71838	_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
71839	_POSIX_SS_REPL_MAX	_SC_SS_REPL_MAX
71840	_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
71841	_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
71842	_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
71843	_POSIX_THREAD_CPUTIME	_SC_THREAD_CPUTIME
71844	_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
71845	_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
71846	_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
71847	_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED



	Variable	Value of Name
71848		
71849	_POSIX_THREAD_ROBUST_PRIO_INHERIT	_SC_THREAD_ROBUST_PRIO_INHERIT
71850	_POSIX_THREAD_ROBUST_PRIO_PROTECT	_SC_THREAD_ROBUST_PRIO_PROTECT
71851	_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
71852	_POSIX_THREAD_SPORADIC_SERVER	_SC_THREAD_SPORADIC_SERVER
71853	_POSIX_THREADS	_SC_THREADS
71854	_POSIX_TIMEOUTS	_SC_TIMEOUTS
71855	_POSIX_TIMERS	_SC_TIMERS
71856	_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
71857	_POSIX_VERSION	_SC_VERSION
71858	_POSIX_V8_ILP32_OFF32	_SC_V8_ILP32_OFF32
71859	_POSIX_V8_ILP32_OFFBIG	_SC_V8_ILP32_OFFBIG
71860	_POSIX_V8_LP64_OFF64	_SC_V8_LP64_OFF64
71861	_POSIX_V8_LPBIG_OFFBIG	_SC_V8_LPBIG_OFFBIG
71862	OB _POSIX_V7_ILP32_OFF32	_SC_V7_ILP32_OFF32
71863	_POSIX_V7_ILP32_OFFBIG	_SC_V7_ILP32_OFFBIG
71864	_POSIX_V7_LP64_OFF64	_SC_V7_LP64_OFF64
71865	_POSIX_V7_LPBIG_OFFBIG	_SC_V7_LPBIG_OFFBIG
71866	_POSIX2_C_BIND	_SC_2_C_BIND
71867	_POSIX2_C_DEV	_SC_2_C_DEV
71868	_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
71869	_POSIX2_FORT_RUN	_SC_2_FORT_RUN
71870	_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
71871	_POSIX2_SW_DEV	_SC_2_SW_DEV
71872	_POSIX2_UPE	_SC_2_UPE
71873	_POSIX2_VERSION	_SC_2_VERSION
71874	_XOPEN_CRYPT	_SC_XOPEN_CRYPT
71875	_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
71876	_XOPEN_REALTIME	_SC_XOPEN_REALTIME
71877	_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
71878	_XOPEN_SHM	_SC_XOPEN_SHM
71879	_XOPEN_UNIX	_SC_XOPEN_UNIX
71880	_XOPEN_UUCP	_SC_XOPEN_UUCP
71881	_XOPEN_VERSION	_SC_XOPEN_VERSION

71882 † The nature of an execution unit and the precise conditions under which an execution unit is  
71883 considered to be available, or can be made available, or how many threads it can execute in  
71884 parallel, are implementation-defined.

## 71885 RETURN VALUE

71886 If *name* is an invalid value, *sysconf()* shall return  $-1$  and set *errno* to indicate the error. If the  
71887 variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum value and  
71888 the variable has no limit, *sysconf()* shall return  $-1$  without changing the value of *errno*. Note that  
71889 indefinite limits do not imply infinite limits; see **<limits.h>**.

71890 Otherwise, *sysconf()* shall return the current variable value on the system. The value returned  
71891 shall not be more restrictive than the corresponding value described to the application when it  
71892 was compiled with the implementation's **<limits.h>** or **<unistd.h>**. The value returned for *name*  
71893 arguments other than *\_SC\_NPROCESSORS\_ONLN* shall not change during the lifetime of the  
71894 calling process, except that *sysconf(\_SC\_OPEN\_MAX)* may return different values before and  
71895 after a call to *setrlimit()* which changes the *RLIMIT\_NOFILE* soft limit.

71896 If the variable corresponding to *name* is dependent on an unsupported option, the results are  
71897 unspecified.

71898 **ERRORS**71899 The *sysconf()* function shall fail if:71900 [EINVAL] The value of the *name* argument is invalid.71901 **EXAMPLES**

71902 None.

71903 **APPLICATION USAGE**71904 As `-1` is a permissible return value in a successful situation, an application wishing to check for  
71905 error situations should set *errno* to 0, then call *sysconf()*, and, if it returns `-1`, check to see if *errno*  
71906 is non-zero.71907 Application developers should check whether an option, such as `_POSIX_SPORADIC_SERVER`,  
71908 is supported prior to obtaining and using values for related variables, such as  
71909 `_POSIX_SS_REPL_MAX`.71910 Although the queries `_SC_NPROCESSORS_CONF` and `_SC_NPROCESSORS_ONLN` provide a  
71911 way for a class of “heavy-load” application to estimate the optimal number of threads that can  
71912 be created to maximize throughput, real-world environments have complications that affect the  
71913 actual efficiency that can be achieved. For example:

- 71914
- There may be more than one “heavy-load” application running on the system.
  - The system may be on battery power, and applications should co-ordinate with the system  
71915 to ensure that a long-running task can pause, resume, and successfully complete even in  
71916 the event of a power outage.
- 71917

71918 In case a portable “heavy-load” application wants to avoid the use of extensions, its developers  
71919 may wish to create threads based on the logical partition of the long-running task, or utilize  
71920 heuristics such as the ratio between CPU time and real time.71921 **RATIONALE**71922 This functionality was added in response to requirements of application developers and of  
71923 system vendors who deal with many international system configurations. It is closely related to  
71924 *pathconf()* and *fpathconf()*.71925 Although a conforming application can run on all systems by never demanding more resources  
71926 than the minimum values published in this volume of POSIX.1-2024, it is useful for that  
71927 application to be able to use the actual value for the quantity of a resource available on any  
71928 given system. To do this, the application makes use of the value of a symbolic constant in  
71929 `<limits.h>` or `<unistd.h>`.71930 However, once compiled, the application must still be able to cope if the amount of resource  
71931 available is increased. To that end, an application may need a means of determining the quantity  
71932 of a resource, or the presence of an option, at execution time.

71933 Two examples are offered:

- 71934
1. Applications may wish to act differently on systems with or without job control.  
71935 Applications vendors who wish to distribute only a single binary package to all instances  
71936 of a computer architecture would be forced to assume job control is never available if it  
71937 were to rely solely on the `<unistd.h>` value published in this volume of POSIX.1-2024.
  2. International applications vendors occasionally require knowledge of the number of clock  
71938 ticks per second. Without these facilities, they would be required to either distribute their  
71939 applications partially in source form or to have 50 Hz and 60 Hz versions for the various  
71940 countries in which they operate.
- 71941

71942 It is the knowledge that many applications are actually distributed widely in executable form  
 71943 that leads to this facility. If limited to the most restrictive values in the headers, such applications  
 71944 would have to be prepared to accept the most limited environments offered by the smallest  
 71945 microcomputers. Although this is entirely portable, there was a consensus that they should be  
 71946 able to take advantage of the facilities offered by large systems, without the restrictions  
 71947 associated with source and object distributions.

71948 During the discussions of this feature, it was pointed out that it is almost always possible for an  
 71949 application to discern what a value might be at runtime by suitably testing the various functions  
 71950 themselves. And, in any event, it could always be written to adequately deal with error returns  
 71951 from the various functions. In the end, it was felt that this imposed an unreasonable level of  
 71952 complication and sophistication on the application developer.

71953 This runtime facility is not meant to provide ever-changing values that applications have to  
 71954 check multiple times. The values are seen as changing no more frequently than once per system  
 71955 initialization, such as by a system administrator or operator with an automatic configuration  
 71956 program. This volume of POSIX.1-2024 specifies that they shall not change within the lifetime of  
 71957 the process.

71958 Some values apply to the system overall and others vary at the file system or directory level. The  
 71959 latter are described in *fpathconf()*.

71960 Note that all values returned must be expressible as integers. String values were considered, but  
 71961 the additional flexibility of this approach was rejected due to its added complexity of  
 71962 implementation and use.

71963 Some values, such as {PATH\_MAX}, are sometimes so large that they must not be used to, say,  
 71964 allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic  
 71965 constant is not even defined in this case.

71966 Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is  
 71967 infinite, returning an error indicating that some other resource limit has been reached is  
 71968 conforming behavior.

#### 71969 **FUTURE DIRECTIONS**

71970 None.

#### 71971 **SEE ALSO**

71972 *confstr()*, *fpathconf()*

71973 XBD <limits.h>, <unistd.h>

71974 XCU *getconf*

#### 71975 **CHANGE HISTORY**

71976 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

#### 71977 **Issue 5**

71978 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
 71979 Threads Extension.

71980 The *\_XBS\_* variables and name values are added to the table of system variables in the  
 71981 DESCRIPTION. These are all marked EX.

#### 71982 **Issue 6**

71983 The symbol CLK\_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks  
 71984 per second”.

71985 The symbol {PASS\_MAX} is removed.

71986 The following changes were made to align with the IEEE P1003.1a draft standard:

- 71987 • Table entries are added for the following variables: `_SC_REGEX`, `_SC_SHELL`,  
71988 `_SC_REGEX_VERSION`, `_SC_SYMLINK_MAX`.

71989 The following `sysconf()` variables and their associated names are added for alignment with  
71990 IEEE Std 1003.1d-1999:

71991 `_POSIX_ADVISORY_INFO`  
71992 `_POSIX_CPUTIME`  
71993 `_POSIX_SPAWN`  
71994 `_POSIX_SPORADIC_SERVER`  
71995 `_POSIX_THREAD_CPUTIME`  
71996 `_POSIX_THREAD_SPORADIC_SERVER`  
71997 `_POSIX_TIMEOUTS`

71998 The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

- 71999 • A statement expressing the dependency of support for some system variables on  
72000 implementation options is added.
- 72001 • The following system variables are added:

72002 `_POSIX_BARRIERS`  
72003 `_POSIX_CLOCK_SELECTION`  
72004 `_POSIX_MONOTONIC_CLOCK`  
72005 `_POSIX_READER_WRITER_LOCKS`  
72006 `_POSIX_SPIN_LOCKS`  
72007 `_POSIX_TYPED_MEMORY_OBJECTS`

72008 The following system variables are added for alignment with IEEE Std 1003.2d-1994:

72009 `_POSIX2_PBS`  
72010 `_POSIX2_PBS_ACCOUNTING`  
72011 `_POSIX2_PBS_LOCATE`  
72012 `_POSIX2_PBS_MESSAGE`  
72013 `_POSIX2_PBS_TRACK`

72014 The following `sysconf()` variables and their associated names are added for alignment with  
72015 IEEE Std 1003.1q-2000:

72016 `_POSIX_TRACE`  
72017 `_POSIX_TRACE_EVENT_FILTER`  
72018 `_POSIX_TRACE_INHERIT`  
72019 `_POSIX_TRACE_LOG`

72020 The macros associated with the `c89` programming models are marked `LEGACY`, and new  
72021 equivalent macros associated with `c99` are introduced.

72022 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the  
72023 DESCRIPTION to denote that the `_PC*` and `_SC*` symbols are now required to be supported. A  
72024 corresponding change has been made in the Base Definitions volume of POSIX.1-2024. The  
72025 deletion in the second paragraph removes some duplicated text. Additional symbols that were  
72026 erroneously omitted from this reference page have been added.

72027 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the  
72028 RETURN VALUE section that the value returned for `sysconf(_SC_OPEN_MAX)` may change if a

- 72029 call to *setrlimit()* adjusts the RLIMIT\_NOFILE soft limit.
- 72030 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/134 is applied, updating the  
72031 DESCRIPTION to remove an erroneous entry for \_POSIX\_SYMLINK\_MAX. This corrects an  
72032 error in IEEE Std 1003.1-2001/Cor 1-2002.
- 72033 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/135 is applied, removing  
72034 \_POSIX\_FILE\_LOCKING, \_POSIX\_MULTI\_PROCESS, \_POSIX2\_C\_VERSION, and  
72035 \_XOPEN\_XCU\_VERSION (and their associated \_SC\_\* variables) from the DESCRIPTION and  
72036 APPLICATION USAGE sections.
- 72037 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/136 is applied, adding the following  
72038 constants (and their associated \_SC\_\* variables) to the DESCRIPTION:
- 72039        \_POSIX\_SS\_REPL\_MAX  
72040        \_POSIX\_TRACE\_EVENT\_NAME\_MAX  
72041        \_POSIX\_TRACE\_NAME\_MAX  
72042        \_POSIX\_TRACE\_SYS\_MAX  
72043        \_POSIX\_TRACE\_USER\_EVENT\_MAX
- 72044 The RETURN VALUE and APPLICATION USAGE sections are updated to note that if variables  
72045 are dependent on unsupported options, the results are unspecified.
- 72046 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/137 is applied, removing  
72047 \_REGEX\_VERSION and \_SC\_REGEX\_VERSION.
- 72048 **Issue 7**
- 72049 Austin Group Interpretation 1003.1-2001 #160 is applied.
- 72050 SD5-XSH-ERN-166 is applied, changing “Maximum size” to “Initial size” for the “Maximum  
72051 size of ...” entries in the table in the DESCRIPTION.
- 72052 The variables for the supported programming environments are updated to be V7 and the  
72053 LEGACY variables are removed.
- 72054 The following constants are added:
- 72055        \_POSIX\_THREAD\_ROBUST\_PRIO\_INHERIT  
72056        \_POSIX\_THREAD\_ROBUST\_PRIO\_PROTECT
- 72057 The \_XOPEN\_UUCP variable and its associated \_SC\_XOPEN\_UUCP value is added to the table  
72058 of system variables.
- 72059 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0364 [752] is applied.
- 72060 **Issue 8**
- 72061 Austin Group Defect 51 is applied, moving the *getrlimit()* and *setrlimit()* functions, excluding  
72062 the RLIMIT\_CPU and RLIMIT\_FSIZE limits, from the XSI option to the Base.
- 72063 Austin Group Defects 339 and 1608 are applied, adding \_SC\_NPROCESSORS\_CONF and  
72064 \_SC\_NPROCESSORS\_ONLN.
- 72065 Austin Group Defect 729 is applied, adding \_SC\_DEVICE\_CONTROL.
- 72066 Austin Group Defect 741 is applied, adding \_SC\_NSIG.
- 72067 Austin Group Defect 1330 is applied, removing obsolescent interfaces and changing “\_V7\_” to  
72068 “\_V8\_” and “\_V6\_” to “\_V7\_”.

72069 **NAME**

72070 syslog — log a message

72071 **SYNOPSIS**

72072 XSI `#include <syslog.h>`

72073 `void syslog(int priority, const char *message, ... /* argument */);`

72074 **DESCRIPTION**

72075 Refer to [closelog\(\)](#).

72076 **NAME**

72077 system — issue a command

72078 **SYNOPSIS**

72079 #include &lt;stdlib.h&gt;

72080 int system(const char \*command);

72081 **DESCRIPTION**

72082 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 72083 conflict between the requirements described here and the ISO C standard is unintentional. This  
 72084 volume of POSIX.1-2024 defers to the ISO C standard.

72085 If *command* is a null pointer, the *system()* function shall determine whether the host environment  
 72086 has a command processor. If *command* is not a null pointer, the *system()* function shall pass the  
 72087 string pointed to by *command* to that command processor to be executed in an implementation-  
 72088 defined manner; this might then cause the program calling *system()* to behave in a non-  
 72089 conforming manner or to terminate.

72090 CX The *system()* function shall behave as if a child process were created using *fork()*, and the child  
 72091 process invoked the *sh* utility using *execl()* as follows:

72092 

```
execl(<shell path>, "sh", "-c", "--", command, (char *)0);
```

72093 where <shell path> is an unspecified pathname for the *sh* utility. It is implementation-defined  
 72094 whether the handlers registered with *pthread\_atfork()* are called as part of the creation of the  
 72095 child process.

72096 The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the  
 72097 SIGCHLD signal, while waiting for the command to terminate. If this might cause the  
 72098 application to miss a signal that would have killed it, then the application should examine the  
 72099 return value from *system()* and take whatever action is appropriate to the application if the  
 72100 command terminated due to receipt of a signal.

72101 The *system()* function shall not affect the termination status of any child of the calling processes  
 72102 other than the process or processes it itself creates.

72103 The *system()* function shall not return until the child process has terminated.

72104 If concurrent calls to *system()* are made from multiple threads, it is unspecified whether:

- 72105 • each call saves and restores the dispositions of the SIGINT and SIGQUIT signals  
 72106 independently, or
- 72107 • in a set of concurrent calls the dispositions in effect after the last call returns are those that  
 72108 were in effect on entry to the first call.

72109 If a thread is cancelled while it is in a call to *system()*, it is unspecified whether the child process  
 72110 is terminated and waited for, or is left running.

72111 **RETURN VALUE**

72112 If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor  
 72113 CX is available, or zero if none is available. The *system()* function shall always return non-zero  
 72114 when *command* is NULL.

72115 CX If *command* is not a null pointer, *system()* shall return the termination status of the command  
 72116 language interpreter in the format specified by *waitpid()*. The termination status shall be as  
 72117 defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents  
 72118 the command language interpreter from executing after the child process is created, the return  
 72119 value from *system()* shall be as if the command language interpreter had terminated using

72120 `exit(127)` or `_exit(127)`. If a child process cannot be created, or if the termination status for the  
 72121 command language interpreter cannot be obtained, `system()` shall return `-1` and set `errno` to  
 72122 indicate the error.

## 72123 ERRORS

72124 CX The `system()` function may set `errno` values as described by `fork()`.

72125 In addition, `system()` may fail if:

72126 CX [ECHILD] The status of the child process created by `system()` is no longer available.

## 72127 EXAMPLES

72128 None.

## 72129 APPLICATION USAGE

72130 If the return value of `system()` is not `-1`, its value can be decoded through the use of the macros  
 72131 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

72132 Note that, while `system()` must ignore `SIGINT` and `SIGQUIT` and block `SIGCHLD` while waiting  
 72133 for the child to terminate, the handling of signals in the executed command is as specified by  
 72134 `fork()` and `exec`. For example, if `SIGINT` is being caught or is set to `SIG_DFL` when `system()` is  
 72135 called, then the child is started with `SIGINT` handling set to `SIG_DFL`.

72136 Ignoring `SIGINT` and `SIGQUIT` in the parent process prevents coordination problems (two  
 72137 processes reading from the same terminal, for example) when the executed command ignores or  
 72138 catches one of the signals. It is also usually the correct action when the user has given a  
 72139 command to the application to be executed synchronously (as in the `!` command in many  
 72140 interactive applications). In either case, the signal should be delivered only to the child process,  
 72141 not to the application itself. There is one situation where ignoring the signals might have less  
 72142 than the desired effect. This is when the application uses `system()` to perform some task invisible  
 72143 to the user. If the user typed the interrupt character ("`^C`", for example) while `system()` is being  
 72144 used in this way, one would expect the application to be killed, but only the executed command  
 72145 is killed. Applications that use `system()` in this way should carefully check the return status from  
 72146 `system()` to see if the executed command was successful, and should take appropriate action  
 72147 when the command fails.

72148 Blocking `SIGCHLD` while waiting for the child to terminate prevents the application from  
 72149 catching the signal and obtaining status from `system()`'s child process before `system()` can get the  
 72150 status itself.

72151 The context in which the utility is ultimately executed may differ from that in which `system()`  
 72152 was called. For example, file descriptors that have the `FD_CLOEXEC` or `FD_CLOFORK` flag set  
 72153 are closed, and the process ID and parent process ID are different. Also, if the executed utility  
 72154 changes its environment variables or its current working directory, that change is not reflected in  
 72155 the caller's context.

72156 There is no defined way for an application to find the specific path for the shell. However,  
 72157 `confstr()` can provide a value for `PATH` that is guaranteed to find the `sh` utility.

72158 Although `system()` is required to be thread-safe, it is recommended that concurrent calls from  
 72159 multiple threads are avoided, since `system()` is not required to coordinate the saving and  
 72160 restoring of the dispositions of the `SIGINT` and `SIGQUIT` signals across a set of overlapping  
 72161 calls, and therefore the signals might end up being set to ignored after the last call returns.  
 72162 Applications should also avoid cancelling a thread while it is in a call to `system()` as the child  
 72163 process may be left running in that event. In addition, if another thread alters the disposition of  
 72164 the `SIGCHLD` signal, a call to `signal()` may produce unexpected results.



72165 **RATIONALE**

72166 The *system()* function should not be used by programs that have set user (or group) ID  
 72167 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used  
 72168 instead. This prevents any unforeseen manipulation of the environment of the user that could  
 72169 cause execution of commands not anticipated by the calling program.

72170 There are three levels of specification for the *system()* function. The ISO C standard gives the  
 72171 most basic. It requires that the function exists, and defines a way for an application to query  
 72172 whether a command language interpreter exists. It says nothing about the command language  
 72173 or the environment in which the command is interpreted.

72174 POSIX.1-2024 places additional restrictions on *system()*. It requires that if there is a command  
 72175 language interpreter, the environment must be as specified by *fork()* and *exec*. This ensures, for  
 72176 example, that close-on-exec works, that process-owned file locks are not inherited, and that the  
 72177 process ID is different. It also specifies the return value from *system()* when the command line  
 72178 can be run, thus giving the application some information about the command's completion  
 72179 status.

72180 Finally, POSIX.1-2024 requires the command to be interpreted as in the shell command language  
 72181 defined in the Shell and Utilities volume of POSIX.1-2024.

72182 Note that, *system(NULL)* is required to return non-zero, indicating that there is a command  
 72183 language interpreter. At first glance, this would seem to conflict with the ISO C standard which  
 72184 allows *system(NULL)* to return zero. There is no conflict, however. A system must have a  
 72185 command language interpreter, and is non-conforming if none is present. It is therefore  
 72186 permissible for the *system()* function on such a system to implement the behavior specified by  
 72187 the ISO C standard as long as it is understood that the implementation does not conform to  
 72188 POSIX.1-2024 if *system(NULL)* returns zero.

72189 It was explicitly decided that when *command* is NULL, *system()* should not be required to check  
 72190 to make sure that the command language interpreter actually exists with the correct mode, that  
 72191 there are enough processes to execute it, and so on. The call *system(NULL)* could, theoretically,  
 72192 check for such problems as too many existing child processes, and return zero. However, it  
 72193 would be inappropriate to return zero due to such a (presumably) transient condition. If some  
 72194 condition exists that is not under the control of this application and that would cause any  
 72195 *system()* call to fail, that system has been rendered non-conforming.

72196 Early drafts required, or allowed, *system()* to return with *errno* set to [EINTR] if it was  
 72197 interrupted with a signal. This error return was removed, and a requirement that *system()* not  
 72198 return until the child has terminated was added. This means that if a *waitpid()* call in *system()*  
 72199 exits with *errno* set to [EINTR], *system()* must reissue the *waitpid()*. This change was made for  
 72200 two reasons:

- 72201 1. There is no way for an application to clean up if *system()* returns [EINTR], short of calling  
 72202 *wait()*, and that could have the undesirable effect of returning the status of children other  
 72203 than the one started by *system()*.
- 72204 2. While it might require a change in some historical implementations, those  
 72205 implementations already have to be changed because they use *wait()* instead of *waitpid()*.

72206 Note that if the application is catching SIGCHLD signals, it will receive such a signal before a  
 72207 successful *system()* call returns.

72208 To conform to POSIX.1-2024, *system()* must use *waitpid()*, or some similar function, instead of  
 72209 *wait()*.

72210 The following code sample illustrates how *system()* might be implemented on an

```

72211 implementation conforming to POSIX.1-2024.
72212 int system(const char *cmd)
72213 {
72214     int stat;
72215     pid_t pid;
72216     struct sigaction sa, savintr, savequit;
72217     sigset_t saveblock;
72218     if (cmd == NULL)
72219         return(1);
72220     sa.sa_handler = SIG_IGN;
72221     sigemptyset(&sa.sa_mask);
72222     sa.sa_flags = 0;
72223     sigemptyset(&savintr.sa_mask);
72224     sigemptyset(&savequit.sa_mask);
72225     sigaction(SIGINT, &sa, &savintr);
72226     sigaction(SIGQUIT, &sa, &savequit);
72227     sigaddset(&sa.sa_mask, SIGCHLD);
72228     pthread_sigmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
72229     if ((pid = fork()) == 0) {
72230         sigaction(SIGINT, &savintr, (struct sigaction *)0);
72231         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
72232         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
72233         execl("/bin/sh", "sh", "-c", "--", cmd, (char *)0);
72234         _exit(127);
72235     }
72236     if (pid == -1) {
72237         stat = -1; /* errno comes from fork() */
72238     } else {
72239         while (waitpid(pid, &stat, 0) == -1) {
72240             if (errno != EINTR) {
72241                 stat = -1;
72242                 break;
72243             }
72244         }
72245     }
72246     sigaction(SIGINT, &savintr, (struct sigaction *)0);
72247     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
72248     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
72249     return(stat);
72250 }

```

72251 Note that, while a particular implementation of *system()* (such as the one above) can assume a  
72252 particular path for the shell, such a path is not necessarily valid on another system. The above  
72253 example is not portable, and is not intended to be.

72254 Earlier versions of this standard did not require *system()* to be thread-safe because it alters the  
72255 process-wide disposition of the SIGINT and SIGQUIT signals. It is now required to be thread-  
72256 safe to align with the ISO C standard, which (since the introduction of threads in 2011) requires  
72257 that it avoids data races. However, the function is not required to coordinate the saving and  
72258 restoring of the dispositions of the SIGINT and SIGQUIT signals across a set of overlapping  
72259 calls, and the above example does not do so. The example also does not terminate and wait for  
72260 the child process if the calling thread is cancelled, and so would leak a process ID in that event.

72261 One reviewer suggested that an implementation of *system()* might want to use an environment  
 72262 variable such as *SHELL* to determine which command interpreter to use. The supposed  
 72263 implementation would use the default command interpreter if the one specified by the  
 72264 environment variable was not available. This would allow a user, when using an application that  
 72265 prompts for command lines to be processed using *system()*, to specify a different command  
 72266 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not  
 72267 follow the command line syntax specified in the Shell and Utilities volume of POSIX.1-2024, then  
 72268 changing *SHELL* would render *system()* non-conforming. This would affect applications that  
 72269 expected the specified behavior from *system()*, and since the Shell and Utilities volume of  
 72270 POSIX.1-2024 does not mention that *SHELL* affects *system()*, the application would not know  
 72271 that it needed to unset *SHELL*.

72272 Earlier versions of this standard required the *command* string to be passed as the next argument  
 72273 after "-c" (omitting "--"). This meant that portable applications needed to take care not to  
 72274 pass a command string beginning with <hyphen-minus> ('-') or <plus-sign> ('+'), as it  
 72275 would then be interpreted as containing options. Now that implementations are required to pass  
 72276 the "--", applications no longer need to do this.

#### 72277 FUTURE DIRECTIONS

72278 None.

#### 72279 SEE ALSO

72280 [Section 2.9.5.2](#) (on page 543), *exec*, *pipe()*, *pthread\_atfork()*, *wait()*

72281 XBD [<limits.h>](#), [<signal.h>](#), [<stdlib.h>](#), [<sys/wait.h>](#)

72282 XCU *sh*

#### 72283 CHANGE HISTORY

72284 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 72285 Issue 6

72286 Extensions beyond the ISO C standard are marked.

#### 72287 Issue 7

72288 Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this  
 72289 function and treatment of *at\_fork()* handlers.

72290 Austin Group Interpretation 1003.1-2001 #156 is applied.

72291 SD5-XSH-ERN-30 is applied.

72292 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0365 [627] is applied.

#### 72293 Issue 8

72294 Austin Group Defect 768 is applied, adding OFD-owned file locks.

72295 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
 72296 standard.

72297 Austin Group Defect 1317 is applied, making it implementation-defined whether the handlers  
 72298 registered with *pthread\_atfork()* are called.

72299 Austin Group Defect 1318 is applied, adding *FD\_CLOFORK*.

72300 Austin Group Defect 1440 is applied, adding a "--" argument to the specified *execl()* call.

72301 **NAME**

72302 tan, tanf, tanl — tangent function

72303 **SYNOPSIS**

```
72304 #include <math.h>
72305 double tan(double x);
72306 float tanf(float x);
72307 long double tanl(long double x);
```

72308 **DESCRIPTION**

72309 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 72310 conflict between the requirements described here and the ISO C standard is unintentional. This  
 72311 volume of POSIX.1-2024 defers to the ISO C standard.

72312 These functions shall compute the tangent of their argument  $x$ , measured in radians.

72313 An application wishing to check for error situations should set *errno* to zero and call  
 72314 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 72315 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 72316 zero, an error has occurred.

72317 **RETURN VALUE**

72318 Upon successful completion, these functions shall return the tangent of  $x$ .

72319 MXX If the correct value would cause underflow, and is not representable, a range error may occur,  
 72320 MXX and *tan()*, *tanf()*, and *tanl()* shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an  
 72321 implementation-defined value no greater in magnitude than DBL\_MIN, FLT\_MIN, and  
 72322 LDBL\_MIN, respectively.

72323 MX If  $x$  is NaN, a NaN shall be returned.

72324 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

72325 If  $x$  is  $\pm\text{Inf}$ , a domain error shall occur, and either a NaN (if supported), or an implementation-  
 72326 defined value shall be returned.

72327 MXX If  $x$  is subnormal,  $x$  should be returned.

72328 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *tan()*, *tanf()*, and *tanl()* shall  
 72329 return an implementation-defined value no greater in magnitude than DBL\_MIN, FLT\_MIN,  
 72330 and LDBL\_MIN, respectively.

72331 MXX If the correct value would cause underflow, and is representable, a range error may occur and  
 72332 the correct value shall be returned.

72333 XSI If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*  
 72334 shall return  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  $\pm\text{HUGE\_VALL}$ , respectively, with the same sign  
 72335 as the correct value of the function.

72336 **ERRORS**

72337 These functions shall fail if:

72338 MX **Domain Error** The value of  $x$  is  $\pm\text{Inf}$ .

72339 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 72340 then *errno* shall be set to [EDOM]. If the integer expression (*math\_errhandling*  
 72341 & MATH\_ERREXCEPT) is non-zero, then the invalid floating-point exception  
 72342 shall be raised.

72343 XSI **Range Error** The result overflows  
 72344 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 72345 then *errno* shall be set to [ERANGE]. If the integer expression  
 72346 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the overflow  
 72347 floating-point exception shall be raised.

72348 These functions may fail if:

72349 MX **Range Error** The result underflows, or the value of *x* is subnormal.  
 72350 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
 72351 then *errno* shall be set to [ERANGE]. If the integer expression  
 72352 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
 72353 floating-point exception shall be raised.

## 72354 EXAMPLES

### 72355 Taking the Tangent of a 45-Degree Angle

```
72356 #include <math.h>
72357 ...
72358 double radians = 45.0 * M_PI / 180;
72359 double result;
72360 ...
72361 result = tan (radians);
```

## 72362 APPLICATION USAGE

72363 There are no known floating-point representations such that for a normal argument,  $\tan(x)$  is  
 72364 either overflow or underflow.

72365 These functions may lose accuracy when their argument is near a multiple of  $\pi/2$  or is far from  
 72366 0.0.

72367 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
 72368 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

## 72369 RATIONALE

72370 None.

## 72371 FUTURE DIRECTIONS

72372 None.

## 72373 SEE ALSO

72374 [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

72375 XBD Section 4.23 (on page 109), [<math.h>](#)

## 72376 CHANGE HISTORY

72377 First released in Issue 1. Derived from Issue 1 of the SVID.

### 72378 Issue 5

72379 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
 72380 in previous issues.

### 72381 Issue 6

72382 The [tanf\(\)](#) and [tanl\(\)](#) functions are added for alignment with the ISO/IEC 9899:1999 standard.

72383 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
 72384 revised to align with the ISO/IEC 9899:1999 standard.

- 72385 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
72386 marked.
- 72387 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last  
72388 paragraph in the RETURN VALUE section.
- 72389 **Issue 7**
- 72390 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0635 [68], XSH/TC1-2008/0636 [68],  
72391 and XSH/TC1-2008/0637 [68] are applied.
- 72392 **Issue 8**
- 72393 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when  $x$  is  
72394 subnormal to avoid the need for two shading changes.

72395 **NAME**

72396 tanh, tanhf, tanhl — hyperbolic tangent functions

72397 **SYNOPSIS**

```
72398 #include <math.h>
72399 double tanh(double x);
72400 float tanhf(float x);
72401 long double tanhl(long double x);
```

72402 **DESCRIPTION**

72403 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
72404 conflict between the requirements described here and the ISO C standard is unintentional. This  
72405 volume of POSIX.1-2024 defers to the ISO C standard.

72406 These functions shall compute the hyperbolic tangent of their argument  $x$ .

72407 An application wishing to check for error situations should set *errno* to zero and call  
72408 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
72409 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
72410 zero, an error has occurred.

72411 **RETURN VALUE**

72412 Upon successful completion, these functions shall return the hyperbolic tangent of  $x$ .

72413 MX If  $x$  is NaN, a NaN shall be returned.

72414 If  $x$  is  $\pm 0$ ,  $x$  shall be returned.

72415 If  $x$  is  $\pm\text{Inf}$ ,  $\pm 1$  shall be returned.

72416 MXX If  $x$  is subnormal,  $x$  should be returned.

72417 MX If  $x$  is subnormal, a range error may occur and, if  $x$  is not returned, *tanh()*, *tanhf()*, and *tanhl()*  
72418 shall return an implementation-defined value no greater in magnitude than DBL\_MIN,  
72419 FLT\_MIN, and LDBL\_MIN, respectively.

72420 **ERRORS**

72421 These functions may fail if:

72422 MX **Range Error** The value of  $x$  is subnormal.

72423 If the integer expression (*math\_errhandling* & MATH\_ERRNO) is non-zero,  
72424 then *errno* shall be set to [ERANGE]. If the integer expression  
72425 (*math\_errhandling* & MATH\_ERREXCEPT) is non-zero, then the underflow  
72426 floating-point exception shall be raised.

72427 **EXAMPLES**

72428 None.

72429 **APPLICATION USAGE**

72430 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
72431 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

72432 **RATIONALE**

72433 None.

72434 **FUTURE DIRECTIONS**

72435 None.

72436 **SEE ALSO**72437 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*

72438 XBD Section 4.23 (on page 109), &lt;math.h&gt;

72439 **CHANGE HISTORY**

72440 First released in Issue 1. Derived from Issue 1 of the SVID.

72441 **Issue 5**72442 The DESCRIPTION is updated to indicate how an application should check for an error. This  
72443 text was previously published in the APPLICATION USAGE section.72444 **Issue 6**72445 The *tanhf()* and *tanhfll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.72446 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are  
72447 revised to align with the ISO/IEC 9899:1999 standard.72448 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are  
72449 marked.72450 **Issue 7**

72451 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0638 [68] is applied.

72452 **Issue 8**72453 Austin Group Defect 1382 is applied, rearranging the text describing the behavior when *x* is  
72454 subnormal to avoid the need for two shading changes.



72455 **NAME**

72456 tanl — tangent function

72457 **SYNOPSIS**

72458 #include &lt;math.h&gt;

72459 long double tanl(long double *x*);72460 **DESCRIPTION**72461 Refer to *tan()*.

72462 **NAME**

72463 tcdrain — wait for transmission of output

72464 **SYNOPSIS**

72465 #include &lt;termios.h&gt;

72466 int tcdrain(int *fildev*);72467 **DESCRIPTION**72468 The *tcdrain()* function shall block until all output written to the object referred to by *fildev* is  
72469 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.72470 Any attempts to use *tcdrain()* from a process which is a member of a background process group  
72471 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a  
72472 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring  
72473 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.72474 **RETURN VALUE**72475 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
72476 indicate the error.72477 **ERRORS**72478 The *tcdrain()* function shall fail if:72479 [EBADF] The *fildev* argument is not a valid file descriptor.72480 [EINTR] A signal interrupted *tcdrain()*.72481 [EIO] The process group of the writing process is orphaned, the calling thread is not  
72482 blocking SIGTTOU, and the process is not ignoring SIGTTOU.72483 [ENOTTY] The file associated with *fildev* is not a terminal.72484 **EXAMPLES**

72485 None.

72486 **APPLICATION USAGE**

72487 None.

72488 **RATIONALE**

72489 None.

72490 **FUTURE DIRECTIONS**

72491 None.

72492 **SEE ALSO**72493 [tcflush\(\)](#)

72494 XBD Chapter 11 (on page 199), &lt;termios.h&gt;

72495 **CHANGE HISTORY**

72496 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72497 **Issue 6**

72498 The following new requirements on POSIX implementations derive from alignment with the  
72499 Single UNIX Specification:

72500 • In the DESCRIPTION, the final paragraph is no longer conditional on  
72501 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.

72502 • The [EIO] error is added.

72503 **Issue 7**

72504 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0639 [79], XSH/TC1-2008/0640 [79],  
72505 and XSH/TC1-2008/0641 [79] are applied.

72506 **NAME**

72507 tcflow — suspend or restart the transmission or reception of data

72508 **SYNOPSIS**

72509 #include &lt;termios.h&gt;

72510 int tcflow(int *fildes*, int *action*);72511 **DESCRIPTION**72512 The *tcflow()* function shall suspend or restart transmission or reception of data on the object  
72513 referred to by *fildes*, depending on the value of *action*. The *fildes* argument is an open file  
72514 descriptor associated with a terminal.

- 72515 • If *action* is TCOOFF, output shall be suspended.
- 72516 • If *action* is TCOON, suspended output shall be restarted.
- 72517 • If *action* is TCIOFF and *fildes* refers to a terminal device, the system shall transmit a STOP  
72518 character, which is intended to cause the terminal device to stop transmitting data to the  
72519 system. If *fildes* is associated with a pseudo-terminal, the STOP character need not be  
72520 transmitted.
- 72521 • If *action* is TCION and *fildes* refers to a terminal device, the system shall transmit a START  
72522 character, which is intended to cause the terminal device to start transmitting data to the  
72523 system. If *fildes* is associated with a pseudo-terminal, the START character need not be  
72524 transmitted.

72525 The default on the opening of a terminal file is that neither its input nor its output are  
72526 suspended.

72527 Attempts to use *tcflow()* from a process which is a member of a background process group on a  
72528 *fildes* associated with its controlling terminal, shall cause the process group to be sent a  
72529 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring  
72530 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

72531 **RETURN VALUE**

72532 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
72533 indicate the error.

72534 **ERRORS**72535 The *tcflow()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 72536 | [EBADF]  | The <i>fildes</i> argument is not a valid file descriptor.  |
| 72537 | [EINVAL] | The <i>action</i> argument is not a supported value.  |
| 72538 | [EIO]    | The process group of the writing process is orphaned, the calling thread is not<br>72539 blocking SIGTTOU, and the process is not ignoring SIGTTOU. |
| 72540 | [ENOTTY] | The file associated with <i>fildes</i> is not a terminal.   |

72541 **EXAMPLES**

72542 None.

72543 **APPLICATION USAGE**

72544 None.

72545 **RATIONALE**

72546 None.

72547 **FUTURE DIRECTIONS**

72548 None.

72549 **SEE ALSO**72550 [tcsendbreak\(\)](#)72551 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)72552 **CHANGE HISTORY**

72553 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72554 **Issue 6**72555 The following new requirements on POSIX implementations derive from alignment with the  
72556 Single UNIX Specification:

- 72557
- The [EIO] error is added.

72558 **Issue 7**72559 SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and  
72560 STOP characters.72561 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0642 [79], XSH/TC1-2008/0643 [79],  
72562 and XSH/TC1-2008/0644 [79] are applied.

72563 **NAME**

72564 tcflush — flush non-transmitted output data, non-read input data, or both

72565 **SYNOPSIS**

72566 #include &lt;termios.h&gt;

72567 int tcflush(int *fildev*, int *queue\_selector*);72568 **DESCRIPTION**72569 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildev*  
72570 (an open file descriptor associated with a terminal) but not transmitted, or data received but not  
72571 read, depending on the value of *queue\_selector*:

- 72572 • If *queue\_selector* is TCIFLUSH, it shall flush data received but not read.
- 72573 • If *queue\_selector* is TCOFLUSH, it shall flush data written but not transmitted.
- 72574 • If *queue\_selector* is TCIOFLUSH, it shall flush both data received but not read and data  
72575 written but not transmitted.

72576 Attempts to use *tcflush()* from a process which is a member of a background process group on a  
72577 *fildev* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU  
72578 signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU  
72579 signals, the process shall be allowed to perform the operation, and no signal is sent.

72580 **RETURN VALUE**72581 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
72582 indicate the error.72583 **ERRORS**72584 The *tcflush()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 72585 | [EBADF]  | The <i>fildev</i> argument is not a valid file descriptor.                      |
| 72586 | [EINVAL] | The <i>queue_selector</i> argument is not a supported value.                    |
| 72587 | [EIO]    | The process group of the writing process is orphaned, the calling thread is not |
| 72588 |          | blocking SIGTTOU, and the process is not ignoring SIGTTOU.                      |
| 72589 | [ENOTTY] | The file associated with <i>fildev</i> is not a terminal.                       |

72590 **EXAMPLES**

72591 None.

72592 **APPLICATION USAGE**

72593 None.

72594 **RATIONALE**

72595 None.

72596 **FUTURE DIRECTIONS**

72597 None.

72598 **SEE ALSO**72599 [tcdrain\(\)](#)72600 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)72601 **CHANGE HISTORY**

72602 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72603 **Issue 6**

72604 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE  
72605 sections, references to *tcflow()* are replaced with *tcflush()*.

72606 The following new requirements on POSIX implementations derive from alignment with the  
72607 Single UNIX Specification:

- 72608 • In the DESCRIPTION, the final paragraph is no longer conditional on  
72609 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- 72610 • The [EIO] error is added.

72611 **Issue 7**

72612 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0645 [79], XSH/TC1-2008/0646 [79],  
72613 and XSH/TC1-2008/0647 [79] are applied.

72614 **NAME**

72615 tcgetattr — get the parameters associated with the terminal

72616 **SYNOPSIS**

72617 #include &lt;termios.h&gt;

72618 int tcgetattr(int *fildev*, struct termios \**termios\_p*);72619 **DESCRIPTION**72620 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev*  
72621 and store them in the **termios** structure referenced by *termios\_p*. The *fildev* argument is an open  
72622 file descriptor associated with a terminal.72623 The *termios\_p* argument is a pointer to a **termios** structure.72624 The *tcgetattr()* operation is allowed from any process.72625 If the terminal device supports different input and output baud rates, the baud rates stored in  
72626 the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are  
72627 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall  
72628 be the actual baud rate. If the terminal device does not support split baud rates, the input baud  
72629 rate stored in the **termios** structure shall be the output rate (as one of the symbolic values).72630 **RETURN VALUE**72631 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
72632 indicate the error.72633 **ERRORS**72634 The *tcgetattr()* function shall fail if:72635 [EBADF] The *fildev* argument is not a valid file descriptor.72636 [ENOTTY] The file associated with *fildev* is not a terminal.72637 **EXAMPLES**

72638 None.

72639 **APPLICATION USAGE**

72640 None.

72641 **RATIONALE**72642 Care must be taken when changing the terminal attributes. Applications should always do a  
72643 *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()*, changing only  
72644 the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the  
72645 terminal state whenever it is done with the terminal. This is necessary because terminal  
72646 attributes apply to the underlying port and not to each individual open instance; that is, all  
72647 processes that have used the terminal see the latest attribute changes.72648 A program that uses these functions should be written to catch all signals and take other  
72649 appropriate actions to ensure that when the program terminates, whether planned or not, the  
72650 terminal device's state is restored to its original state.72651 Existing practice dealing with error returns when only part of a request can be honored is based  
72652 on calls to the *ioctl()* function. In historical BSD and System V implementations, the  
72653 corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if  
72654 some of the requested changes could not be made. Many existing applications assume this  
72655 behavior and would no longer work correctly if the return value were changed from zero to -1  
72656 in this case.

72657 Note that either specification has a problem. When zero is returned, it implies everything



72658 succeeded even if some of the changes were not made. When -1 is returned, it implies  
72659 everything failed even though some of the changes were made.

72660 Applications that need all of the requested changes made to work properly should follow  
72661 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

72662 **FUTURE DIRECTIONS**

72663 None.

72664 **SEE ALSO**

72665 [\*tcsetattr\(\)\*](#)

72666 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)

72667 **CHANGE HISTORY**

72668 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72669 **Issue 6**

72670 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.  
72671 Previously, the number zero was also allowed but was obsolescent.

72672 **NAME**

72673 tcgetpgrp — get the foreground process group ID

72674 **SYNOPSIS**

72675 #include &lt;unistd.h&gt;

72676 pid\_t tcgetpgrp(int *fildev*);72677 **DESCRIPTION**72678 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process  
72679 group associated with the terminal.72680 If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does  
72681 not match the process group ID of any existing process group.72682 The *tcgetpgrp()* function is allowed from a process that is a member of a background process  
72683 group; however, the information may be subsequently changed by a process that is a member of  
72684 a foreground process group.72685 **RETURN VALUE**72686 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the  
72687 foreground process associated with the terminal. Otherwise,  $-1$  shall be returned and *errno* set to  
72688 indicate the error.72689 **ERRORS**72690 The *tcgetpgrp()* function shall fail if:72691 [EBADF] The *fildev* argument is not a valid file descriptor.72692 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
72693 controlling terminal.72694 **EXAMPLES**

72695 None.

72696 **APPLICATION USAGE**

72697 None.

72698 **RATIONALE**

72699 None.

72700 **FUTURE DIRECTIONS**

72701 None.

72702 **SEE ALSO**72703 [setsid\(\)](#), [setpgid\(\)](#), [tcsetpgrp\(\)](#)72704 XBD [<sys/types.h>](#), [<unistd.h>](#)72705 **CHANGE HISTORY**

72706 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72707 **Issue 6**72708 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.72709 The following new requirements on POSIX implementations derive from alignment with the  
72710 Single UNIX Specification:

- 72711
- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
72712 required for conforming implementations of previous POSIX specifications, it was not  
72713 required for UNIX applications.

72714  
72715

- In the DESCRIPTION, text previously conditional on support for `_POSIX_JOB_CONTROL` is now mandatory. This is a FIPS requirement.

72716 **NAME**

72717 tcgetsid — get the process group ID for the session leader for the controlling terminal

72718 **SYNOPSIS**

72719 #include <termios.h>

72720 pid\_t tcgetsid(int *fildev*);

72721 **DESCRIPTION**

72722 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal  
72723 specified by *fildev* is the controlling terminal.

72724 **RETURN VALUE**

72725 Upon successful completion, *tcgetsid()* shall return the process group ID of the session  
72726 associated with the terminal. Otherwise, a value of -1 shall be returned and *errno* set to indicate  
72727 the error.

72728 **ERRORS**

72729 The *tcgetsid()* function shall fail if:

72730 [EBADF] The *fildev* argument is not a valid file descriptor.

72731 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
72732 controlling terminal.

72733 **EXAMPLES**

72734 None.

72735 **APPLICATION USAGE**

72736 None.

72737 **RATIONALE**

72738 None.

72739 **FUTURE DIRECTIONS**

72740 None.

72741 **SEE ALSO**

72742 XBD <[termios.h](#)>

72743 **CHANGE HISTORY**

72744 First released in Issue 4, Version 2.

72745 **Issue 5**

72746 Moved from X/OPEN UNIX extension to BASE.

72747 The [EACCES] error has been removed from the list of mandatory errors, and the description of  
72748 [ENOTTY] has been reworded.

72749 **Issue 7**

72750 SD5-XSH-ERN-180 is applied, clarifying the RETURN VALUE section.

72751 The *tcgetsid()* function is moved from the XSI option to the Base.

72752 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0648 [421] is applied.

72753 **NAME**

72754 tcgetwinsize — get the size of a terminal window

72755 **SYNOPSIS**

72756 #include &lt;termios.h&gt;

72757 int tcgetwinsize(int *fildev*, struct winsize \**winsize\_p*);72758 **DESCRIPTION**

72759 The *tcgetwinsize()* function shall get the terminal window size associated with the terminal  
 72760 referred to by *fildev* and store it in the **winsize** structure pointed to by *winsize\_p*. The *fildev*  
 72761 argument is an open file descriptor associated with a terminal. The *winsize\_p* argument is a  
 72762 pointer to a **winsize** structure.

72763 If the terminal referred to by *fildev* was opened without O\_TTY\_INIT and is not a pseudo-  
 72764 terminal, and the terminal window size has not been set by a call to *tcsetwinsize()*, the terminal  
 72765 window size is unspecified.

72766 If the terminal was opened with O\_TTY\_INIT or is a pseudo-terminal, and the terminal window  
 72767 size has not been set by a call to *tcsetwinsize()*, the terminal window size shall be set to an  
 72768 appropriate default (see *open()*).

72769 If the terminal window size has been set by a call to *tcsetwinsize()*, the values set by that call  
 72770 shall be returned.

72771 **RETURN VALUE**

72772 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 72773 indicate the error.

72774 **ERRORS**72775 The *tcgetwinsize()* function shall fail if:72776 [EBADF] The *fildev* argument is not a valid file descriptor.72777 [ENOTTY] The file associated with *fildev* is not a terminal.72778 **EXAMPLES**

72779 #include &lt;stdint.h&gt;

72780 #include &lt;stdio.h&gt;

72781 #include &lt;stdlib.h&gt;

72782 #include &lt;errno.h&gt;

72783 #include &lt;signal.h&gt;

72784 #include &lt;termios.h&gt;

72785 #include &lt;unistd.h&gt;

72786 static volatile sig\_atomic\_t vrow;

72787 static volatile sig\_atomic\_t vcol;

72788 static volatile sig\_atomic\_t newsize;

72789 static void winch\_handler(int signum)

72790 {

72791 struct winsize ws;

72792 int sav\_errno = errno;

72793 (void)signum; // prevent compiler warning that signum is unused

72794 // set volatile vars to new winsize, or 0 if unavailable or

```
72795         // too large

72796         if (tcgetwinsize(STDERR_FILENO, &ws) == -1)
72797         {
72798             vrow = vcol = 0;
72799         }
72800         else
72801         {
72802             if (ws.ws_row <= SIG_ATOMIC_MAX && ws.ws_col <= SIG_ATOMIC_MAX)
72803             {
72804                 vrow = ws.ws_row;
72805                 vcol = ws.ws_col;
72806             }
72807             else
72808             {
72809                 vrow = vcol = 0;
72810             }
72811         }

72812         newsize = 1;

72813         errno = sav_errno;
72814     }

72815     int main(void)
72816     {
72817         struct sigaction sa;
72818         struct winsize ws;
72819         sigset_t winch_set;
72820         char inbuf[512];

72821         sa.sa_handler = winch_handler;
72822         sigemptyset(&sa.sa_mask);
72823         sa.sa_flags = 0;
72824         sigaction(SIGWINCH, &sa, NULL);

72825         sigemptyset(&winch_set);
72826         sigaddset(&winch_set, SIGWINCH);

72827         raise(SIGWINCH); // gets the initial winsize

72828         for (;;)
72829         {
72830             if (fgets(inbuf, sizeof inbuf, stdin) == NULL)
72831             {
72832                 if (feof(stdin))
72833                     exit(0);
72834                 else if (errno == EINTR)
72835                     continue;
72836                 else
72837                 {
72838                     perror("Error reading stdin");
```

```

72839         exit(1);
72840     }
72841 }
72842 else
72843 {
72844     if (newsize)
72845     {
72846         // prevent updates to volatile vars while we read them
72847         sigprocmask(SIG_BLOCK, &winch_set, NULL);
72848         ws.ws_row = vrow;
72849         ws.ws_col = vcol;
72850         sigprocmask(SIG_UNBLOCK, &winch_set, NULL);
72851         newsize = 0;
72852     }
72853     printf("row = %3hu, col = %3hu\n", ws.ws_row, ws.ws_col);
72854
72855     // process inbuf ...
72856 }
72857 }

```

#### 72858 APPLICATION USAGE

72859 Applications should take care to avoid race conditions and other undefined behavior when  
72860 calling `tcgetwinsize()` from signal handlers. A common but incorrect idiom is to establish a signal  
72861 handler for SIGWINCH from which `tcgetwinsize()` is called to update a global **winsize** structure.  
72862 This usage is incorrect as accessing a **winsize** structure is not guaranteed to be an atomic  
72863 operation. Instead, applications should have `tcgetwinsize()` write to a local structure and copy  
72864 each member the application is interested in to a global variable of type **volatile sig\_atomic\_t**.  
72865 Furthermore, SIGWINCH should be blocked from delivery while the terminal size is read from  
72866 these global variables to further avoid race conditions. A simpler alternative, if the application is  
72867 structured in a suitable way, is just to set a flag in the signal handler and then call `tcgetwinsize()`  
72868 (and clear the flag) at an appropriate place in the code if the flag has been set.

72869 Multi-threaded applications should avoid the signal handler idiom in general. Instead, it is  
72870 advised to use `sigwait()` to wait for the delivery of a SIGWINCH signal.

72871 If the terminal window size changes while a process is in the background, it is not notified via  
72872 SIGWINCH (which is sent only to the foreground process group). Applications can handle this  
72873 case by calling `tcgetwinsize()` if the process receives SIGCONT, to check whether the terminal  
72874 window size changed while the process was stopped.

72875 If a background process writes to a terminal and the TOSTOP flag is clear (see XBD [Section](#)  
72876 [11.2.5](#), on page 210), the process might not receive SIGTTOU or SIGWINCH signals and thus  
72877 might not be notified when the terminal window size might have changed. Such processes must  
72878 periodically poll the current terminal window size if needed.

#### 72879 RATIONALE

72880 The `tcgetwinsize()` function is provided to allow applications to query the current terminal  
72881 window size. This is necessary for applications intended to be run in terminals whose terminal  
72882 window size can be changed at runtime. Conventionally, a SIGWINCH signal is delivered to a  
72883 controlling terminal's foreground process group whenever its terminal window size is changed.  
72884 By installing a signal handler for SIGWINCH, a process can detect the change to the controlling  
72885 terminal's window size and take action, e.g. by redrawing its user interface to the new size.

72886 **FUTURE DIRECTIONS**

72887       None.

72888 **SEE ALSO**

72889       [tcsetwinsize\(\)](#)

72890       XBD [<termios.h>](#)

72891 **CHANGE HISTORY**

72892       First released in Issue 8.



72893 **NAME**

72894 tcsendbreak — send a break for a specific duration

72895 **SYNOPSIS**

72896 #include &lt;termios.h&gt;

72897 int tcsendbreak(int *fildev*, int *duration*);72898 **DESCRIPTION**

72899 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause  
72900 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it  
72901 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5  
72902 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period  
72903 of time.

72904 The *fildev* argument is an open file descriptor associated with a terminal.

72905 If the terminal is not using asynchronous serial data transmission, it is implementation-defined  
72906 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any  
72907 action.

72908 Attempts to use *tcsendbreak()* from a process which is a member of a background process group  
72909 on a *fildev* associated with its controlling terminal shall cause the process group to be sent a  
72910 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring  
72911 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

72912 **RETURN VALUE**

72913 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
72914 indicate the error.

72915 **ERRORS**

72916 The *tcsendbreak()* function shall fail if:

72917 [EBADF] The *fildev* argument is not a valid file descriptor.

72918 [EIO] The process group of the writing process is orphaned, the calling thread is not  
72919 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

72920 [ENOTTY] The file associated with *fildev* is not a terminal.

72921 **EXAMPLES**

72922 None.

72923 **APPLICATION USAGE**

72924 None.

72925 **RATIONALE**

72926 None.

72927 **FUTURE DIRECTIONS**

72928 None.

72929 **SEE ALSO**

72930 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)

72931 **CHANGE HISTORY**

72932 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

72933 **Issue 6**

72934 The following new requirements on POSIX implementations derive from alignment with the  
72935 Single UNIX Specification:

- 72936 • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now  
72937 mandated. This is a FIPS requirement.
- 72938 • The [EIO] error is added.

72939 **Issue 7**

72940 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0649 [79], XSH/TC1-2008/0650 [79],  
72941 and XSH/TC1-2008/0651 [79] are applied.

72942 **NAME**

72943 tcsetattr — set the parameters associated with the terminal

72944 **SYNOPSIS**

72945 #include &lt;termios.h&gt;

72946 int tcsetattr(int *fildev*, int *optional\_actions*,  
72947 const struct termios \**termios\_p*);72948 **DESCRIPTION**72949 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the  
72950 open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios**  
72951 structure referenced by *termios\_p* as follows:

- 72952 • If *optional\_actions* is TCSANOW, the change shall occur immediately.
- 72953 • If *optional\_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is  
72954 transmitted. This function should be used when changing parameters that affect output.
- 72955 • If *optional\_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is  
72956 transmitted, and all input so far received but not read shall be discarded before the change  
72957 is made.

72958 If the output baud rate stored in the **termios** structure pointed to by *termios\_p* is the zero baud  
72959 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the  
72960 line.72961 If the input baud rate stored in the **termios** structure pointed to by *termios\_p* is 0, the input baud  
72962 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.72963 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested  
72964 actions, even if some of the requested actions could not be performed. It shall set all the  
72965 attributes that the implementation supports as requested and leave all the attributes not  
72966 supported by the implementation unchanged. If no part of the request can be honored, it shall  
72967 return  $-1$  and set *errno* to [EINVAL]. If the input and output baud rates differ and are a  
72968 combination that is not supported, neither baud rate shall be changed. A subsequent call to  
72969 *tcsetattr()* shall return the actual state of the terminal device (reflecting both the changes made  
72970 and not made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values  
72971 found in the **termios** structure under any circumstances.72972 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios\_p*  
72973 was not derived from the result of a call to *tcgetattr()* on *fildev*; an application should modify  
72974 only fields and flags defined by this volume of POSIX.1-2024 between the call to *tcgetattr()* and  
72975 *tcsetattr()*, leaving all other fields and flags unmodified.72976 No actions defined by this volume of POSIX.1-2024, other than a call to *tcsetattr()*, a close of the  
72977 last file descriptor in the system associated with this terminal device, or an open of the first file  
72978 descriptor in the system associated with this terminal device (using the O\_TTY\_INIT flag if it is  
72979 non-zero and the device is not a pseudo-terminal), shall cause any of the terminal attributes  
72980 defined by this volume of POSIX.1-2024 to change.72981 If *tcsetattr()* is called from a process which is a member of a background process group on a *fildev*  
72982 associated with its controlling terminal:

- 72983 • If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU  
72984 signals, the operation completes normally and no signal is sent.

- 72985           • Otherwise, a SIGTTOU signal shall be sent to the process group.

#### 72986 RETURN VALUE

72987           Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
72988           indicate the error.

#### 72989 ERRORS

72990           The *tcsetattr()* function shall fail if:

- |       |          |   |
|-------|----------|---|
| 72991 | [EBADF]  | The <i>fildev</i> argument is not a valid file descriptor.  |
| 72992 | [EINTR]  | A signal interrupted <i>tcsetattr()</i> .   |
| 72993 | [EINVAL] | The <i>optional_actions</i> argument is not a supported value, or an attempt was<br>72994           made to change an attribute represented in the <b>termios</b> structure to an<br>72995           unsupported value. |
| 72996 | [EIO]    | The process group of the writing process is orphaned, the calling thread is not<br>72997           blocking SIGTTOU, and the process is not ignoring SIGTTOU.   |
| 72998 | [ENOTTY] | The file associated with <i>fildev</i> is not a terminal.   |

#### 72999 EXAMPLES

73000           None.

#### 73001 APPLICATION USAGE

73002           If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to  
73003           determine what baud rates were actually selected.

73004           In general, there are two reasons for an application to change the parameters associated with a  
73005           terminal device:

- |       |    |   |
|-------|----|---|
| 73006 | 1. | The device already has working parameter settings but the application needs a different<br>73007           behavior, such as non-canonical mode instead of canonical mode. The application sets (or<br>73008           clears) only a few flags or <i>c_cc[]</i> values. Typically, the terminal device in this case is either<br>73009           the controlling terminal for the process or a pseudo-terminal.  |
| 73010 | 2. | The device is a modem or similar piece of equipment connected by a serial line, and it<br>73011           was not open before the application opened it. In this case, the application needs to<br>73012           initialize all of the parameter settings “from scratch”. However, since the <b>termios</b><br>73013           structure may include both standard and non-standard parameters, the application<br>73014           cannot just initialize the whole structure in an arbitrary way (e.g., using <i>memset()</i> ) as this<br>73015           may cause some of the non-standard parameters to be set incorrectly, resulting in non-<br>73016           conforming behavior of the terminal device. Conversely, the application cannot just set<br>73017           the standard parameters, assuming that the non-standard parameters will already have<br>73018           suitable values, as the device might previously have been used with non-conforming<br>73019           parameter settings (and some implementations retain the settings from one use to the<br>73020           next). The solution is to open the terminal device using the <i>O_TTY_INIT</i> flag to initialize<br>73021           the terminal device to have conforming parameter settings, obtain those settings using<br>73022 <i>tcgetattr()</i> , and then set all of the standard parameters to the desired settings. |

#### 73023 RATIONALE

73024           The *tcsetattr()* function can be interrupted in the following situations:

- |       |   |  |
|-------|---|--|
| 73025 | • | It is interrupted while waiting for output to drain.                             |
| 73026 | • | It is called from a process in a background process group and SIGTTOU is caught. |

73027           See also the RATIONALE section in *tcgetattr()*.

73028 **FUTURE DIRECTIONS**

73029 Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be  
73030 supported in a future version of this volume of POSIX.1-2024.

73031 **SEE ALSO**

73032 *cfgetispeed()*, *tcgetattr()*

73033 XBD Chapter 11 (on page 199), `<termios.h>`

73034 **CHANGE HISTORY**

73035 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

73036 **Issue 6**

73037 The following new requirements on POSIX implementations derive from alignment with the  
73038 Single UNIX Specification:

- 73039 • In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now  
73040 mandated. This is a FIPS requirement.
- 73041 • The [EIO] error is added.

73042 In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of  
73043 a background process group is clarified.

73044 **Issue 7**

73045 Austin Group Interpretation 1003.1-2001 #144 is applied.

73046 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0652 [79], XSH/TC1-2008/0653 [79],  
73047 and XSH/TC1-2008/0654 [79] are applied.

73048 **NAME**

73049 tcsetpgrp — set the foreground process group ID

73050 **SYNOPSIS**

73051 #include &lt;unistd.h&gt;

73052 int tcsetpgrp(int *fildes*, pid\_t *pgid\_id*);73053 **DESCRIPTION**

73054 If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID  
 73055 associated with the terminal to *pgid\_id*. The application shall ensure that the file associated with  
 73056 *fildes* is the controlling terminal of the calling process and the controlling terminal is currently  
 73057 associated with the session of the calling process. The application shall ensure that the value of  
 73058 *pgid\_id* matches a process group ID of a process in the same session as the calling process.

73059 Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on  
 73060 a *fildes* associated with its controlling terminal shall cause the process group to be sent a  
 73061 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring  
 73062 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

73063 **RETURN VALUE**

73064 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
 73065 indicate the error.

73066 **ERRORS**73067 The *tcsetpgrp()* function shall fail if:73068 [EBADF] The *fildes* argument is not a valid file descriptor.73069 [EINTR] A signal interrupted *tcsetpgrp()*.73070 [EINVAL] This implementation does not support the value in the *pgid\_id* argument.

73071 [EIO] The process group of the writing process is orphaned, the calling thread is not  
 73072 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

73073 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the  
 73074 controlling terminal, or the controlling terminal is no longer associated with  
 73075 the session of the calling process.

73076 [EPERM] The value of *pgid\_id* is a value supported by the implementation, but does not  
 73077 match the process group ID of a process in the same session as the calling  
 73078 process.

73079 **EXAMPLES**

73080 None.

73081 **APPLICATION USAGE**

73082 None.

73083 **RATIONALE**

73084 None.

73085 **FUTURE DIRECTIONS**

73086 None.

73087 **SEE ALSO**73088 [tcgetpgrp\(\)](#)73089 XBD [<sys/types.h>](#), [<unistd.h>](#)

73090 **CHANGE HISTORY**

73091 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

73092 **Issue 6**

73093 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

73094 The following new requirements on POSIX implementations derive from alignment with the  
73095 Single UNIX Specification:

73096 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
73097 required for conforming implementations of previous POSIX specifications, it was not  
73098 required for UNIX applications.

73099 • In the DESCRIPTION and ERRORS sections, text previously conditional on  
73100 `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

73101 The normative text is updated to avoid use of the term “must” for application requirements.

73102 The Open Group Corrigendum U047/4 is applied.

73103 **Issue 7**

73104 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0655 [79] and XSH/TC1-2008/0656  
73105 [79] are applied.

73106 **Issue 8**

73107 Austin Group Defect 1126 is applied, adding the [EINTR] error.

73108 **NAME**

73109 tcsetwinsize — set the size of a terminal window

73110 **SYNOPSIS**

73111 #include &lt;termios.h&gt;

73112 int tcsetwinsize(int *fildev*, const struct winsize \**winsize\_p*);73113 **DESCRIPTION**

73114 The *tcsetwinsize()* function shall set the terminal window size associated with the terminal  
 73115 referred to by the open file descriptor *fildev* (an open file descriptor associated with a terminal)  
 73116 from the **winsize** structure referenced by *winsize\_p*. The change shall occur immediately.

73117 If the terminal size was changed successfully, a SIGWINCH shall be delivered to the foreground  
 73118 process group associated with the terminal. No signal shall be delivered if the terminal size was  
 73119 changed to the same value it had before the *tcsetwinsize()* call. A SIGWINCH may also be  
 73120 delivered to an implementation-defined set of other processes.

73121 The *tcsetwinsize()* function shall return successfully if it was able to update all members of the  
 73122 **winsize** structure associated with the terminal.

73123 It is unspecified whether changing the terminal window size causes any changes to the size of  
 73124 the terminal's font.

73125 The effect of *tcsetwinsize()* is undefined if the value of the **winsize** structure pointed to by  
 73126 *winsize\_p* was not derived from the result of a call to *tcgetwinsize()* on *fildev*; an application  
 73127 should modify only fields defined by this volume of POSIX.1-2024 between the call to  
 73128 *tcgetwinsize()* and *tcsetwinsize()*, leaving all other fields unmodified.

73129 No actions defined by this volume of POSIX.1-2024, other than a call to *tcsetwinsize()*, a close of  
 73130 the last file descriptor in the system associated with this terminal device, or an open of the first  
 73131 file descriptor in the system associated with this terminal device (using the O\_TTY\_INIT flag if it  
 73132 is non-zero and the device is not a pseudo-terminal), shall cause the terminal window size to  
 73133 change.

73134 If *tcsetwinsize()* is called from a process which is a member of a background process group on a  
 73135 *fildev* associated with its controlling terminal:

- 73136 • If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU  
 73137 signals, the operation completes normally and no signal is sent.
- 73138 • Otherwise, a SIGTTOU signal shall be sent to the process group.

73139 **RETURN VALUE**

73140 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the terminal  
 73141 window size shall not be changed, and *errno* shall be set to indicate the error.

73142 **ERRORS**

73143 The *tcsetwinsize()* function shall fail if:

- 73144 [EBADF] The *fildev* argument is not a valid file descriptor.
- 73145 [EIO] The process group of the writing process is orphaned, the calling thread is not  
 73146 blocking SIGTTOU, and the process is not ignoring SIGTTOU.
- 73147 [ENOTTY] The file associated with *fildev* is not a terminal.

73148 The *tcsetwinsize()* function may fail if:



73149 [EINVAL] An attempt was made to change an attribute represented in the **winsize**  
73150 structure to an unsupported value.

73151 **EXAMPLES**

73152 None.

73153 **APPLICATION USAGE**

73154 If the terminal window of a pseudo terminal is resized, the attached manager process should  
73155 invoke *tcsetwinsize()* to relay the new terminal window size to the foreground process group.

73156 If a process attached to the subsidiary device of a pseudo-terminal calls *tcsetwinsize()*, the  
73157 attached manager process should attempt to change the screen to reflect the new size.

73158 **RATIONALE**

73159 This standard does not mention the *ws\_xpixel* and *ws\_ypixel* fields that appear in the **winsize**  
73160 structure of some historical implementations. With current hardware, it is not obvious that the  
73161 **unsigned short** type used for these fields is sufficient and no uses of these fields in portable code  
73162 were found. However, since these and other fields may be included in the **winsize** structure, the  
73163 standard requires that applications use *tcgetwinsize()* to initialize any fields that may be  
73164 provided by an implementation before setting the *ws\_cols* and *ws\_rows* fields using *tcsetwinsize()*  
73165 to avoid unintentionally destroying data in other fields in this structure.

73166 **FUTURE DIRECTIONS**

73167 None.

73168 **SEE ALSO**

73169 [\*tcgetwinsize\(\)\*](#)

73170 XBD [<termios.h>](#)

73171 **CHANGE HISTORY**

73172 First released in Issue 8.

73173 **NAME**

73174 tdelete, tfind, tsearch, twalk — manage a binary search tree

73175 **SYNOPSIS**

```

73176 XSI #include <search.h>
73177 void *tdelete(const void *restrict key,
73178             posix_tnode **restrict rootp,
73179             int (*compar)(const void *, const void *));
73180 posix_tnode *tfind(const void *key,
73181                  posix_tnode *const *rootp,
73182                  int (*compar)(const void *, const void *));
73183 posix_tnode *tsearch(const void *key,
73184                    posix_tnode **rootp,
73185                    int (*compar)(const void *, const void *));
73186 void twalk(const posix_tnode *root,
73187           void (*action)(const posix_tnode *, VISIT, int));

```

73188 **DESCRIPTION**

73189 The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees.  
 73190 Comparisons are made with a user-supplied routine, the address of which is passed as the  
 73191 *compar* argument. This routine is called with two arguments, which are the pointers to the  
 73192 elements being compared. The application shall ensure that the user-supplied routine returns an  
 73193 integer less than, equal to, or greater than 0, according to whether the first argument is to be  
 73194 considered less than, equal to, or greater than the second argument. The comparison function  
 73195 need not compare every byte, so arbitrary data may be contained in the elements in addition to  
 73196 the values being compared.

73197 The *tsearch()* function shall build and access the tree. The *key* argument is a pointer to an element  
 73198 to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed  
 73199 to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key*  
 73200 shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a  
 73201 pointer to this node returned. Only pointers are copied, so the application shall ensure that the  
 73202 calling routine stores the data. The *rootp* argument points to a variable that points to the root  
 73203 node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree;  
 73204 in this case, the variable shall be set to point to the node which shall be at the root of the new  
 73205 tree.

73206 Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found.  
 73207 However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the  
 73208 same as for *tsearch()*.

73209 The *tdelete()* function shall delete a node from a binary search tree. The arguments are the same  
 73210 as for *tsearch()*. The variable pointed to by *rootp* shall be set to a pointer to the new root of the  
 73211 tree if the root of the tree was changed. If the deleted node was the root of the tree and had no  
 73212 children, the variable pointed to by *rootp* shall be set to a null pointer. The *tdelete()* function shall  
 73213 return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the  
 73214 deleted node was the root node, or a null pointer if the node is not found.

73215 If *tsearch()* adds an element to a tree, or *tdelete()* successfully deletes an element from a tree, the  
 73216 concurrent use of that tree in another thread, or use of pointers produced by a previous call to  
 73217 *tfind()* or *tsearch()*, produces undefined results.

73218 The *twalk()* function shall traverse a binary search tree. The *root* argument is a pointer to the root  
 73219 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below

73220 that node.) The argument *action* is the name of a routine to be invoked at each node. This routine  
 73221 is, in turn, called with three arguments. The first argument shall be the address of the node being  
 73222 visited. The structure pointed to by this argument is unspecified and shall not be modified by  
 73223 the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-  
 73224 element to access the element stored in the node. The second argument shall be a value from an  
 73225 enumeration data type:

```
73226 typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

73227 (defined in `<search.h>`), depending on whether this is the first, second, or third time that the  
 73228 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a  
 73229 leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

73230 If the calling function alters the pointer to the root, the result is undefined.

73231 If the functions pointed to by *action* or *compar* (for any of these binary search functions) change  
 73232 the tree, the results are undefined.

73233 These functions are thread-safe only as long as multiple threads do not access the same tree.

#### 73234 RETURN VALUE

73235 If the node is found, both *tsearch()* and *tfind()* shall return a pointer to it. If not, *tfind()* shall  
 73236 return a null pointer, and *tsearch()* shall return a pointer to the inserted item.

73237 A null pointer shall be returned by *tsearch()* if there is not enough space available to create a new  
 73238 node.

73239 A null pointer shall be returned by *tdelete()*, *tfind()*, and *tsearch()* if *rootp* is a null pointer on  
 73240 entry.

73241 The *tdelete()* function shall return a pointer to the parent of the deleted node, or an unspecified  
 73242 non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

73243 The *twalk()* function shall not return a value.

73244 In all cases where a pointer to a node is returned, the structure pointed to by the return value is  
 73245 unspecified and shall not be modified by the application, but it shall be possible to cast a  
 73246 pointer-to-node into a pointer-to-pointer-to-element to access the element stored in the node.

#### 73247 ERRORS

73248 No errors are defined.

#### 73249 EXAMPLES

73250 The following code reads in strings and stores structures containing a pointer to each string and  
 73251 a count of its length. It then walks the tree, printing out the stored strings and their lengths in  
 73252 alphabetical order.

```
73253 #include <limits.h>
73254 #include <search.h>
73255 #include <stdlib.h>
73256 #include <string.h>
73257 #include <stdio.h>

73258 struct element {          /* Pointers to these are stored in the tree. */
73259     int     count;
73260     char    string[];
73261 };

73262 posix_tnode *root = NULL;          /* This points to the root. */
```

```

73263     int main(void)
73264     {
73265         char    str[_POSIX2_LINE_MAX+1];
73266         int     length = 0;
73267         struct element *elementptr;
73268         posix_tnode *node;
73269         void    print_node(const posix_tnode *, VISIT, int);
73270         int     node_compare(const void *, const void *);

73271         while (fgets(str, sizeof(str), stdin)) {
73272             /* Set element. */
73273             length = strlen(str);
73274             if (str[length-1] == '\n')
73275                 str[--length] = '\0';
73276             elementptr = malloc(sizeof(struct element) + length + 1);
73277             strcpy(elementptr->string, str);
73278             elementptr->count = 1;
73279             /* Put element into the tree. */
73280             node = tsearch((void *)elementptr, &root, node_compare);
73281             if (node == NULL) {
73282                 fprintf(stderr,
73283                     "tsearch: Not enough space available\n");
73284                 exit(EXIT_FAILURE);
73285             }
73286             else if (*(struct element **)node != elementptr) {
73287                 /* A node containing the element already exists */
73288                 (*(struct element **)node)->count++;
73289                 free(elementptr);
73290             }
73291         }
73292         twalk(root, print_node);

73293         /* Delete all nodes in the tree */
73294         while (root != NULL) {
73295             elementptr = *(struct element **)root;
73296             printf("deleting node: string = %s, count = %d\n",
73297                 elementptr->string,
73298                 elementptr->count);
73299             tdelete((void *)elementptr, &root, node_compare);
73300             free(elementptr);
73301         }

73302         return 0;
73303     }

73304     /*
73305     * This routine compares two nodes, based on an
73306     * alphabetical ordering of the string field.
73307     */
73308     int
73309     node_compare(const void *node1, const void *node2)
73310     {
73311         return strcmp(((const struct element *) node1)->string,
73312             ((const struct element *) node2)->string);

```

```

73313     }
73314     /*
73315     *   This routine prints out a node, the second time
73316     *   twalk encounters it or if it is a leaf.
73317     */
73318     void
73319     print_node(const posix_tnode *ptr, VISIT order, int level)
73320     {
73321         const struct element *p = *(const struct element **) ptr;
73322
73323         if (order == postorder || order == leaf) {
73324             (void) printf("string = %s, count = %d\n",
73325                p->string, p->count);
73326         }
73327     }

```

### 73327 APPLICATION USAGE

73328 The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()*  
73329 and *tsearch()*.

73330 There are two nomenclatures used to refer to the order in which tree nodes are visited. The  
73331 *twalk()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node  
73332 before any of its children, after its left child and before its right, and after both its children. The  
73333 alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which  
73334 could result in some confusion over the meaning of **postorder**.

73335 Since the return value of *tdelete()* is an unspecified non-null pointer in the case that the root of  
73336 the tree has been deleted, applications should only use the return value of *tdelete()* as indication  
73337 of success or failure in this case and should not assume it can be dereferenced. However, the  
73338 only way that applications can tell if this case may have occurred is by checking whether the  
73339 variable pointed to by *rootp* changed. Since this variable can change for other reasons (for  
73340 example, tree balancing), using the return value of *tdelete()* as anything other than a boolean  
73341 indicator is unreliable at best and is discouraged. Some implementations in this case will return  
73342 a pointer to the new root of the tree (or to an empty tree if the deleted root node was the only  
73343 node in the tree); other implementations return arbitrary non-null pointers.

### 73344 RATIONALE

73345 Implementations are encouraged to use balanced trees to reduce the depth of the trees that are  
73346 created and improve tree search times.

### 73347 FUTURE DIRECTIONS

73348 None.

### 73349 SEE ALSO

73350 [hcreate\(\)](#), [lsearch\(\)](#)

73351 XBD [<search.h>](#)

### 73352 CHANGE HISTORY

73353 First released in Issue 1. Derived from Issue 1 of the SVID.

### 73354 Issue 5

73355 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in  
73356 previous issues.

73357 **Issue 6**

73358 The normative text is updated to avoid use of the term “must” for application requirements.

73359 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the  
73360 ISO/IEC 9899:1999 standard.

73361 **Issue 7**

73362 Austin Group Interpretation 1003.1-2001 #149 is applied, clarifying concurrent use of the tree in  
73363 another thread.

73364 Austin Group Interpretation 1003.1-2001 #151 is applied, clarifying behavior for *tdelete()* when  
73365 the deleted node is the root node.

73366 Austin Group Interpretation 1003.1-2001 #153 is applied.

73367 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0366 [551] is applied.

73368 **Issue 8**

73369 Austin Group Defect 1011 is applied, changing some prototypes to use **posix\_tnode** instead of  
73370 **void**, and changing the required behavior for *tdelete()* when the root of the tree changes.

73371 Austin Group Defect 1470 is applied, changing the EXAMPLES section.

73372 **NAME**

73373 telldir — current location of a named directory stream

73374 **SYNOPSIS**

```
73375 XSI #include <dirent.h>  
73376 long telldir(DIR *dirp);
```

73377 **DESCRIPTION**73378 The *telldir()* function shall obtain the current location associated with the directory stream  
73379 specified by *dirp*.73380 If the most recent operation on the directory stream was a *seekdir()*, the directory position  
73381 returned from the *telldir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.73382 **RETURN VALUE**73383 Upon successful completion, *telldir()* shall return the current location of the specified directory  
73384 stream.73385 **ERRORS**

73386 No errors are defined.

73387 **EXAMPLES**

73388 None.

73389 **APPLICATION USAGE**

73390 None.

73391 **RATIONALE**

73392 None.

73393 **FUTURE DIRECTIONS**

73394 None.

73395 **SEE ALSO**73396 *fdopendir()*, *readdir()*, *seekdir()*73397 XBD <[dirent.h](#)>73398 **CHANGE HISTORY**

73399 First released in Issue 2.

73400 **NAME**

73401 textdomain — text domain manipulation function

73402 **SYNOPSIS**

73403 #include <libintl.h>

73404 char \*textdomain(const char \*domainname);

73405 **DESCRIPTION**

73406 Refer to *bindtextdomain()*.



73407 **NAME**

73408 tfind — search binary search tree

73409 **SYNOPSIS**

```
73410 XSI #include <search.h>
73411 void *tfind(const void *key, void *const *rootp,
73412 int (*compar)(const void *, const void *));
```

73413 **DESCRIPTION**73414 Refer to [tdelete\(\)](#).

73415 **NAME**

73416 tgamma, tgammaf, tgammal — compute gamma() function

73417 **SYNOPSIS**

```
73418 #include <math.h>
73419 double tgamma(double x);
73420 float tgammaf(float x);
73421 long double tgammal(long double x);
```

73422 **DESCRIPTION**

73423 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 73424 conflict between the requirements described here and the ISO C standard is unintentional. This  
 73425 volume of POSIX.1-2024 defers to the ISO C standard.

73426 These functions shall compute  $\Gamma(x)$  where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t} t^{x-1} dt$ .

73427 An application wishing to check for error situations should set *errno* to zero and call  
 73428 *feclearexcept*(FE\_ALL\_EXCEPT) before calling these functions. On return, if *errno* is non-zero or  
 73429 *fetestexcept*(FE\_INVALID | FE\_DIVBYZERO | FE\_OVERFLOW | FE\_UNDERFLOW) is non-  
 73430 zero, an error has occurred.

73431 **RETURN VALUE**

73432 Upon successful completion, these functions shall return the gamma of *x*.

73433 If *x* is a negative integer, either a domain error or a pole error may occur and either a NaN (if  
 73434 supported) or  $\pm\text{Inf}$  (if supported), respectively, or an implementation-defined value shall be  
 73435 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall  
 73436 occur and a NaN shall be returned.

73437 If *x* is  $\pm 0$ , *tgamma()*, *tgammaf()*, and *tgammal()* shall return  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , and  
 73438 MX  $\pm\text{HUGE\_VALL}$ , respectively. On systems that support the IEC 60559 Floating-Point option, a  
 73439 CX pole error shall occur; otherwise, a pole error may occur.

73440 If the correct value would cause overflow, a range error shall occur and *tgamma()*, *tgammaf()*,  
 73441 and *tgammal()* shall return  $\pm\text{HUGE\_VAL}$ ,  $\pm\text{HUGE\_VALF}$ , or  $\pm\text{HUGE\_VALL}$ , respectively, with  
 73442 the same sign as the correct value of the function.

73443 MX If the correct value would cause underflow, and is not representable, a range error may occur,  
 73444 MX and *tgamma()*, *tgammaf()*, and *tgammal()* shall return 0.0, or (if IEC 60559 Floating-Point is not  
 73445 supported) an implementation-defined value no greater in magnitude than DBL\_MIN,  
 73446 FLT\_MIN, and LDBL\_MIN, respectively.

73447 MX If the correct value would cause underflow, and is representable, a range error may occur and  
 73448 the correct value shall be returned.

73449 If *x* is subnormal and  $1/x$  is representable,  $1/x$  should be returned.

73450 MX If *x* is NaN, a NaN shall be returned.

73451 If *x* is  $+\text{Inf}$ , *x* shall be returned.

73452 If *x* is  $-\text{Inf}$ , a domain error shall occur, and a NaN shall be returned.

73453 **ERRORS**

73454 These functions shall fail if:

73455	MX	<b>Domain Error</b>	The value of $x$ is a negative integer, or $x$ is $-\text{Inf}$ .
73456			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
73457			then <i>errno</i> shall be set to [EDOM]. If the integer expression ( <i>math_errhandling</i>
73458			& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
73459			shall be raised.
73460	MX	<b>Pole Error</b>	The value of $x$ is zero.
73461			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
73462			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
73463			( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
73464			floating-point exception shall be raised.
73465		<b>Range Error</b>	The value overflows.
73466			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
73467			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
73468			( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
73469			floating-point exception shall be raised.
73470			These functions may fail if:
73471		<b>Domain Error</b>	The value of $x$ is a negative integer.
73472			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
73473			then <i>errno</i> shall be set to [EDOM]. If the integer expression ( <i>math_errhandling</i>
73474			& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
73475			shall be raised.
73476		<b>Pole Error</b>	The value of $x$ is zero or a negative integer.
73477			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
73478			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
73479			( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
73480			floating-point exception shall be raised.
73481		<b>Range Error</b>	The result underflows.
73482			If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
73483			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
73484			( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
73485			floating-point exception shall be raised.

**73486 EXAMPLES**

73487 None.

**73488 APPLICATION USAGE**73489 On error, the expressions (*math\_errhandling* & MATH\_ERRNO) and (*math\_errhandling* &  
73490 MATH\_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.**73491 RATIONALE**73492 This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and  
73493 *lgamma()* functions.**73494 FUTURE DIRECTIONS**

73495 None.

73496 **SEE ALSO**73497 *feclearexcept()*, *fetestexcept()*, *lgamma()*

73498 XBD Section 4.23 (on page 109), &lt;math.h&gt;

73499 **CHANGE HISTORY**

73500 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

73501 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third  
73502 paragraph in the RETURN VALUE section.73503 **Issue 7**

73504 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.

73505 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0660 [68], XSH/TC1-2008/0661 [320],  
73506 and XSH/TC1-2008/0662 [68] are applied.73507 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0367 [604] and XSH/TC2-2008/0368  
73508 [630] are applied.73509 **Issue 8**73510 Austin Group Defect 1461 is applied, changing the requirements for a negative integer argument  
73511 to match the ISO C standard.

73512 **NAME**

73513 thrd\_create — thread creation

73514 **SYNOPSIS**

73515 #include &lt;threads.h&gt;

73516 int thrd\_create(thrd\_t \*thr, thrd\_start\_t func, void \*arg);

73517 **DESCRIPTION**

73518 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 73519 conflict between the requirements described here and the ISO C standard is unintentional. This  
 73520 volume of POSIX.1-2024 defers to the ISO C standard.

73521 The *thrd\_create()* function shall create a new thread executing *func(arg)*. If the *thrd\_create()*  
 73522 function succeeds, it shall set the object pointed to by *thr* to the identifier of the newly created  
 73523 thread. (A thread's identifier might be reused for a different thread once the original thread has  
 73524 exited and either been detached or joined to another thread.) The completion of the *thrd\_create()*  
 73525 function shall synchronize with the beginning of the execution of the new thread.

73526 CX The signal state of the new thread shall be initialized as follows:

- 73527 • The signal mask shall be inherited from the creating thread.
- 73528 • The set of signals pending for the new thread shall be empty.

73529 The thread-local current locale shall not be inherited from the creating thread.

73530 The floating-point environment shall be inherited from the creating thread.

73531 XSI The alternate stack shall not be inherited from the creating thread.

73532 Returning from *func* shall have the same behavior as invoking *thrd\_exit()* with the value  
 73533 returned from *func*.

73534 If *thrd\_create()* fails, no new thread shall be created and the contents of the location referenced  
 73535 by *thr* are undefined.

73536 CX The *thrd\_create()* function shall not be affected if the calling thread executes a signal handler  
 73537 during the call.

73538 **RETURN VALUE**

73539 The *thrd\_create()* function shall return *thrd\_success* on success, or *thrd\_nomem* if no  
 73540 memory could be allocated for the thread requested, or *thrd\_error* if the request could not be  
 73541 CX honored, such as if the system-imposed limit on the total number of threads in a process  
 73542 {PTHREAD\_THREADS\_MAX} would be exceeded.

73543 **ERRORS**

73544 See RETURN VALUE.

73545 **EXAMPLES**

73546 None.

73547 **APPLICATION USAGE**

73548 There is no requirement on the implementation that the ID of the created thread be available  
 73549 before the newly created thread starts executing. The calling thread can obtain the ID of the  
 73550 created thread through the *thr* argument of the *thrd\_create()* function, and the newly created  
 73551 thread can obtain its ID by a call to *thrd\_current()*.

73552 **RATIONALE**

73553 The *thrd\_create()* function is not affected by signal handlers for the reasons stated in XRAT  
73554 [Section B.2.3](#) (on page 3742).

73555 **FUTURE DIRECTIONS**

73556 None.

73557 **SEE ALSO**

73558 [pthread\\_create\(\)](#), [thrd\\_current\(\)](#), [thrd\\_detach\(\)](#), [thrd\\_exit\(\)](#), [thrd\\_join\(\)](#)

73559 XBD [Section 4.15.2](#) (on page 104), [<threads.h>](#)

73560 **CHANGE HISTORY**

73561 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

73562 **NAME**

73563           thrd\_current — get the calling thread ID

73564 **SYNOPSIS**

73565           #include &lt;threads.h&gt;

73566           thrd\_t thrd\_current(void);

73567 **DESCRIPTION**

73568 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
73569       conflict between the requirements described here and the ISO C standard is unintentional. This  
73570       volume of POSIX.1-2024 defers to the ISO C standard.

73571       The *thrd\_current()* function shall identify the thread that called it.73572 **RETURN VALUE**73573       The *thrd\_current()* function shall return the thread ID of the thread that called it.73574       The *thrd\_current()* function shall always be successful. No return value is reserved to indicate  
73575       an error.73576 **ERRORS**

73577       No errors are defined.

73578 **EXAMPLES**

73579       None.

73580 **APPLICATION USAGE**

73581       None.

73582 **RATIONALE**

73583       None.

73584 **FUTURE DIRECTIONS**

73585       None.

73586 **SEE ALSO**73587       

*pthread\_self()*, *thrd\_create()*, *thrd\_equal()*

73588       XBD &lt;threads.h&gt;

73589 **CHANGE HISTORY**

73590       First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

73591 **NAME**

73592 thrd\_detach — detach a thread

73593 **SYNOPSIS**

73594 #include &lt;threads.h&gt;

73595 int thrd\_detach(thrd\_t *thr*);73596 **DESCRIPTION**

73597 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
73598 conflict between the requirements described here and the ISO C standard is unintentional. This  
73599 volume of POSIX.1-2024 defers to the ISO C standard.

73600 The *thrd\_detach()* function shall change the thread *thr* from joinable to detached, indicating to  
73601 the implementation that any resources allocated to the thread can be reclaimed when that thread  
73602 terminates. The application shall ensure that the thread identified by *thr* has not been previously  
73603 detached or joined with another thread.

73604 CX The *thrd\_detach()* function shall not be affected if the calling thread executes a signal handler  
73605 during the call.

73606 **RETURN VALUE**

73607 The *thrd\_detach()* function shall return `thrd_success` on success or `thrd_error` if the request  
73608 could not be honored.

73609 **ERRORS**

73610 No errors are defined.

73611 **EXAMPLES**

73612 None.

73613 **APPLICATION USAGE**

73614 None.

73615 **RATIONALE**

73616 The *thrd\_detach()* function is not affected by signal handlers for the reasons stated in XRAT  
73617 [Section B.2.3](#) (on page 3742).

73618 **FUTURE DIRECTIONS**

73619 None.

73620 **SEE ALSO**73621 [pthread\\_detach\(\)](#), [thrd\\_create\(\)](#), [thrd\\_join\(\)](#)73622 XBD [<threads.h>](#)73623 **CHANGE HISTORY**

73624 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.



73625 **NAME**

73626 thrd\_equal — compare thread IDs

73627 **SYNOPSIS**

73628 #include &lt;threads.h&gt;

73629 int thrd\_equal(thrd\_t thr0, thrd\_t thr1);

73630 **DESCRIPTION**

73631 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
73632 conflict between the requirements described here and the ISO C standard is unintentional. This  
73633 volume of POSIX.1-2024 defers to the ISO C standard.

73634 The *thrd\_equal()* function shall determine whether the thread identified by *thr0* refers to the  
73635 thread identified by *thr1*.

73636 CX The *thrd\_equal()* function shall not be affected if the calling thread executes a signal handler  
73637 during the call.

73638 **RETURN VALUE**

73639 The *thrd\_equal()* function shall return a non-zero value if *thr0* and *thr1* are equal; otherwise, zero  
73640 shall be returned.

73641 CX If either *thr0* or *thr1* is not a valid thread ID and is not equal to PTHREAD\_NULL (which is  
73642 defined in <pthread.h>), the behavior is undefined.

73643 **ERRORS**

73644 No errors are defined.

73645 **EXAMPLES**

73646 None.

73647 **APPLICATION USAGE**

73648 None.

73649 **RATIONALE**73650 See the RATIONALE section for *pthread\_equal()*.

73651 The *thrd\_equal()* function is not affected by signal handlers for the reasons stated in XRAT  
73652 Section B.2.3 (on page 3742).

73653 **FUTURE DIRECTIONS**

73654 None.

73655 **SEE ALSO**73656 *pthread\_equal()*, *thrd\_current()*

73657 XBD &lt;pthread.h&gt;, &lt;threads.h&gt;

73658 **CHANGE HISTORY**

73659 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

73660 **NAME**

73661 thrd\_exit — thread termination

73662 **SYNOPSIS**

73663 #include &lt;threads.h&gt;

73664 \_Noreturn void thrd\_exit(int res);

73665 **DESCRIPTION**

73666 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
73667 conflict between the requirements described here and the ISO C standard is unintentional. This  
73668 volume of POSIX.1-2024 defers to the ISO C standard.

73669 CX For every thread-specific storage key (regardless of whether it has type `tss_t` or `pthread_key_t`)  
73670 which was created with a non-null destructor and for which the value is non-null, `thrd_exit()`  
73671 shall set the value associated with the key to a null pointer value and then invoke the destructor  
73672 with its previous value. The order in which destructors are invoked is unspecified.

73673 If after this process there remain keys with both non-null destructors and values, the  
73674 CX implementation shall repeat this process up to `{PTHREAD_DESTRUCTOR_ITERATIONS}`  
73675 times.

73676 Following this, the `thrd_exit()` function shall terminate execution of the calling thread and shall  
73677 CX set its exit status to `res`. Thread termination shall not release any application visible process  
73678 resources, including, but not limited to, mutexes and file descriptors, nor shall it perform any  
73679 process-level cleanup actions, including, but not limited to, calling any `atexit()` routines that  
73680 might exist.

73681 An implicit call to `thrd_exit()` is made when a thread that was created using `thrd_create()` returns  
73682 from the start routine that was used to create it (see `thrd_create()`).

73683 CX The behavior of `thrd_exit()` is undefined if called from a destructor function that was invoked as  
73684 a result of either an implicit or explicit call to `thrd_exit()`.

73685 The process shall exit with an exit status of zero after the last thread has been terminated. The  
73686 behavior shall be as if the implementation called `exit()` with a zero argument at thread  
73687 termination time.

73688 **RETURN VALUE**

73689 This function shall not return a value.

73690 **ERRORS**

73691 No errors are defined.

73692 **EXAMPLES**

73693 None.

73694 **APPLICATION USAGE**

73695 Calls to `thrd_exit()` should not be made from threads created using `pthread_create()` or via a  
73696 SIGEV\_THREAD notification, as their exit status has a different type (`void *` instead of `int`). If  
73697 `thrd_exit()` is called from the initial thread and it is not the last thread to terminate, other threads  
73698 should not try to obtain its exit status using `pthread_join()`.

73699 **RATIONALE**

73700 The normal mechanism by which a thread that was started using `thrd_create()` terminates is to  
73701 return from the function that was specified in the `thrd_create()` call that started it. The `thrd_exit()`  
73702 function provides the capability for such a thread to terminate without requiring a return from  
73703 the start routine of that thread, thereby providing a function analogous to `exit()`.

73704 Regardless of the method of thread termination, the destructors for any existing thread-specific

73705 data are executed.

73706 **FUTURE DIRECTIONS**

73707 None.

73708 **SEE ALSO**

73709 *exit()*, *pthread\_create()*, *thrd\_join()*

73710 XBD <**threads.h**>

73711 **CHANGE HISTORY**

73712 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

73713 **NAME**

73714 thrd\_join — wait for thread termination

73715 **SYNOPSIS**

73716 #include &lt;threads.h&gt;

73717 int thrd\_join(thrd\_t *thr*, int \**res*);73718 **DESCRIPTION**

73719 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 73720 conflict between the requirements described here and the ISO C standard is unintentional. This  
 73721 volume of POSIX.1-2024 defers to the ISO C standard.

73722 The *thrd\_join()* function shall join the thread identified by *thr* with the current thread by  
 73723 blocking until the other thread has terminated. If the parameter *res* is not a null pointer,  
 73724 *thrd\_join()* shall store the thread's exit status in the integer pointed to by *res*. The termination of  
 73725 the other thread shall synchronize with the completion of the *thrd\_join()* function. The  
 73726 application shall ensure that the thread identified by *thr* has not been previously detached or  
 73727 joined with another thread.

73728 The results of multiple simultaneous calls to *thrd\_join()* specifying the same target thread are  
 73729 undefined.

73730 The behavior is undefined if the value specified by the *thr* argument to *thrd\_join()* refers to the  
 73731 calling thread.

73732 CX It is unspecified whether a zombie thread counts against {PTHREAD\_THREADS\_MAX}.

73733 If *thr* refers to a thread that was created using *pthread\_create()* or via a SIGEV\_THREAD  
 73734 notification and the thread terminates, or has already terminated, by returning from its start  
 73735 routine, the behavior of *thrd\_join()* is undefined. If *thr* refers to a thread that terminates, or has  
 73736 already terminated, by calling *pthread\_exit()* or by being cancelled, the behavior of *thrd\_join()* is  
 73737 undefined.

73738 The *thrd\_join()* function shall not be affected if the calling thread executes a signal handler  
 73739 during the call.

73740 **RETURN VALUE**

73741 The *thrd\_join()* function shall return *thrd\_success* on success or *thrd\_error* if the request  
 73742 could not be honored.

73743 CX It is implementation-defined whether *thrd\_join()* detects deadlock situations; if it does detect  
 73744 them, it shall return *thrd\_error* when one is detected.

73745 **ERRORS**

73746 See RETURN VALUE.

73747 **EXAMPLES**

73748 None.

73749 **APPLICATION USAGE**

73750 None.

73751 **RATIONALE**

73752 The *thrd\_join()* function provides a simple mechanism allowing an application to wait for a  
 73753 thread to terminate. After the thread terminates, the application may then choose to clean up  
 73754 resources that were used by the thread. For instance, after *thrd\_join()* returns, any application-  
 73755 provided stack storage could be reclaimed.

73756 The *thrd\_join()* or *thrd\_detach()* function should eventually be called for every thread that is

- 73757 created using *thrd\_create()* so that storage associated with the thread may be reclaimed.
- 73758 The *thrd\_join()* function cannot be used to obtain the exit status of a thread that was created  
73759 using *pthread\_create()* or via a SIGEV\_THREAD notification and which terminates by returning  
73760 from its start routine, or of a thread that terminates by calling *pthread\_exit()*, because such  
73761 threads have a **void \*** exit status, instead of the **int** that *thrd\_join()* returns via its *res* argument.
- 73762 The *thrd\_join()* function cannot be used to obtain the exit status of a thread that terminates by  
73763 being cancelled because it has no way to indicate that a thread was cancelled. (The *pthread\_join()*  
73764 function does this by returning a reserved **void \*** exit status; it is not possible to reserve an **int**  
73765 value for this purpose without introducing a conflict with the ISO C standard.) The standard  
73766 developers considered adding a *thrd\_canceled* enumeration constant that *thrd\_join()* would  
73767 return in this case. However, this return would be unexpected in code that is written to conform  
73768 to the ISO C standard, and it would also not solve the problem that threads which use only ISO  
73769 C **<threads.h>** interfaces (such as ones created by third party libraries written to conform to the  
73770 ISO C standard) have no way to handle being cancelled, as the ISO C standard does not provide  
73771 cancellation cleanup handlers.
- 73772 The *thrd\_join()* function is not affected by signal handlers for the reasons stated in XRAT [Section](#)  
73773 [B.2.3](#) (on page 3742).
- 73774 **FUTURE DIRECTIONS**
- 73775 None.
- 73776 **SEE ALSO**
- 73777 [pthread\\_create\(\)](#), [pthread\\_exit\(\)](#), [pthread\\_join\(\)](#), [thrd\\_create\(\)](#), [thrd\\_exit\(\)](#)
- 73778 [XBD Section 4.15.2](#) (on page 104), **<threads.h>**
- 73779 **CHANGE HISTORY**
- 73780 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

73781 **NAME**

73782 thrd\_sleep — suspend execution for an interval

73783 **SYNOPSIS**

73784 #include &lt;threads.h&gt;

73785 int thrd\_sleep(const struct timespec \*duration,  
73786 struct timespec \*remaining);73787 **DESCRIPTION**73788 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
73789 conflict between the requirements described here and the ISO C standard is unintentional. This  
73790 volume of POSIX.1-2024 defers to the ISO C standard.73791 The *thrd\_sleep()* function shall suspend execution of the calling thread until either the interval  
73792 specified by *duration* has elapsed or a signal is delivered to the calling thread whose action is to  
73793 invoke a signal-catching function or to terminate the process. If interrupted by a signal and the  
73794 *remaining* argument is not null, the amount of time remaining (the requested interval minus the  
73795 time actually slept) shall be stored in the interval it points to. The *duration* and *remaining*  
73796 arguments can point to the same object.73797 The suspension time may be longer than requested because the interval is rounded up to an  
73798 integer multiple of the sleep resolution or because of the scheduling of other activity by the  
73799 system. But, except for the case of being interrupted by a signal, the suspension time shall not be  
73800 less than that specified, as measured by the system clock TIME\_UTC.73801 **RETURN VALUE**73802 The *thrd\_sleep()* function shall return zero if the requested time has elapsed,  $-1$  if it has been  
73803 interrupted by a signal, or a negative value (which may also be  $-1$ ) if it fails for any other reason.73804 CX If it returns a negative value, it shall set *errno* to indicate the error.73805 **ERRORS**73806 CX The *thrd\_sleep()* function shall fail if:73807 [EINTR] The *thrd\_sleep()* function was interrupted by a signal.73808 [EINVAL] The *duration* argument specified a nanosecond value less than zero or greater  
73809 than or equal to 1 000 million.73810 **EXAMPLES**

73811 None.

73812 **APPLICATION USAGE**73813 Since the return value may be  $-1$  for errors other than [EINTR], applications should examine  
73814 *errno* to distinguish [EINTR] from other errors (and thus determine whether the unslept time is  
73815 available in the interval pointed to by *remaining*).73816 **RATIONALE**73817 The *thrd\_sleep()* function is identical to the *nanosleep()* function except that the return value may  
73818 be any negative value when it fails with an error other than [EINTR].73819 **FUTURE DIRECTIONS**

73820 None.

73821 **SEE ALSO**73822 [nanosleep\(\)](#)73823 XBD [<threads.h>](#), [<time.h>](#)

73824 **CHANGE HISTORY**

73825 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

73826 **NAME**

73827 thrd\_yield — yield the processor

73828 **SYNOPSIS**

73829 #include &lt;threads.h&gt;

73830 void thrd\_yield(void);

73831 **DESCRIPTION**

73832 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
73833 conflict between the requirements described here and the ISO C standard is unintentional. This  
73834 volume of POSIX.1-2024 defers to the ISO C standard.

73835 CX The *thrd\_yield()* function shall force the running thread to relinquish the processor until it again  
73836 becomes the head of its thread list.

73837 **RETURN VALUE**

73838 This function shall not return a value.

73839 **ERRORS**

73840 No errors are defined.

73841 **EXAMPLES**

73842 None.

73843 **APPLICATION USAGE**73844 See the APPLICATION USAGE section for *sched\_yield()*.73845 **RATIONALE**

73846 The *thrd\_yield()* function is identical to the *sched\_yield()* function except that it does not return a  
73847 value.

73848 **FUTURE DIRECTIONS**

73849 None.

73850 **SEE ALSO**73851 *sched\_yield()*

73852 XBD &lt;threads.h&gt;

73853 **CHANGE HISTORY**

73854 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.



73855 **NAME**

73856           time — get time

73857 **SYNOPSIS**

```
73858       #include <time.h>
73859       time_t time(time_t *tloc);
```

73860 **DESCRIPTION**

73861 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
73862       conflict between the requirements described here and the ISO C standard is unintentional. This  
73863       volume of POSIX.1-2024 defers to the ISO C standard.

73864 CX       The *time()* function shall return the value of time in seconds since the Epoch.

73865       The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer,  
73866       no value is stored.

73867 **RETURN VALUE**

73868       Upon successful completion, *time()* shall return the value of time. Otherwise, **(time\_t)−1** shall be  
73869       returned.

73870 **ERRORS**

73871       The *time()* function may fail if:

73872 CX       [**EOverflow**] The number of seconds since the Epoch will not fit in an object of type **time\_t**.

73873 **EXAMPLES**73874           **Getting the Current Time**

73875       The following example uses the *time()* function to calculate the time elapsed, in seconds, since  
73876       the Epoch, *localtime()* to convert that value to a broken-down time, and *asctime()* to convert the  
73877       broken-down time values into a printable string.

```
73878       #include <stdio.h>
73879       #include <time.h>
73880       int main(void)
73881       {
73882       time_t result;
73883           result = time(NULL);
73884           printf("%s%ju secs since the Epoch\n",
73885                 asctime(localtime(&result)),
73886                 (uintmax_t)result);
73887           return(0);
73888       }
```

73889       This example writes the current time to *stdout* in a form like this:

```
73890       Wed Jun 26 10:32:15 1996
73891       835810335 secs since the Epoch
```

73892 **Timing an Event**

73893 The following example gets the current time, prints it out in the user's format, and prints the  
73894 number of minutes to an event being timed.

```
73895 #include <time.h>
73896 #include <stdio.h>
73897 ...
73898 time_t now;
73899 int minutes_to_event;
73900 ...
73901 time(&now);
73902 minutes_to_event = ...;
73903 printf("The time is ");
73904 puts(asctime(localtime(&now)));
73905 printf("There are %d minutes to the event.\n",
73906        minutes_to_event);
73907 ...
```

73908 **APPLICATION USAGE**

73909 None.

73910 **RATIONALE**

73911 The *time()* function returns a value in seconds while *clock\_gettime()* returns a **struct timespec**  
73912 (seconds and nanoseconds) and is therefore capable of returning more precise times. The *times()*  
73913 function is also capable of more precision than *time()* as it returns a value in clock ticks,  
73914 although it returns the elapsed time since an arbitrary point such as system boot time, not since  
73915 the epoch.

73916 Earlier versions of this standard allowed the width of **time\_t** to be less than 64 bits. A 32-bit  
73917 signed integer (as used in many historical implementations) fails in the year 2038, and although  
73918 a 32-bit unsigned integer does not fail until 2106 the preferred solution is to make **time\_t** wider  
73919 rather than to make it unsigned.

73920 On some systems the *time()* function is implemented using a system call that does not return an  
73921 error condition in addition to the return value. On these systems it is impossible to differentiate  
73922 between valid and invalid return values and hence overflow conditions cannot be reliably  
73923 detected.

73924 The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C  
73925 standard.

73926 Many historical implementations (including Version 7) and the 1984 /usr/group standard use  
73927 **long** instead of **time\_t**. This volume of POSIX.1-2024 uses the latter type in order to agree with  
73928 the ISO C standard.

73929 **FUTURE DIRECTIONS**

73930 None.

73931 **SEE ALSO**

73932 *asctime()*, *clock()*, *clock\_getres()*, *ctime()*, *difftime()*, *futimens()*, *gmtime()*, *localtime()*, *mktime()*,  
73933 *strftime()*, *strptime()*, *times()*

73934 XBD **<time.h>**

73935 **CHANGE HISTORY**

73936 First released in Issue 1. Derived from Issue 1 of the SVID.

73937 **Issue 6**

73938 Extensions beyond the ISO C standard are marked.

73939 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

73940 **Issue 7**

73941 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0663 [106], XSH/TC1-2008/0664 [350],  
73942 XSH/TC1-2008/0665 [106], XSH/TC1-2008/0666 [350], and XSH/TC1-2008/0667 [350] are  
73943 applied.

73944 **Issue 8**

73945 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

73946 Austin Group Defect 1462 is applied, changing the RATIONALE and FUTURE DIRECTIONS  
73947 sections.

73948 **NAME**

73949 timer\_create — create a per-process timer

73950 **SYNOPSIS**

```
73951 CX #include <signal.h>
73952 #include <time.h>
73953 int timer_create(clockid_t clockid, struct sigevent *restrict evp,
73954 timer_t *restrict timerid);
```

73955 **DESCRIPTION**

73956 The *timer\_create()* function shall create a per-process timer using the specified clock, *clock\_id*, as  
 73957 the timing base. The *timer\_create()* function shall return, in the location referenced by *timerid*, a  
 73958 timer ID of type **timer\_t** used to identify the timer in timer requests. This timer ID shall be  
 73959 unique within the calling process until the timer is deleted. The particular clock, *clock\_id*, is  
 73960 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return  
 73961 from *timer\_create()*.

73962 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the  
 73963 application, defines the asynchronous notification to occur as specified in [Section 2.4.1](#) (on page  
 73964 513) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument  
 73965 pointed to a **sigevent** structure with the *sigev\_notify* member having the value SIGEV\_SIGNAL,  
 73966 the *sigev\_signo* having a default signal number, and the *sigev\_value* member having the value of  
 73967 the timer ID.

73968 Each implementation shall define a set of clocks that can be used as timing bases for per-process  
 73969 timers. All implementations shall support CLOCK\_REALTIME and CLOCK\_MONOTONIC as  
 73970 values for *clock\_id*.

73971 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed  
 73972 and deleted by an *exec*.

73973 CPT If \_POSIX\_CPUTIME is defined, implementations shall support *clock\_id* values representing the  
 73974 CPU-time clock of the calling process.

73975 TCT If \_POSIX\_THREAD\_CPUTIME is defined, implementations shall support *clock\_id* values  
 73976 representing the CPU-time clock of the calling thread.

73977 CPT|TCT It is implementation-defined whether a *timer\_create()* function will succeed if the value defined  
 73978 by *clock\_id* corresponds to the CPU-time clock of a process or thread different from the process  
 73979 or thread invoking the function.

73980 TSA If *evp->sigev\_notify* is SIGEV\_THREAD and *sev->sigev\_notify\_attributes* is not NULL, if the  
 73981 attribute pointed to by *sev->sigev\_notify\_attributes* has a thread stack address specified by a call  
 73982 to *pthread\_attr\_setstack()*, the results are unspecified if the signal is generated more than once.

73983 **RETURN VALUE**

73984 If the call succeeds, *timer\_create()* shall return zero and update the location referenced by *timerid*  
 73985 to a **timer\_t**, which can be passed to the per-process timer calls. If an error occurs, the function  
 73986 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if  
 73987 an error occurs.

73988 **ERRORS**73989 The *timer\_create()* function shall fail if:

73990 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

73991	[EAGAIN]	The calling process has already created all of the timers it is allowed by this implementation.
73992		
73993	[EINVAL]	The specified clock ID is not defined.
73994	CPT TCT [ENOTSUP]	The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by <i>clock_id</i> and associated with a process or thread different from the process or thread invoking <i>timer_create()</i> .
73995		
73996		

#### 73997 EXAMPLES

73998 None.

#### 73999 APPLICATION USAGE

74000 If a timer is created which has *evp->sigev\_sigev\_notify* set to SIGEV\_THREAD and the attribute pointed to by *evp->sigev\_notify\_attributes* has a thread stack address specified by a call to *pthread\_attr\_setstack()*, the memory dedicated as a thread stack cannot be recovered. The reason for this is that the threads created in response to a timer expiration are created detached, or in an unspecified way if the thread attribute's *detachstate* is PTHREAD\_CREATE\_JOINABLE. In neither case is it valid to call *pthread\_join()*, which makes it impossible to determine the lifetime of the created thread which thus means the stack memory cannot be reused.

#### 74007 RATIONALE

##### 74008 Periodic Timer Overrun and Resource Allocation

74009 The specified timer facilities may deliver realtime signals (that is, queued signals) on implementations that support this option. Since realtime applications cannot afford to lose notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it must be possible to ensure that sufficient resources exist to deliver the signal when the event occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a request and a subsequent signal generation. If the request cannot allocate the signal delivery resources, it can fail the call with an [EAGAIN] error.

74016 Periodic timers are a special case. A single request can generate an unspecified number of signals. This is not a problem if the requesting process can service the signals as fast as they are generated, thus making the signal delivery resources available for delivery of subsequent periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the currently pending signal has been delivered.

74022 Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a pending signal are generated, it is implementation-defined whether a signal is delivered for each occurrence. This is not adequate for some realtime applications. So a mechanism is required to allow applications to detect how many timer expirations were delayed without requiring an indefinite amount of system resources to store the delayed expirations.

74027 The specified facilities provide for an overrun count. The overrun count is defined as the number of extra timer expirations that occurred between the time a timer expiration signal is generated and the time the signal is delivered. The signal-catching function, if it is concerned with overruns, can retrieve this count on entry. With this method, a periodic timer only needs one “signal queuing resource” that can be allocated at the time of the *timer\_create()* function call.

74032 A function is defined to retrieve the overrun count so that an application need not allocate static storage to contain the count, and an implementation need not update this storage asynchronously on timer expirations. But, for some high-frequency periodic applications, the overhead of an additional system call on each timer expiration may be prohibitive. The functions, as defined, permit an implementation to maintain the overrun count in user space,

74037 associated with the *timerid*. The *timer\_getoverrun()* function can then be implemented as a macro  
 74038 that uses the *timerid* argument (which may just be a pointer to a user space structure containing  
 74039 the counter) to locate the overrun count with no system call overhead. Other implementations,  
 74040 less concerned with this class of applications, can avoid the asynchronous update of user space  
 74041 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

#### 74042 **Timer Expiration Signal Parameters**

74043 The realtime signals functionality supports an application-specific datum that is delivered to the  
 74044 extended signal handler. This value is explicitly specified by the application, along with the  
 74045 signal number to be delivered, in a **sigevent** structure. The type of the application-defined value  
 74046 can be either an integer constant or a pointer. This explicit specification of the value, as opposed  
 74047 to always sending the timer ID, was selected based on existing practice.

74048 It is common practice for realtime applications (on non-POSIX systems or realtime extended  
 74049 POSIX systems) to use the parameters of event handlers as the case label of a switch statement or  
 74050 as a pointer to an application-defined data structure. Since *timer\_ids* are dynamically allocated  
 74051 by the *timer\_create()* function, they can be used for neither of these functions without additional  
 74052 application overhead in the signal handler; for example, to search an array of saved timer IDs to  
 74053 associate the ID with a constant or application data structure.

#### 74054 **FUTURE DIRECTIONS**

74055 None.

#### 74056 **SEE ALSO**

74057 [clock\\_getres\(\)](#), [timer\\_delete\(\)](#), [timer\\_getoverrun\(\)](#)

74058 XBD [<signal.h>](#), [<time.h>](#)

#### 74059 **CHANGE HISTORY**

74060 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

##### 74061 **Issue 6**

74062 The *timer\_create()* function is marked as part of the Timers option.

74063 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
 74064 implementation does not support the Timers option.

74065 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

74066 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the  
 74067 requirement for the CLOCK\_MONOTONIC clock under the Monotonic Clock option.

74068 The **restrict** keyword is added to the *timer\_create()* prototype for alignment with the  
 74069 ISO/IEC 9899:1999 standard.

74070 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/138 is applied, updating the  
 74071 DESCRIPTION and APPLICATION USAGE sections to describe the case when a timer is created  
 74072 with the notification method set to SIGEV\_THREAD.

##### 74073 **Issue 7**

74074 The *timer\_create()* function is moved from the Timers option to the Base.

##### 74075 **Issue 8**

74076 Austin Group Defect 1116 is applied, removing a reference to the Realtime Signals Extension  
 74077 option that existed in earlier versions of this standard.

74078 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

74079 **NAME**

74080 timer\_delete — delete a per-process timer

74081 **SYNOPSIS**

```
74082 CX #include <time.h>  
74083 int timer_delete(timer_t timerid);
```

74084 **DESCRIPTION**

74085 The *timer\_delete()* function deletes the specified timer, *timerid*, previously created by the  
74086 *timer\_create()* function. If the timer is armed when *timer\_delete()* is called, the behavior shall be  
74087 as if the timer is automatically disarmed before removal. The disposition of pending signals for  
74088 the deleted timer is unspecified.

74089 The behavior is undefined if the value specified by the *timerid* argument to *timer\_delete()* does  
74090 not correspond to a timer ID returned by *timer\_create()* but not yet deleted by *timer\_delete()*.

74091 **RETURN VALUE**

74092 If successful, the *timer\_delete()* function shall return a value of zero. Otherwise, the function shall  
74093 return a value of -1 and set *errno* to indicate the error.

74094 **ERRORS**

74095 No errors are defined.

74096 **EXAMPLES**

74097 None.

74098 **APPLICATION USAGE**

74099 None.

74100 **RATIONALE**

74101 If an implementation detects that the value specified by the *timerid* argument to *timer\_delete()*  
74102 does not correspond to a timer ID returned by *timer\_create()* but not yet deleted by  
74103 *timer\_delete()*, it is recommended that the function should fail and report an [EINVAL] error.

74104 **FUTURE DIRECTIONS**

74105 None.

74106 **SEE ALSO**74107 [timer\\_create\(\)](#)74108 XBD [<time.h>](#)74109 **CHANGE HISTORY**

74110 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

74111 **Issue 6**74112 The *timer\_delete()* function is marked as part of the Timers option.

74113 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
74114 implementation does not support the Timers option.

74115 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/139 is applied, updating the ERRORS  
74116 section so that the [EINVAL] error becomes optional.

74117 **Issue 7**74118 The *timer\_delete()* function is moved from the Timers option to the Base.

74119 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0369 [659] is applied.

74120 **NAME**

74121 timer\_getoverrun, timer\_gettime, timer\_settime — per-process timers

74122 **SYNOPSIS**

```
74123 CX #include <time.h>
74124 int timer_getoverrun(timer_t timerid);
74125 int timer_gettime(timer_t timerid, struct itimerspec *value);
74126 int timer_settime(timer_t timerid, int flags,
74127     const struct itimerspec *restrict value,
74128     struct itimerspec *restrict ovalue);
```

74129 **DESCRIPTION**

74130 The *timer\_gettime()* function shall store the amount of time until the specified timer, *timerid*,  
 74131 expires and the reload value of the timer into the space pointed to by the *value* argument. The  
 74132 *it\_value* member of this structure shall contain the amount of time before the timer expires, or  
 74133 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if  
 74134 the timer was armed with absolute time. The *it\_interval* member of *value* shall contain the reload  
 74135 value last set by *timer\_settime()*.

74136 The *timer\_settime()* function shall set the time until the next expiration of the timer specified by  
 74137 *timerid* from the *it\_value* member of the *value* argument and arm the timer if the *it\_value* member  
 74138 of *value* is non-zero. If the specified timer was already armed when *timer\_settime()* is called, this  
 74139 call shall reset the time until next expiration to the *value* specified. If the *it\_value* member of *value*  
 74140 is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending  
 74141 expiration notifications is unspecified.

74142 If the flag `TIMER_ABSTIME` is not set in the argument *flags*, *timer\_settime()* shall behave as if the  
 74143 time until next expiration is set to be equal to the interval specified by the *it\_value* member of  
 74144 *value*. That is, the timer shall expire in *it\_value* nanoseconds from when the call is made. If the  
 74145 flag `TIMER_ABSTIME` is set in the argument *flags*, *timer\_settime()* shall behave as if the time  
 74146 until next expiration is set to be equal to the difference between the absolute time specified by  
 74147 the *it\_value* member of *value* and the current value of the clock associated with *timerid*. That is,  
 74148 the timer shall expire when the clock reaches the value specified by the *it\_value* member of *value*.  
 74149 If the specified time has already passed, the function shall succeed and the expiration  
 74150 notification shall be made.

74151 The reload value of the timer shall be set to the value specified by the *it\_interval* member of  
 74152 *value*. When a timer is armed with a non-zero *it\_interval*, a periodic (or repetitive) timer is  
 74153 specified.

74154 Time values that are between two consecutive non-negative integer multiples of the resolution of  
 74155 the specified timer shall be rounded up to the larger multiple of the resolution. Quantization  
 74156 error shall not cause the timer to expire earlier than the rounded time value.

74157 If the argument *ovalue* is not `NULL`, the *timer\_settime()* function shall store, in the location  
 74158 referenced by *ovalue*, a value representing the previous amount of time before the timer would  
 74159 have expired, or zero if the timer was disarmed, together with the previous timer reload value.  
 74160 Timers shall not expire before their scheduled time.

74161 Only a single signal shall be queued to the process for a given timer at any point in time. When a  
 74162 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun  
 74163 shall occur. When a timer expiration signal is delivered to or accepted by a process, the  
 74164 *timer\_getoverrun()* function shall return the timer expiration overrun count for the specified  
 74165 timer. The overrun count returned contains the number of extra timer expirations that occurred  
 74166 between the time the signal was generated (queued) and when it was delivered or accepted, up



74167 to but not including an implementation-defined maximum of {DELAYTIMER\_MAX}. If the  
 74168 number of such extra expirations is greater than or equal to {DELAYTIMER\_MAX}, then the  
 74169 overrun count shall be set to {DELAYTIMER\_MAX}. The value returned by *timer\_getoverrun()*  
 74170 shall apply to the most recent expiration signal delivery or acceptance for the timer. If no  
 74171 expiration signal has been delivered for the timer, the return value of *timer\_getoverrun()* is  
 74172 unspecified.

74173 The behavior is undefined if the value specified by the *timerid* argument to *timer\_getoverrun()*,  
 74174 *timer\_gettime()*, or *timer\_settime()* does not correspond to a timer ID returned by *timer\_create()*  
 74175 but not yet deleted by *timer\_delete()*.

#### 74176 RETURN VALUE

74177 If the *timer\_getoverrun()* function succeeds, it shall return the timer expiration overrun count as  
 74178 explained above.

74179 If the *timer\_gettime()* or *timer\_settime()* functions succeed, a value of 0 shall be returned.

74180 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to  
 74181 indicate the error.

#### 74182 ERRORS

74183 The *timer\_settime()* function shall fail if:

74184 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than  
 74185 or equal to 1 000 million, and the *it\_value* member of that structure did not  
 74186 specify zero seconds and nanoseconds.

74187 The *timer\_settime()* function may fail if:

74188 [EINVAL] The *it\_interval* member of *value* is not zero and the timer was created with  
 74189 notification by creation of a new thread (*sigev\_sigev\_notify* was  
 74190 SIGEV\_THREAD) and a fixed stack address has been set in the thread  
 74191 attribute pointed to by *sigev\_notify\_attributes*.

#### 74192 EXAMPLES

74193 None.

#### 74194 APPLICATION USAGE

74195 Using fixed stack addresses is problematic when timer expiration is signaled by the creation of a  
 74196 new thread. Since it cannot be assumed that the thread created for one expiration is finished  
 74197 before the next expiration of the timer, it could happen that two threads use the same memory as  
 74198 a stack at the same time. This is invalid and produces undefined results.

#### 74199 RATIONALE

74200 Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The  
 74201 inverse of this tick rate is the clock resolution, also called the clock granularity, which in either  
 74202 case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for  
 74203 these common rates. The granularity of practical clocks implies that if one reads a given clock  
 74204 twice in rapid succession, one may get the same time value twice; and that timers must wait for  
 74205 the next clock tick after the theoretical expiration time, to ensure that a timer never returns too  
 74206 soon. Note also that the granularity of the clock may be significantly coarser than the resolution  
 74207 of the data format used to set and get time and interval values. Also note that some  
 74208 implementations may choose to adjust time and/or interval values to exactly match the ticks of  
 74209 the underlying clock.

74210 This volume of POSIX.1-2024 defines functions that allow an application to determine the  
 74211 implementation-supported resolution for the clocks and requires an implementation to  
 74212 document the resolution supported for timers and *nanosleep()* if they differ from the supported

- 74213 clock resolution. This is more of a procurement issue than a runtime application issue.
- 74214 If an implementation detects that the value specified by the *timerid* argument to  
74215 *timer\_getoverrun()*, *timer\_gettime()*, or *timer\_settime()* does not correspond to a timer ID returned  
74216 by *timer\_create()* but not yet deleted by *timer\_delete()*, it is recommended that the function  
74217 should fail and report an [EINVAL] error.
- 74218 **FUTURE DIRECTIONS**
- 74219 None.
- 74220 **SEE ALSO**
- 74221 *clock\_getres()*, *timer\_create()*
- 74222 XBD <time.h>
- 74223 **CHANGE HISTORY**
- 74224 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 74225 **Issue 6**
- 74226 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions are marked as part of the  
74227 Timers option.
- 74228 The [ENOSYS] error condition has been removed as stubs need not be provided if an  
74229 implementation does not support the Timers option.
- 74230 The [EINVAL] error condition is updated to include the following: “and the *it\_value* member of  
74231 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC  
74232 Interpretation 1003.1 #89.
- 74233 The DESCRIPTION for *timer\_getoverrun()* is updated to clarify that “If no expiration signal has  
74234 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return  
74235 value of *timer\_getoverrun()* is unspecified”.
- 74236 The **restrict** keyword is added to the *timer\_settime()* prototype for alignment with the  
74237 ISO/IEC 9899:1999 standard.
- 74238 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/140 is applied, updating the ERRORS  
74239 section so that the mandatory [EINVAL] error (“The *timerid* argument does not correspond to an  
74240 ID returned by *timer\_create()* but not yet deleted by *timer\_delete()*”) becomes optional.
- 74241 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/141 is applied, updating the ERRORS  
74242 section to include an optional [EINVAL] error for the case when a timer is created with the  
74243 notification method set to SIGEV\_THREAD. APPLICATION USAGE text is also added.
- 74244 **Issue 7**
- 74245 The *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* functions are moved from the Timers  
74246 option to the Base.
- 74247 Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 74248 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0370 [659] is applied.

74249 **NAME**

74250 times — get process and waited-for child process times

74251 **SYNOPSIS**

74252 #include &lt;sys/times.h&gt;

74253 clock\_t times(struct tms \*buffer);

74254 **DESCRIPTION**74255 The *times()* function shall fill the **tms** structure pointed to by *buffer* with time-accounting  
74256 information. The **tms** structure is defined in <sys/times.h>.

74257 All times are measured in terms of the number of clock ticks used.

74258 The times of a terminated child process shall be included in the *tms\_cutime* and *tms\_cstime*  
74259 elements of the parent when *wait()*, *waitid()*, or *waitpid()* returns the process ID of this  
74260 terminated child. If a child process has not waited for its children, their times shall not be  
74261 included in its times.74262 • The *tms\_utime* structure member is the CPU time charged for the execution of user  
74263 instructions of the calling process.74264 • The *tms\_stime* structure member is the CPU time charged for execution by the system on  
74265 behalf of the calling process.74266 • The *tms\_cutime* structure member is the sum of the *tms\_utime* and *tms\_cutime* times of the  
74267 child processes.74268 • The *tms\_cstime* structure member is the sum of the *tms\_stime* and *tms\_cstime* times of the  
74269 child processes.74270 **RETURN VALUE**74271 Upon successful completion, *times()* shall return the elapsed real time, in clock ticks, since an  
74272 arbitrary point in the past (for example, system start-up time). This point does not change from  
74273 one invocation of *times()* within the process to another. The return value may overflow the  
74274 possible range of type **clock\_t**. If *times()* fails, (**clock\_t**)-1 shall be returned and *errno* set to  
74275 indicate the error.74276 **ERRORS**74277 The *times()* function shall fail if:74278 [EOVERFLOW] The return value would overflow the range of **clock\_t**.74279 **EXAMPLES**74280 **Timing a Database Lookup**74281 The following example defines two functions, *start\_clock()* and *end\_clock()*, that are used to time  
74282 a lookup. It also defines variables of type **clock\_t** and **tms** to measure the duration of  
74283 transactions. The *start\_clock()* function saves the beginning times given by the *times()* function.  
74284 The *end\_clock()* function gets the ending times and prints the difference between the two times.74285 #include <sys/times.h>  
74286 #include <stdio.h>  
74287 ...  
74288 void start\_clock(void);  
74289 void end\_clock(char \*msg);  
74290 ...  
74291 static clock\_t st\_time;  
74292 static clock\_t en\_time;

```

74293     static struct tms st_cpu;
74294     static struct tms en_cpu;
74295     ...
74296     void
74297     start_clock()
74298     {
74299         st_time = times(&st_cpu);
74300     }

74301     /* This example assumes that the result of each subtraction
74302        is within the range of values that can be represented in
74303        an integer type. */
74304     void
74305     end_clock(char *msg)
74306     {
74307         en_time = times(&en_cpu);

74308         fputs(msg, stdout);
74309         printf("Real Time: %jd, User Time %jd, System Time %jd\n",
74310             (intmax_t)(en_time - st_time),
74311             (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
74312             (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
74313     }

```

#### 74314 APPLICATION USAGE

74315 Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per  
74316 second as it may vary from system to system.

#### 74317 RATIONALE

74318 The accuracy of the times reported is intentionally left unspecified to allow implementations  
74319 flexibility in design, from uniprocessor to multi-processor networks.

74320 The inclusion of times of child processes is recursive, so that a parent process may collect the  
74321 total times of all of its descendants. But the times of a child are only added to those of its parent  
74322 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process  
74323 can always see the total times of all its descendants; see also the discussion of the term  
74324 “realtime” in `alarm()`.

74325 If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a  
74326 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual  
74327 systems that run continuously for longer than that. This volume of POSIX.1-2024 permits an  
74328 implementation to make the reference point for the returned value be the start-up time of the  
74329 process, rather than system start-up time.

74330 The term “charge” in this context has nothing to do with billing for services. The operating  
74331 system accounts for time used in this way. That information must be correct, regardless of how  
74332 that information is used.

#### 74333 FUTURE DIRECTIONS

74334 None.

#### 74335 SEE ALSO

74336 `alarm()`, `exec`, `fork()`, `sysconf()`, `time()`, `wait()`, `waitid()`

74337 XBD <`sys/times.h`>

74338 **CHANGE HISTORY**

74339 First released in Issue 1. Derived from Issue 1 of the SVID.

74340 **Issue 7**

74341 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0371 [644] is applied.

74342 **NAME**

74343       timespec\_get — get time

74344 **SYNOPSIS**

74345       #include &lt;time.h&gt;

74346       int timespec\_get(struct timespec \*ts, int base);

74347 **DESCRIPTION**

74348 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
74349 conflict between the requirements described here and the ISO C standard is unintentional. This  
74350 volume of POSIX.1-2024 defers to the ISO C standard.

74351       The *timespec\_get()* function shall set the interval pointed to by *ts* to hold the current calendar  
74352 time based on the specified time base.

74353 CX       If *base* is `TIME_UTC`, the members of *ts* shall be set to the same values as would be set by a call  
74354 to *clock\_gettime*(`CLOCK_REALTIME`, *ts*). If the number of seconds will not fit in an object of  
74355 type `time_t`, the function shall return zero.

74356 **RETURN VALUE**

74357       If the *timespec\_get()* function is successful it shall return the non-zero value *base*; otherwise, it  
74358 shall return zero.

74359 **ERRORS**

74360       See DESCRIPTION.

74361 **EXAMPLES**

74362       None.

74363 **APPLICATION USAGE**

74364       None.

74365 **RATIONALE**

74366       None.

74367 **FUTURE DIRECTIONS**

74368       None.

74369 **SEE ALSO**74370       *clock\_getres()*, *time()*74371       XBD <[time.h](#)>74372 **CHANGE HISTORY**

74373       First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

74374 **NAME**

74375            timezone — difference from UTC and local standard time

74376 **SYNOPSIS**

```
74377 XSI        #include <time.h>
74378            extern long timezone;
```

74379 **DESCRIPTION**74380            Refer to [tzset\(\)](#).

74381 **NAME**

74382 tmpfile — create a temporary file

74383 **SYNOPSIS**

74384 #include &lt;stdio.h&gt;

74385 FILE \*tmpfile(void);

74386 **DESCRIPTION**

74387 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 74388 conflict between the requirements described here and the ISO C standard is unintentional. This  
 74389 volume of POSIX.1-2024 defers to the ISO C standard.

74390 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file  
 74391 shall be automatically deleted when all references to the file are closed. The file shall be opened  
 74392 as in *fopen()* for update (*wb+*), except that implementations may restrict the permissions, either  
 74393 by clearing the file mode bits or setting them to the value `S_IRUSR | S_IWUSR`.

74394 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is  
 74395 killed while it is processing a call to *tmpfile()*.

74396 An error message may be written to standard error if the stream cannot be opened.

74397 **RETURN VALUE**

74398 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is  
 74399 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

74400 **ERRORS**74401 The *tmpfile()* function shall fail if:

74402 CX [EINTR] A signal was caught during *tmpfile()*.

74403 CX [EMFILE] All file descriptors available to the process are currently open.

74404 CX [EMFILE] {STREAM\_MAX} streams are currently open in the calling process.

74405 CX [ENFILE] The maximum allowable number of files is currently open in the system.

74406 CX [ENOSPC] The directory or file system which would contain the new file cannot be  
 74407 expanded.

74408 The *tmpfile()* function may fail if:

74409 CX [EMFILE] {FOPEN\_MAX} streams are currently open in the calling process.

74410 CX [ENOMEM] Insufficient storage space is available.

74411 **EXAMPLES**74412 **Creating a Temporary File**

74413 The following example creates a temporary file for update, and returns a pointer to a stream for  
 74414 the created file in the *fp* variable.

74415 #include &lt;stdio.h&gt;

74416 ...

74417 FILE \*fp;

74418 fp = tmpfile ();



74419 **APPLICATION USAGE**

74420 It should be possible to open at least {TMP\_MAX} temporary files during the lifetime of the  
 74421 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number  
 74422 simultaneously open other than this limit and any limit on the number of open file descriptors  
 74423 or streams ({OPEN\_MAX}, {FOPEN\_MAX}, {STREAM\_MAX}).

74424 In multi-threaded applications, the *tmpfile()* function can leak file descriptors into child  
 74425 processes. Applications should instead use *mkostemp()* with the O\_CLOEXEC or O\_CLOFORK  
 74426 flag, or both, followed by *fdopen()*, to avoid the leak.

74427 **RATIONALE**

74428 None.

74429 **FUTURE DIRECTIONS**

74430 None.

74431 **SEE ALSO**

74432 [Section 2.5](#) (on page 521), *fopen()*, *mkdtemp()*, *tmpnam()*, *unlink()*

74433 XBD <stdio.h>

74434 **CHANGE HISTORY**

74435 First released in Issue 1. Derived from Issue 1 of the SVID.

74436 **Issue 5**

74437 Large File Summit extensions are added.

74438 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes  
 74439 in previous issues.

74440 **Issue 6**

74441 Extensions beyond the ISO C standard are marked.

74442 The following new requirements on POSIX implementations derive from alignment with the  
 74443 Single UNIX Specification:

- 74444 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support  
 74445 large files.
- 74446 • The [EMFILE] optional error condition is added.

74447 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999  
 74448 standard.

74449 **Issue 7**

74450 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may  
 74451 restrict the permissions of the file created.

74452 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

74453 SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for  
 74454 {STREAM\_MAX} streams open.

74455 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0668 [14] is applied.

74456 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0372 [678] is applied.

74457 **Issue 8**

74458 Austin Group Defects 411 and 1318 are applied, changing the APPLICATION USAGE section.

74459

Austin Group Defect 1296 is applied, removing [Eoverflow] from the ERRORS section.

74460 **NAME**

74461 tmpnam — create a name for a temporary file

74462 **SYNOPSIS**

```
74463 OB #include <stdio.h>
74464 char *tmpnam(char *s);
```

74465 **DESCRIPTION**

74466 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 74467 conflict between the requirements described here and the ISO C standard is unintentional. This  
 74468 volume of POSIX.1-2024 defers to the ISO C standard.

74469 The *tmpnam()* function shall generate a string that is a valid pathname that does not name an  
 74470 existing file. The function is potentially capable of generating {TMP\_MAX} different strings, but  
 74471 any or all of them may already be in use by existing files and thus not be suitable return values.

74472 The *tmpnam()* function generates a different string each time it is called from the same process,  
 74473 up to {TMP\_MAX} times. If it is called more than {TMP\_MAX} times, the behavior is  
 74474 implementation-defined.

74475 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 74476 *tmpnam()*.

74477 If called with a null pointer argument, the *tmpnam()* function need not be thread-safe; however,  
 74478 such calls shall avoid data races with calls to *tmpnam()* with a non-null argument and with calls  
 74479 to all other functions.

74480 **RETURN VALUE**

74481 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can  
 74482 be generated, the *tmpnam()* function shall return a null pointer.

74483 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and  
 74484 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the  
 74485 argument *s* is not a null pointer, it is presumed to point to an array of at least `L_tmpnam` **chars**;  
 74486 *tmpnam()* shall write its result in that array and shall return the argument as its value.

74487 **ERRORS**

74488 No errors are defined.

74489 **EXAMPLES**74490 **Generating a Pathname**74491 The following example generates a unique pathname and stores it in the array pointed to by *ptr*.

```
74492 #include <stdio.h>
74493 ...
74494 char pathname[L_tmpnam+1];
74495 char *ptr;
74496 ptr = tmpnam(pathname);
```

74497 **APPLICATION USAGE**74498 This function only creates pathnames. It is the application's responsibility to create and remove  
74499 the files.74500 Between the time a pathname is created and the file is opened, it is possible for some other  
74501 process to create a file with the same name. Applications may find *tmpfile()* more useful.

74502 Applications should use the *tmpfile()*, *mkostemp()*, *mkstemp()*, or *mkdtemp()* functions instead of  
74503 the obsolescent *tmpnam()* function.

74504 **RATIONALE**

74505 None.

74506 **FUTURE DIRECTIONS**

74507 The *tmpnam()* function may be removed in a future version, but not until after it has been  
74508 removed from the ISO C standard.

74509 **SEE ALSO**

74510 *fopen()*, *open()*, *mkdtemp()*, *tmpfile()*, *unlink()*

74511 XBD <stdio.h>

74512 **CHANGE HISTORY**

74513 First released in Issue 1. Derived from Issue 1 of the SVID.

74514 **Issue 5**

74515 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

74516 **Issue 6**

74517 Extensions beyond the ISO C standard are marked.

74518 The normative text is updated to avoid use of the term “must” for application requirements.

74519 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

74520 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/142 is applied, updating the  
74521 DESCRIPTION to allow implementations of the *tempnam()* function to call *tmpnam()*.

74522 **Issue 7**

74523 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *tmpnam()* function  
74524 need not be thread-safe if called with a NULL parameter.

74525 The *tmpnam()* function is marked obsolescent.

74526 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0669 [291] and XSH/TC1-2008/0670  
74527 [291,429] are applied.

74528 **Issue 8**

74529 Austin Group Defect 411 is applied, adding *mkostemp()*.

74530 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
74531 standard.

74532 Austin Group Defect 1330 is applied, changing the FUTURE DIRECTIONS section and  
74533 removing obsolescent interfaces.

74534 **NAME**

74535           tolower, tolower\_l — transliterate uppercase characters to lowercase

74536 **SYNOPSIS**

74537           #include &lt;ctype.h&gt;

74538           int tolower(int c);

74539 CX       int tolower\_l(int c, locale\_t locale);

74540 **DESCRIPTION**

74541 CX       For *tolower()*: The functionality described on this reference page is aligned with the ISO C  
74542 standard. Any conflict between the requirements described here and the ISO C standard is  
74543 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

74544 CX       The *tolower()* and *tolower\_l()* functions have as a domain a type **int**, the value of which is  
74545 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the  
74546 CX       behavior is undefined. If the argument of *tolower()* or *tolower\_l()* represents an uppercase letter,  
74547 and there exists a corresponding lowercase letter as defined by character type information in the  
74548 CX       current locale or in the locale represented by *locale*, respectively (category *LC\_CTYPE*), the  
74549 result shall be the corresponding lowercase letter. All other arguments in the domain are  
74550 returned unchanged.

74551 CX       The behavior is undefined if the *locale* argument to *tolower\_l()* is the special locale object  
74552 *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.

74553 **RETURN VALUE**

74554 CX       Upon successful completion, the *tolower()* and *tolower\_l()* functions shall return the lowercase  
74555 letter corresponding to the argument passed; otherwise, they shall return the argument  
74556 unchanged.

74557 **ERRORS**

74558           No errors are defined.

74559 **EXAMPLES**

74560           None.

74561 **APPLICATION USAGE**

74562           None.

74563 **RATIONALE**

74564           None.

74565 **FUTURE DIRECTIONS**

74566           None.

74567 **SEE ALSO**74568           [setlocale\(\)](#), [uselocale\(\)](#)74569           XBD Chapter 7 (on page 127), [<ctype.h>](#), [<locale.h>](#)74570 **CHANGE HISTORY**

74571           First released in Issue 1. Derived from Issue 1 of the SVID.

74572 **Issue 6**

74573           Extensions beyond the ISO C standard are marked.

74574 **Issue 7**

74575  
74576

The *tolower\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

74577  
74578

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0671 [283] and XSH/TC1-2008/0672 [283] are applied.

74579 **NAME**

74580            toupper, toupper\_l — transliterate lowercase characters to uppercase

74581 **SYNOPSIS**

```
74582            #include <ctype.h>
74583            int toupper(int c);
74584 CX         int toupper_l(int c, locale_t locale);
```

74585 **DESCRIPTION**

74586 CX         For `toupper()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

74589 CX         The `toupper()` and `toupper_l()` functions have as a domain a type `int`, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined.

74592 CX         If the argument of `toupper()` or `toupper_l()` represents a lowercase letter, and there exists a corresponding uppercase letter as defined by character type information in the current locale or in the locale represented by `locale`, respectively (category `LC_CTYPE`), the result shall be the corresponding uppercase letter.

74596            All other arguments in the domain are returned unchanged.

74597 CX         The behavior is undefined if the `locale` argument to `toupper_l()` is the special locale object `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

74599 **RETURN VALUE**

74600 CX         Upon successful completion, `toupper()` and `toupper_l()` shall return the uppercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

74602 **ERRORS**

74603            No errors are defined.

74604 **EXAMPLES**

74605            None.

74606 **APPLICATION USAGE**

74607            None.

74608 **RATIONALE**

74609            None.

74610 **FUTURE DIRECTIONS**

74611            None.

74612 **SEE ALSO**

74613            [setlocale\(\)](#), [uselocale\(\)](#)

74614            XBD Chapter 7 (on page 127), [<ctype.h>](#), [<locale.h>](#)

74615 **CHANGE HISTORY**

74616            First released in Issue 1. Derived from Issue 1 of the SVID.

74617 **Issue 6**

74618            Extensions beyond the ISO C standard are marked.

74619 **Issue 7**

74620 SD5-XSH-ERN-181 is applied, clarifying the RETURN VALUE section.

74621 The *toupper\_1()* function is added from The Open Group Technical Standard, 2006, Extended API  
74622 Set Part 4.

74623 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0673 [283] and XSH/TC1-2008/0674  
74624 [283] are applied.



74625 **NAME**

74626 towctrans, towctrans\_l — wide-character transliteration

74627 **SYNOPSIS**

```
74628 #include <wctype.h>
74629 wint_t towctrans(wint_t wc, wctrans_t desc);
74630 CX wint_t towctrans_l(wint_t wc, wctrans_t desc,
74631 locale_t locale);
```

74632 **DESCRIPTION**

74633 CX For `towctrans()`: The functionality described on this reference page is aligned with the ISO C  
 74634 standard. Any conflict between the requirements described here and the ISO C standard is  
 74635 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

74636 CX The `towctrans()` and `towctrans_l()` functions shall transliterate the wide-character code `wc` using  
 74637 the mapping described by `desc`.

74638 CX The current setting of the `LC_CTYPE` category in the current locale or in the locale represented  
 74639 CX by `locale`, respectively, should be the same as during the call to `wctrans()` or `wctrans_l()` that  
 74640 returned the value `desc`.

74641 If the value of `desc` is invalid (that is, not obtained by a call to `wctrans()` or `desc` is invalidated by a  
 74642 subsequent call to `setlocale()` that has affected category `LC_CTYPE`), the result is unspecified.

74643 CX If the value of `desc` is invalid (that is, not obtained by a call to `wctrans_l()` with the same locale  
 74644 object `locale`) the result is unspecified.

74645 CX An application wishing to check for error situations should set `errno` to 0 before calling  
 74646 `towctrans()` or `towctrans_l()`.

74647 If `errno` is non-zero on return, an error has occurred.

74648 The behavior is undefined if the `locale` argument to `towctrans_l()` is the special locale object  
 74649 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

74650 **RETURN VALUE**

74651 CX If successful, the `towctrans()` and `towctrans_l()` functions shall return the mapped value of `wc`  
 74652 CX using the mapping described by `desc`, or the value of `wc` unchanged if `desc` is zero. Otherwise,  
 74653 they shall return `wc` unchanged.

74654 **ERRORS**

74655 These functions may fail if:

74656 CX `[EINVAL]` `desc` contains an invalid transliteration descriptor.

74657 **EXAMPLES**

74658 None.

74659 **APPLICATION USAGE**

74660 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the  
 74661 table below, the functions in the left column are equivalent to the functions in the right column.

74662	<code>tolower(wc)</code>	<code>towctrans(wc, wctrans("tolower"))</code>
74663	<code>tolower_l(wc, locale)</code>	<code>towctrans_l(wc, wctrans("tolower"), locale)</code>
74664	<code>toupper(wc)</code>	<code>towctrans(wc, wctrans("toupper"))</code>
74665	<code>toupper_l(wc, locale)</code>	<code>towctrans_l(wc, wctrans("toupper"), locale)</code>

74666 **RATIONALE**

74667 None.

74668 **FUTURE DIRECTIONS**

74669 None.

74670 **SEE ALSO**74671 *tolower()*, *toupper()*, *wctrans()*74672 XBD <**wctype.h**>74673 **CHANGE HISTORY**

74674 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

74675 **Issue 6**

74676 Extensions beyond the ISO C standard are marked.

74677 **Issue 7**74678 The *towctrans\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.74679  
74680 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0675 [302], XSH/TC1-2008/0676 [283],  
74681 and XSH/TC1-2008/0677 [283] are applied.74682 **Issue 8**74683 Austin Group Defect 1302 is applied, aligning the *towctrans()* function with the  
74684 ISO/IEC 9899:2018 standard.

74685 **NAME**

74686           tolower, tolower\_l — transliterate uppercase wide-character code to lowercase

74687 **SYNOPSIS**

74688           #include &lt;wctype.h&gt;

74689           wint\_t tolower(wint\_t wc);

74690 CX       wint\_t tolower\_l(wint\_t wc, locale\_t locale);

74691 **DESCRIPTION**74692 CX       For *tolower()*: The functionality described on this reference page is aligned with the ISO C  
74693 standard. Any conflict between the requirements described here and the ISO C standard is  
74694 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.74695 CX       The *tolower()* and *tolower\_l()* functions have as a domain a type **wint\_t**, the value of which  
74696 the application shall ensure is a character representable as a **wchar\_t**, and a wide-character code  
74697 corresponding to a valid character in the locale used by the function or the value of WEOF. If  
74698 CX       the argument has any other value, the behavior is undefined. If the argument of *tolower()* or  
74699 *tolower\_l()* represents an uppercase wide-character code, and there exists a corresponding  
74700 CX       lowercase wide-character code as defined by character type information in the current locale or  
74701 in the locale represented by *locale*, respectively (category *LC\_CTYPE*), the result shall be the  
74702 corresponding lowercase wide-character code. All other arguments in the domain are returned  
74703 unchanged.74704 CX       The behavior is undefined if the *locale* argument to *tolower\_l()* is the special locale object  
74705 *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.74706 **RETURN VALUE**74707 CX       Upon successful completion, the *tolower()* and *tolower\_l()* functions shall return the  
74708 lowercase letter corresponding to the argument passed; otherwise, they shall return the  
74709 argument unchanged.74710 **ERRORS**

74711           No errors are defined.

74712 **EXAMPLES**

74713           None.

74714 **APPLICATION USAGE**

74715           None.

74716 **RATIONALE**

74717           None.

74718 **FUTURE DIRECTIONS**

74719           None.

74720 **SEE ALSO**74721           [setlocale\(\)](#), [uselocale\(\)](#)74722           XBD Chapter 7 (on page 127), [<locale.h>](#), [<wctype.h>](#)74723 **CHANGE HISTORY**

74724           First released in Issue 4.

74725 **Issue 5**

74726 The following change has been made in this version for alignment with  
74727 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 74728       • The SYNOPSIS has been changed to indicate that this function and associated data types  
74729       are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

74730 **Issue 6**

74731 The normative text is updated to avoid use of the term “must” for application requirements.

74732 **Issue 7**

74733 The `towlower_l()` function is added from The Open Group Technical Standard, 2006, Extended  
74734 API Set Part 4.

74735 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0678 [302], XSH/TC1-2008/0679 [283],  
74736 and XSH/TC1-2008/0680 [283] are applied.

74737 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0373 [685] is applied.

74738 **NAME**

74739 towupper, towupper\_l — transliterate lowercase wide-character code to uppercase

74740 **SYNOPSIS**

74741 #include &lt;wctype.h&gt;

74742 wint\_t towupper(wint\_t wc);

74743 CX wint\_t towupper\_l(wint\_t wc, locale\_t locale);

74744 **DESCRIPTION**74745 CX For *towupper()*: The functionality described on this reference page is aligned with the ISO C  
74746 standard. Any conflict between the requirements described here and the ISO C standard is  
74747 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.74748 CX The *towupper()* and *towupper\_l()* functions have as a domain a type **wint\_t**, the value of which  
74749 the application shall ensure is a character representable as a **wchar\_t**, and a wide-character code  
74750 corresponding to a valid character in the locale used by the function or the value of WEOF. If  
74751 CX the argument has any other value, the behavior is undefined. If the argument of *towupper()* or  
74752 *towupper\_l()* represents a lowercase wide-character code, and there exists a corresponding  
74753 CX uppercase wide-character code as defined by character type information in the current locale or  
74754 in the locale represented by *locale*, respectively (category *LC\_CTYPE*), the result shall be the  
74755 corresponding uppercase wide-character code. All other arguments in the domain are returned  
74756 unchanged.74757 CX The behavior is undefined if the *locale* argument to *towupper\_l()* is the special locale object  
74758 *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.74759 **RETURN VALUE**74760 CX Upon successful completion, the *towupper()* and *towupper\_l()* functions shall return the  
74761 uppercase letter corresponding to the argument passed. Otherwise, they shall return the  
74762 argument unchanged.74763 **ERRORS**

74764 No errors are defined.

74765 **EXAMPLES**

74766 None.

74767 **APPLICATION USAGE**

74768 None.

74769 **RATIONALE**

74770 None.

74771 **FUTURE DIRECTIONS**

74772 None.

74773 **SEE ALSO**74774 [setlocale\(\)](#), [uselocale\(\)](#)74775 XBD [Chapter 7](#) (on page 127), [<locale.h>](#), [<wctype.h>](#)74776 **CHANGE HISTORY**

74777 First released in Issue 4.

74778 **Issue 5**

74779 The following change has been made in this version for alignment with  
74780 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 74781 • The SYNOPSIS has been changed to indicate that this function and associated data types  
74782 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

74783 **Issue 6**

74784 The normative text is updated to avoid use of the term “must” for application requirements.

74785 **Issue 7**

74786 The `towupper_l()` function is added from The Open Group Technical Standard, 2006, Extended  
74787 API Set Part 4.

74788 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0681 [302], XSH/TC1-2008/0682 [283],  
74789 and XSH/TC1-2008/0683 [283] are applied.

74790 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0374 [685] is applied.

74791 **NAME**

74792 trunc, truncf, trunc1 — round to truncated integer value

74793 **SYNOPSIS**

74794 #include &lt;math.h&gt;

74795 double trunc(double x);

74796 float truncf(float x);

74797 long double trunc1(long double x);

74798 **DESCRIPTION**74799 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
74800 conflict between the requirements described here and the ISO C standard is unintentional. This  
74801 volume of POSIX.1-2024 defers to the ISO C standard.74802 These functions shall round their argument to the integer value, in floating format, nearest to but  
74803 no larger in magnitude than the argument.

74804 MX These functions may raise the inexact floating-point exception for finite non-integer arguments.

74805 **RETURN VALUE**

74806 Upon successful completion, these functions shall return the truncated integer value.

74807 MX The returned value shall be exact, shall be independent of the current rounding direction mode,  
74808 and shall have the same sign as  $x$ .74809 MX If  $x$  is NaN, a NaN shall be returned.74810 If  $x$  is  $\pm 0$  or  $\pm \text{Inf}$ ,  $x$  shall be returned.74811 **ERRORS**

74812 No errors are defined.

74813 **EXAMPLES**

74814 None.

74815 **APPLICATION USAGE**74816 The integral value returned by these functions need not be expressible as an `intmax_t`. The  
74817 return value should be tested before assigning it to an integer type to avoid the undefined  
74818 results of an integer overflow.74819 **RATIONALE**

74820 None.

74821 **FUTURE DIRECTIONS**

74822 None.

74823 **SEE ALSO**

74824 XBD &lt;math.h&gt;

74825 **CHANGE HISTORY**

74826 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

74827 **Issue 7**

74828 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0684 [346] is applied.

74829 **Issue 8**74830 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
74831 standard.

74832 **NAME**

74833 truncate — truncate a file to a specified length

74834 **SYNOPSIS**

74835 #include &lt;unistd.h&gt;

74836 int truncate(const char \*path, off\_t length);

74837 **DESCRIPTION**74838 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be  
74839 equal to *length* bytes.74840 If the file previously was larger than *length*, the extra data is discarded. If the file was previously  
74841 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

74842 The application shall ensure that the process has write permission for the file.

74843 If the request would cause the file size to exceed the soft file size limit for the process, the  
74844 XSI request shall fail and the implementation shall generate a SIGXFSZ signal for the thread.74845 The *truncate()* function shall not modify the file offset for any open file descriptions associated  
74846 with the file. Upon successful completion, *truncate()* shall mark for update the last data  
74847 modification and last file status change timestamps of the file, and the S\_ISUID and S\_ISGID bits  
74848 of the file mode may be cleared.74849 **RETURN VALUE**74850 Upon successful completion, *truncate()* shall return 0. Otherwise, -1 shall be returned, and *errno*  
74851 set to indicate the error.74852 **ERRORS**74853 The *truncate()* function shall fail if:74854 [EACCES] A component of the path prefix denies search permission, or write permission  
74855 is denied on the file.

74856 [EFBIG] or [EINVAL]

74857 The *length* argument is greater than the maximum file size.74858 XSI [EFBIG] The *length* argument exceeds the file size limit of the process. A SIGFSZ  
74859 signal shall also be generated for the thread.

74860 [EINTR] A signal was caught during execution.

74861 [EINVAL] The *length* argument is less than 0 or the *path* argument refers to a file, other  
74862 than a directory, on which this operation is not possible (for example, a FIFO  
74863 or socket).

74864 [EIO] An I/O error occurred while reading from or writing to a file system.

74865 [EISDIR] The named file is a directory.

74866 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
74867 argument.

74868 [ENAMETOOLONG]

74869 The length of a component of a pathname is longer than {NAME\_MAX}.

74870 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.74871 [ENOTDIR] A component of the path prefix names an existing file that is neither a  
74872 directory nor a symbolic link to a directory, or the *path* argument contains at  
74873 least one non-`<slash>` character and ends with one or more trailing `<slash>`



74874 characters and the last pathname component names an existing file that is  
 74875 neither a directory nor a symbolic link to a directory.

74876 [EROFS] The named file resides on a read-only file system.

74877 The *truncate()* function may fail if:

74878 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
 74879 resolution of the *path* argument.

74880 [ENAMETOOLONG]  
 74881 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a  
 74882 symbolic link produced an intermediate result with a length that exceeds  
 74883 {PATH\_MAX}.

**74884 EXAMPLES**

74885 None.

**74886 APPLICATION USAGE**

74887 None.

**74888 RATIONALE**

74889 None.

**74890 FUTURE DIRECTIONS**

74891 None.

**74892 SEE ALSO**

74893 [open\(\)](#)

74894 XBD <[unistd.h](#)>

**74895 CHANGE HISTORY**

74896 First released in Issue 4, Version 2.

**74897 Issue 5**

74898 Moved from X/OPEN UNIX extension to BASE.

74899 Large File Summit extensions are added.

**74900 Issue 6**

74901 This reference page is split out from the *ftruncate()* reference page.

74902 The normative text is updated to avoid use of the term “must” for application requirements.

74903 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 74904 [ELOOP] error condition is added.

**74905 Issue 7**

74906 Austin Group Interpretation 1003.1-2001 #143 is applied.

74907 The *truncate()* function is moved from the XSI option to the Base.

74908 Changes are made related to support for finegrained timestamps.

74909 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
 74910 pathname exists but is not a directory or a symbolic link to a directory.

74911 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0685 [324] is applied.

74912 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0375 [489] is applied.

74913 **Issue 8**

74914 Austin Group Defects 308 and 1087 are applied, clarifying the handling of [EFBIG] errors.

74915 Austin Group Defect 1381 is applied, adding a second condition to the [EINVAL] error and  
74916 rearranging the ERRORS section into alphabetical order.

74917 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
74918 relating to the file size limit for the process.

74919 **NAME**  
74920 truncf, trunc — round to truncated integer value

74921 **SYNOPSIS**  
74922 #include <math.h>  
74923 float truncf(float x);  
74924 long double trunc (long double x);

74925 **DESCRIPTION**  
74926 Refer to *trunc()*.

74927 **NAME**

74928 tsearch — search a binary search tree

74929 **SYNOPSIS**

```
74930 XSI #include <search.h>
74931 void *tsearch(const void *key, void **rootp,
74932 int (*compar)(const void *, const void *));
```

74933 **DESCRIPTION**

74934 Refer to [tdelete\(\)](#).

74935 **NAME**

74936 tss\_create — thread-specific data key creation

74937 **SYNOPSIS**

74938 #include &lt;threads.h&gt;

74939 int tss\_create(tss\_t \*key, tss\_dtor\_t dtor);

74940 **DESCRIPTION**

74941 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 74942 conflict between the requirements described here and the ISO C standard is unintentional. This  
 74943 volume of POSIX.1-2024 defers to the ISO C standard.

74944 The *tss\_create()* function shall create a thread-specific storage pointer with destructor *dtor*, which  
 74945 can be null.

74946 A null pointer value shall be associated with the newly created key in all existing threads. Upon  
 74947 subsequent thread creation, the value associated with all keys shall be initialized to a null  
 74948 pointer value in the new thread.

74949 Destructors associated with thread-specific storage shall not be invoked at process termination.

74950 The behavior is undefined if the *tss\_create()* function is called from within a destructor.

74951 CX The *tss\_create()* function shall not be affected if the calling thread executes a signal handler  
 74952 during the call.

74953 **RETURN VALUE**

74954 If the *tss\_create()* function is successful, it shall set the thread-specific storage pointed to by *key* to  
 74955 a value that uniquely identifies the newly created pointer and shall return *thrd\_success*;  
 74956 otherwise, *thrd\_error* shall be returned and the thread-specific storage pointed to by *key* has  
 74957 an indeterminate value.

74958 **ERRORS**

74959 No errors are defined.

74960 **EXAMPLES**

74961 None.

74962 **APPLICATION USAGE**

74963 The *tss\_create()* function performs no implicit synchronization. It is the responsibility of the  
 74964 application writer to ensure that it is called exactly once per key before use of the key.

74965 **RATIONALE**

74966 If the value associated with a key needs to be updated during the lifetime of the thread, it may  
 74967 be necessary to release the storage associated with the old value before the new value is bound.  
 74968 Although the *tss\_set()* function could do this automatically, this feature is not needed often  
 74969 enough to justify the added complexity. Instead, the application is responsible for freeing the  
 74970 stale storage:

74971 old = tss\_get(key);

74972 new = allocate();

74973 destructor(old);

74974 tss\_set(key, new);

74975 There is no notion of a destructor-safe function. If an application does not call *thrd\_exit()* or  
 74976 *pthread\_exit()* from a signal handler, or if, while calling async-unsafe functions, it blocks any  
 74977 signal whose handler might call *thrd\_exit()* or *pthread\_exit()*, all functions can be safely called  
 74978 from destructors.

74979           The *tss\_create()* function is not affected by signal handlers for the reasons stated in XRAT [Section](#)  
74980           [B.2.3](#) (on page 3742).

74981   **FUTURE DIRECTIONS**

74982           None.

74983   **SEE ALSO**

74984           [pthread\\_exit\(\)](#), [pthread\\_key\\_create\(\)](#), [thrd\\_exit\(\)](#), [tss\\_delete\(\)](#), [tss\\_get\(\)](#)

74985           XBD [<threads.h>](#)

74986   **CHANGE HISTORY**

74987           First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

74988 **NAME**

74989 tss\_delete — thread-specific data key deletion

74990 **SYNOPSIS**

74991 #include &lt;threads.h&gt;

74992 void tss\_delete(tss\_t key);

74993 **DESCRIPTION**

74994 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
74995 conflict between the requirements described here and the ISO C standard is unintentional. This  
74996 volume of POSIX.1-2024 defers to the ISO C standard.

74997 The *tss\_delete()* function shall release any resources used by the thread-specific storage identified  
74998 by *key*. The thread-specific data values associated with *key* need not be null at the time  
74999 *tss\_delete()* is called. It is the responsibility of the application to free any application storage or  
75000 perform any cleanup actions for data structures related to the deleted key or associated thread-  
75001 specific data in any threads; this cleanup can be done either before or after *tss\_delete()* is called.

75002 The application shall ensure that the *tss\_delete()* function is only called with a value for *key* that  
75003 was returned by a call to *tss\_create()* before the thread commenced executing destructors.

75004 If *tss\_delete()* is called while another thread is executing destructors, whether this will affect the  
75005 number of invocations of the destructor associated with *key* on that thread is unspecified.

75006 The *tss\_delete()* function shall be callable from within destructor functions. Calling *tss\_delete()*  
75007 shall not result in the invocation of any destructors. Any destructor function that was associated  
75008 with *key* shall no longer be called upon thread exit.

75009 Any attempt to use *key* following the call to *tss\_delete()* results in undefined behavior.

75010 CX The *tss\_delete()* function shall not be affected if the calling thread executes a signal handler  
75011 during the call.

75012 **RETURN VALUE**

75013 This function shall not return a value.

75014 **ERRORS**

75015 No errors are defined.

75016 **EXAMPLES**

75017 None.

75018 **APPLICATION USAGE**

75019 None.

75020 **RATIONALE**

75021 A thread-specific data key deletion function has been included in order to allow the resources  
75022 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys  
75023 can arise, among other scenarios, when a dynamically loaded module that allocated a key is  
75024 unloaded.

75025 Conforming applications are responsible for performing any cleanup actions needed for data  
75026 structures associated with the key to be deleted, including data referenced by thread-specific  
75027 data values. No such cleanup is done by *tss\_delete()*. In particular, destructor functions are not  
75028 called. See the RATIONALE for *pthread\_key\_delete()* for the reasons for this division of  
75029 responsibility.

75030 The *tss\_delete()* function is not affected by signal handlers for the reasons stated in XRAT [Section](#)  
75031 [B.2.3](#) (on page 3742).

75032 **FUTURE DIRECTIONS**

75033 None.

75034 **SEE ALSO**

75035 *pthread\_key\_delete()*, *tss\_create()*

75036 XBD <threads.h>

75037 **CHANGE HISTORY**

75038 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.



75039 **NAME**

75040 tss\_get, tss\_set — thread-specific data management

75041 **SYNOPSIS**

```
75042 #include <threads.h>
75043 void *tss_get(tss_t key);
75044 int tss_set(tss_t key, void *val);
```

75045 **DESCRIPTION**

75046 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 75047 conflict between the requirements described here and the ISO C standard is unintentional. This  
 75048 volume of POSIX.1-2024 defers to the ISO C standard.

75049 The `tss_get()` function shall return the value for the current thread held in the thread-specific  
 75050 storage identified by `key`.

75051 The `tss_set()` function shall set the value for the current thread held in the thread-specific storage  
 75052 identified by `key` to `val`. This action shall not invoke the destructor associated with the `key` on the  
 75053 value being replaced.

75054 The application shall ensure that the `tss_get()` and `tss_set()` functions are only called with a value  
 75055 for `key` that was returned by a call to `tss_create()` before the thread commenced executing  
 75056 destructors.

75057 The effect of calling `tss_get()` or `tss_set()` after `key` has been deleted with `tss_delete()` is undefined.

75058 CX Both `tss_get()` and `tss_set()` can be called from a thread-specific data destructor function. A call  
 75059 to `tss_get()` for the thread-specific data key being destroyed shall return a null pointer, unless the  
 75060 value is changed (after the destructor starts) by a call to `tss_set()`. Calling `tss_set()` from a  
 75061 thread-specific data destructor function may result either in lost storage (after at least  
 75062 {PTHREAD\_DESTRUCTOR\_ITERATIONS} attempts at destruction) or in an infinite loop.

75063 These functions shall not be affected if the calling thread executes a signal handler during the  
 75064 call.

75065 **RETURN VALUE**

75066 The `tss_get()` function shall return the value for the current thread. If no thread-specific data  
 75067 value is associated with `key`, then a null pointer shall be returned.

75068 The `tss_set()` function shall return `thrd_success` on success or `thrd_error` if the request  
 75069 could not be honored.

75070 **ERRORS**

75071 No errors are defined.

75072 **EXAMPLES**

75073 None.

75074 **APPLICATION USAGE**

75075 None.

75076 **RATIONALE**

75077 These functions are not affected by signal handlers for the reasons stated in XRAT [Section B.2.3](#)  
 75078 (on page 3742).

75079 **FUTURE DIRECTIONS**

75080 None.

75081 **SEE ALSO**

75082 *pthread\_getspecific(), tss\_create()*

75083 XBD <**threads.h**>

75084 **CHANGE HISTORY**

75085 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

75086 **NAME**75087 `ttyname, ttyname_r` — find the pathname of a terminal75088 **SYNOPSIS**75089 `#include <unistd.h>`75090 `char *ttyname(int fd);`75091 `int ttyname_r(int fd, char *name, size_t namesize);`75092 **DESCRIPTION**

75093 The `ttyname()` function shall return a pointer to a string containing a null-terminated pathname  
75094 of the terminal associated with file descriptor *fd*. The application shall not modify the string  
75095 returned. The returned pointer might be invalidated or the string content might be overwritten  
75096 by a subsequent call to `ttyname()`. The returned pointer and the string content might also be  
75097 invalidated if the calling thread is terminated.

75098 The `ttyname()` function need not be thread-safe.

75099 The `ttyname_r()` function shall store the null-terminated pathname of the terminal associated  
75100 with the file descriptor *fd* in the character array referenced by *name*. The array is *namesize*  
75101 characters long and should have space for the name and the terminating null character. The  
75102 maximum length of the terminal name shall be {TTY\_NAME\_MAX}.

75103 **RETURN VALUE**

75104 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null  
75105 pointer shall be returned and `errno` set to indicate the error.

75106 If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be  
75107 returned to indicate the error.

75108 **ERRORS**

75109 These functions may fail if:

75110 [EBADF] The *fd* argument is not a valid file descriptor.

75111 [ENOTTY] The file associated with the *fd* argument is not a terminal.

75112 The `ttyname_r()` function shall fail if:

75113 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned  
75114 including the terminating null character.

75115 **EXAMPLES**

75116 None.

75117 **APPLICATION USAGE**

75118 None.

75119 **RATIONALE**

75120 The term “terminal” is used instead of the historical term “terminal device” in order to avoid a  
75121 reference to an undefined term.

75122 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-  
75123 zero value if it fails. The non-thread-safe version may return the name in a static data area that  
75124 may be overwritten by each call.

75125 **FUTURE DIRECTIONS**

75126 None.

75127 **SEE ALSO**

75128 XBD <unistd.h>

75129 **CHANGE HISTORY**

75130 First released in Issue 1. Derived from Issue 1 of the SVID.

75131 **Issue 5**

75132 The *ttynamename\_r()* function is included for alignment with the POSIX Threads Extension.

75133 A note indicating that the *ttynamename()* function need not be reentrant is added to the  
75134 DESCRIPTION.

75135 **Issue 6**

75136 The *ttynamename\_r()* function is marked as part of the Thread-Safe Functions option.

75137 The following new requirements on POSIX implementations derive from alignment with the  
75138 Single UNIX Specification:

- 75139 • The statement that *errno* is set on error is added.
- 75140 • The [EBADF] and [ENOTTY] optional error conditions are added.

75141 **Issue 7**

75142 Austin Group Interpretation 1003.1-2001 #156 is applied.

75143 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

75144 The *ttynamename\_r()* function is moved from the Thread-Safe Functions option to the Base.

75145 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0686 [75] is applied.

75146 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0376 [656] is applied.

75147 **Issue 8**

75148 Austin Group Defect 398 is applied, combining the duplicated [EBADF] and [ENOTTY] errors  
75149 and changing the [ERANGE] error from “may fail” to “shall fail”.

75150 **NAME**

75151 twalk — traverse a binary search tree

75152 **SYNOPSIS**

```
75153 XSI #include <search.h>
75154 void twalk(const void *root,
75155           void (*action)(const void *, VISIT, int ));
```

75156 **DESCRIPTION**75157 Refer to *tdelete()*.

75158 **NAME**

75159 daylight, timezone, tzname, tzset — set timezone conversion information

75160 **SYNOPSIS**

75161 #include &lt;time.h&gt;

```
75162 XSI extern int daylight;
75163      extern long timezone;
75164 CX   extern char *tzname[2];
75165      void tzset(void);
```

75166 **DESCRIPTION**

75167 The `tzset()` function shall use the value of the environment variable `TZ` to set time conversion information used by `ctime()`, `localtime()`, `mktime()`, and `strptime()`. If `TZ` is absent from the environment, implementation-defined default timezone information shall be used.

75170 The `tzset()` function shall set the external variable `tzname` as follows:

```
75171      tzname[0] = "std";
75172      tzname[1] = "dst";
```

75173 where `std` and `dst` are as described in XBD Chapter 8 (on page 167).

75174 XSI The `tzset()` function also shall set the external variable `daylight` to 0 if Daylight Saving Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable `timezone` shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

75178 XSI If a thread accesses `tzname`, `daylight`, or `timezone` directly while another thread is in a call to `tzset()`, or to any function that is required or allowed to set timezone information as if by calling `tzset()`, the behavior is undefined.

75181 **RETURN VALUE**

75182 The `tzset()` function shall not return a value.

75183 **ERRORS**

75184 No errors are defined.

75185 **EXAMPLES**

75186 Example `TZ` variables and their timezone differences are given in the table below:

<i>TZ</i>	<i>timezone</i>
EST5EDT	5*60*60
GMT0	0*60*60
JST-9	-9*60*60
MET-1MEST	-1*60*60
MST7MDT	7*60*60
PST8PDT	8*60*60

75194 **APPLICATION USAGE**

75195 Since the `ctime()`, `localtime()`, `mktime()`, `strptime()`, and `strptime_l()` functions are required to set timezone information as if by calling `tzset()`, there is no need for an explicit `tzset()` call before using these functions. However, portable applications should call `tzset()` explicitly before using `localtime_r()` because setting timezone information is optional for that function.

75199 **RATIONALE**

75200 None.

75201 **FUTURE DIRECTIONS**

75202 None.

75203 **SEE ALSO**75204 *ctime()*, *localtime()*, *mktime()*, *strftime()*75205 XBD Chapter 8 (on page 167), [<time.h>](#)75206 **CHANGE HISTORY**

75207 First released in Issue 1. Derived from Issue 1 of the SVID.

75208 **Issue 6**

75209 The example is corrected.

75210 **Issue 7**

75211 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0377 [880] is applied.

75212 **Issue 8**

75213 Austin Group Defect 1253 is applied, changing ``Daylight Savings'' to ``Daylight Saving''.

75214 Austin Group Defect 1410 is applied, removing the *ctime\_r()* function.

75215 **NAME**

75216 umask — set and get the file mode creation mask

75217 **SYNOPSIS**

75218 #include &lt;sys/stat.h&gt;

75219 mode\_t umask(mode\_t *cmask*);75220 **DESCRIPTION**

75221 The *umask()* function shall set the file mode creation mask of the process to *cmask* and return the  
 75222 previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) shall be  
 75223 XSI used; the S\_ISVTX bit shall be ignored, and the meaning of the other bits is implementation-  
 75224 defined.

75225 The file mode creation mask of the process is used to turn off permission bits in the *mode*  
 75226 argument supplied during calls to the following functions:

- 75227 • *open()*, *openat()*, *creat()*, *mkdir()*, *mkdirat()*, *mkfifo()*, and *mkfifoat()*
- 75228 XSI • *mknod()*, *mknodat()*
- 75229 MSG • *mq\_open()*
- 75230 • *sem\_open()*

75231 Permission bit positions that are set in *cmask* are cleared in the mode of the created file.

75232 **RETURN VALUE**

75233 The file permission bits in the value returned by *umask()* shall be the previous value of the file  
 75234 XSI mode creation mask. The S\_ISVTX bit in the returned value shall be clear. The state of any  
 75235 other bits in the returned value is unspecified, except that a subsequent call to *umask()* with the  
 75236 returned value as *cmask* shall leave the state of the mask the same as its state before the first call,  
 75237 including any unspecified use of those bits.

75238 **ERRORS**

75239 No errors are defined.

75240 **EXAMPLES**

75241 None.

75242 **APPLICATION USAGE**

75243 None.

75244 **RATIONALE**

75245 Unsigned argument and return types for *umask()* were proposed. The return type and the  
 75246 argument were both changed to **mode\_t**.

75247 Historical implementations have made use of additional bits in *cmask* for their implementation-  
 75248 defined purposes. The addition of the text that the meaning of other bits of the field is  
 75249 implementation-defined permits these implementations to conform to this volume of  
 75250 POSIX.1-2024.

75251 **FUTURE DIRECTIONS**

75252 None.

75253 **SEE ALSO**75254 *creat()*, *exec*, *mkdir()*, *mkfifo()*, *mknod()*, *mq\_open()*, *open()*, *sem\_open()*

75255 XBD &lt;sys/stat.h&gt;, &lt;sys/types.h&gt;



75256 **CHANGE HISTORY**

75257 First released in Issue 1. Derived from Issue 1 of the SVID.

75258 **Issue 6**

75259 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

75260 The following new requirements on POSIX implementations derive from alignment with the  
75261 Single UNIX Specification:

- 75262 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was  
75263 required for conforming implementations of previous POSIX specifications, it was not  
75264 required for UNIX applications.

75265 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/143 is applied, adding the `mknod()`,  
75266 `mq_open()`, and `sem_open()` functions to the DESCRIPTION and SEE ALSO sections.

75267 **Issue 8**

75268 Austin Group Defect 1522 is applied, adding requirements relating to the S\_ISVTX bit.

75269 **NAME**

75270            uname — get the name of the current system

75271 **SYNOPSIS**

75272            #include <sys/utsname.h>

75273            int uname(struct utsname \*name);

75274 **DESCRIPTION**

75275            The *uname()* function shall store information identifying the current system in the structure pointed to by *name*.

75277            The *uname()* function uses the **utsname** structure defined in <sys/utsname.h>.

75278            The *uname()* function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

75283            The format of each member is implementation-defined.

75284 **RETURN VALUE**

75285            Upon successful completion, a non-negative value shall be returned. Otherwise,  $-1$  shall be returned and *errno* set to indicate the error.

75287 **ERRORS**

75288            No errors are defined.

75289 **EXAMPLES**

75290            None.

75291 **APPLICATION USAGE**

75292            The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

75294 **RATIONALE**

75295            The values of the structure members are not constrained to have any relation to the version of this volume of POSIX.1-2024 implemented in the operating system. An application should instead depend on `_POSIX_VERSION` and related constants defined in <unistd.h>.

75298            This volume of POSIX.1-2024 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

75302            The *uname()* function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

75305            4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value, respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

75309 **FUTURE DIRECTIONS**

75310            None.

75311 **SEE ALSO**

75312 XBD <[sys/utsname.h](#)>

75313 **CHANGE HISTORY**

75314 First released in Issue 1. Derived from Issue 1 of the SVID.

75315 **NAME**

75316 ungetc — push byte back into input stream

75317 **SYNOPSIS**

75318 #include &lt;stdio.h&gt;

75319 int ungetc(int *c*, FILE \**stream*);75320 **DESCRIPTION**

75321 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 75322 conflict between the requirements described here and the ISO C standard is unintentional. This  
 75323 volume of POSIX.1-2024 defers to the ISO C standard.

75324 The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back  
 75325 onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by  
 75326 subsequent reads on that stream in the reverse order of their pushing. A successful intervening  
 75327 CX call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fseeko()*,  
 75328 CX *fsetpos()*, or *rewind()*) or *fflush()* shall discard any pushed-back bytes for the stream. The  
 75329 external storage corresponding to the stream shall be unchanged.

75330 One byte of push-back shall be provided. If *ungetc()* is called too many times on the same stream  
 75331 without an intervening read or file-positioning operation on that stream, the operation may fail.

75332 If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall  
 75333 be left unchanged.

75334 A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The file-position  
 75335 indicator for the stream shall be decremented by each successful call to *ungetc()*; if its value was  
 75336 0 before a call, its value is unspecified after the call. The value of the file-position indicator after  
 75337 all pushed-back bytes have been read shall be the same as it was before the bytes were pushed  
 75338 back.

75339 **RETURN VALUE**

75340 Upon successful completion, *ungetc()* shall return the byte pushed back after conversion.  
 75341 Otherwise, it shall return EOF.

75342 **ERRORS**

75343 No errors are defined.

75344 **EXAMPLES**

75345 None.

75346 **APPLICATION USAGE**

75347 None.

75348 **RATIONALE**

75349 The ISO C standard includes the text “The value of the file position indicator for the stream after  
 75350 reading or discarding all pushed-back characters shall be the same as it was before the characters  
 75351 were pushed back.” POSIX.1 omits “or discarding” from this because it is redundant—in the  
 75352 ISO C standard the discarding is done by file positioning functions and does not affect the  
 75353 position set by those functions. In particular, a relative seek using *fseek()* or *fseeko()* with  
 75354 SEEK\_CUR adjusts the position relative to the position on entry to the function, not the position  
 75355 after the pushed-back bytes have been discarded. POSIX.1 also requires *fflush()* to discard  
 75356 pushed back bytes in situations where the ISO C standard says the behavior of *fflush()* is  
 75357 undefined.

75358 **FUTURE DIRECTIONS**

75359 The ISO C standard states that the use of *ungetc()* on a binary stream where the file position  
75360 indicator is zero prior to the call is an obsolescent feature. In POSIX.1 there is no distinction  
75361 between binary and text streams, so this applies to all streams. This feature may be removed in  
75362 a future version of this standard.

75363 **SEE ALSO**

75364 [Section 2.5](#) (on page 521), [fseek\(\)](#), [getc\(\)](#), [fsetpos\(\)](#), [read\(\)](#), [rewind\(\)](#), [setbuf\(\)](#)

75365 XBD [<stdio.h>](#)

75366 **CHANGE HISTORY**

75367 First released in Issue 1. Derived from Issue 1 of the SVID.

75368 **Issue 7**

75369 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0687 [87,93], XSH/TC1-2008/0688  
75370 [87], and XSH/TC1-2008/0689 [14] are applied.

75371 **Issue 8**

75372 Austin Group Defect 701 is applied, clarifying how the file-position indicator for the stream is  
75373 updated.

75374 Austin Group Defect 1302 is applied, changing the FUTURE DIRECTIONS section.

75375 **NAME**

75376 ungetwc — push wide-character code back into the input stream

75377 **SYNOPSIS**

75378 #include &lt;stdio.h&gt;

75379 #include &lt;wchar.h&gt;

75380 wint\_t ungetwc(wint\_t *wc*, FILE \**stream*);75381 **DESCRIPTION**

75382 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 75383 conflict between the requirements described here and the ISO C standard is unintentional. This  
 75384 volume of POSIX.1-2024 defers to the ISO C standard.

75385 The *ungetwc()* function shall push the character corresponding to the wide-character code  
 75386 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters  
 75387 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A  
 75388 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function  
 75389 CX (*fseek()*, *fseeko()*, *fsetpos()*, or *rewind()*) or *fflush()* shall discard any pushed-back characters for  
 75390 the stream. The external storage corresponding to the stream is unchanged.

75391 At least one character of push-back shall be provided. If *ungetwc()* is called too many times on  
 75392 the same stream without an intervening read or file-positioning operation on that stream, the  
 75393 operation may fail.

75394 If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream  
 75395 shall be left unchanged.

75396 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the  
 75397 file-position indicator for the stream after a successful call to *ungetwc()* is unspecified until all  
 75398 pushed-back wide characters are read or discarded; its value after all pushed-back wide  
 75399 characters have been read shall be the same as it was before the wide characters were pushed  
 75400 back.

75401 **RETURN VALUE**

75402 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to  
 75403 the pushed-back character. Otherwise, it shall return WEOF.

75404 **ERRORS**75405 The *ungetwc()* function may fail if:

75406 CX [EILSEQ] An invalid character sequence is detected, or a wide-character code does not  
 75407 correspond to a valid character.

75408 **EXAMPLES**

75409 None.

75410 **APPLICATION USAGE**

75411 None.

75412 **RATIONALE**

75413 The ISO C standard includes the text “The value of the file position indicator for the stream after  
 75414 reading or discarding all pushed-back wide characters shall be the same as it was before the  
 75415 wide characters were pushed back.” POSIX.1 omits “or discarding” from this because it is  
 75416 redundant—in the ISO C standard the discarding is done by file positioning functions and does  
 75417 not affect the position set by those functions. In particular, a relative seek using *fseek()* or *fseeko()*  
 75418 with SEEK\_CUR adjusts the position relative to the position on entry to the function, not the  
 75419 position after the pushed-back wide characters have been discarded. POSIX.1 also requires  
 75420 *fflush()* to discard pushed back wide characters in situations where the ISO C standard says the

- 75421 behavior of *fflush()* is undefined.
- 75422 **FUTURE DIRECTIONS**
- 75423 None.
- 75424 **SEE ALSO**
- 75425 [Section 2.5](#) (on page 521), *fseek()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*
- 75426 XBD [<stdio.h>](#), [<wchar.h>](#)
- 75427 **CHANGE HISTORY**
- 75428 First released in Issue 4. Derived from the MSE working draft.
- 75429 **Issue 5**
- 75430 The Optional Header (OH) marking is removed from [<stdio.h>](#).
- 75431 **Issue 6**
- 75432 The [EILSEQ] optional error condition is marked CX.
- 75433 **Issue 7**
- 75434 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0690 [87,93], XSH/TC1-2008/0691
- 75435 [87], and XSH/TC1-2008/0692 [14] are applied.
- 75436 **Issue 8**
- 75437 Austin Group Defect 701 is applied, clarifying how the file-position indicator for the stream is
- 75438 updated.
- 75439 Austin Group Defect 1374 is applied, correcting a conflict with the ISO C standard regarding the
- 75440 value of the file-position indicator for the stream after a successful call to *ungetwc()*.

75441 **NAME**

75442 unlink, unlinkat — remove a directory entry

75443 **SYNOPSIS**

75444 #include <unistd.h>

75445 int unlink(const char \*path);

75446 OH #include <fcntl.h>

75447 int unlinkat(int fd, const char \*path, int flag);

75448 **DESCRIPTION**

75449 The *unlink()* function shall remove the directory entry named by *path* and shall decrement the  
 75450 link count of the file referenced by the directory entry. If *path* names a symbolic link, *unlink()*  
 75451 shall remove the symbolic link and shall not affect any file named by the contents of the  
 75452 symbolic link.

75453 When the file’s link count becomes 0 and no process has a reference to the file via an open file  
 75454 descriptor or a memory mapping (see *mmap()*), the space occupied by the file shall be freed and  
 75455 the file shall no longer be accessible. If one or more processes have such a reference to the file  
 75456 when the last link is removed, the link shall be removed before *unlink()* returns, but the removal  
 75457 of the file contents shall be postponed until there are no such references to the file. When the  
 75458 space occupied by the file has been freed, the file’s serial number (*st\_ino*), and therefore the file  
 75459 identity (see XBD <*sys/stat.h*>), shall become available for reuse.

75460 The *path* argument shall not name a directory unless the process has appropriate privileges and  
 75461 the implementation supports using *unlink()* on directories.

75462 Upon successful completion, *unlink()* shall mark for update the last data modification and last  
 75463 file status change timestamps of the parent directory. Also, if the file’s link count is not 0, the last  
 75464 file status change timestamp of the file shall be marked for update.

75465 The *unlinkat()* function shall be equivalent to the *unlink()* or *rmdir()* function except in the case  
 75466 where *path* specifies a relative path. In this case the directory entry to be removed is determined  
 75467 relative to the directory associated with the file descriptor *fd* instead of the current working  
 75468 directory. If the access mode of the open file description associated with the file descriptor is not  
 75469 O\_SEARCH, the function shall check whether directory searches are permitted using the current  
 75470 permissions of the directory underlying the file descriptor. If the access mode is O\_SEARCH, the  
 75471 function shall not perform the check.

75472 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
 75473 in <*fcntl.h*>:

75474 AT\_REMOVEDIR

75475 Remove the directory entry specified by *fd* and *path* as a directory, not a normal file.

75476 If *unlinkat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
 75477 directory shall be used and the behavior shall be identical to a call to *unlink()* or *rmdir()*  
 75478 respectively, depending on whether or not the AT\_REMOVEDIR bit is set in *flag*.

75479 **RETURN VALUE**

75480 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
 75481 return -1 and set *errno* to indicate the error. If -1 is returned, the named file shall not be changed.

75482 **ERRORS**

75483 These functions shall fail and shall not unlink the file if:



75484	[EACCES]	Search permission is denied for a component of the path prefix, or write permission is denied on the directory containing the directory entry to be removed.
75485		
75486		
75487	[EBUSY]	The file named by the <i>path</i> argument cannot be unlinked because it is being used by the system or another process and the implementation considers this an error.
75488		
75489		
75490	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
75491		
75492	[ENAMETOOLONG]	
75493		The length of a component of a pathname is longer than {NAME_MAX}.
75494	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
75495	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>path</i> argument contains at least one non- <code>&lt;slash&gt;</code> character and ends with one or more trailing <code>&lt;slash&gt;</code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
75496		
75497		
75498		
75499		
75500	XSI [EPERM] or [EACCES]	
75501		The S_ISVTX flag is set on the directory containing the file referred to by the <i>path</i> argument and the process does not satisfy the criteria specified in XBD <a href="#">Section 4.5</a> (on page 96).
75502		
75503		
75504	[EROFS]	The directory entry to be unlinked is part of a read-only file system.
75505		The <i>unlink()</i> function shall fail and shall not unlink the file if:
75506	[EPERM]	The file named by <i>path</i> is a directory, and either the calling process does not have appropriate privileges or the implementation prohibits using <i>unlink()</i> on directories.
75507		
75508		
75509		The <i>unlinkat()</i> function shall fail and shall not unlink the file if:
75510	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
75511		
75512		
75513	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
75514		
75515	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
75516		
75517	[EEXIST] or [ENOTEMPTY]	
75518		The <i>flag</i> parameter has the AT_REMOVEDIR bit set and the <i>path</i> argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.
75519		
75520		
75521	[ENOTDIR]	The <i>flag</i> parameter has the AT_REMOVEDIR bit set and <i>path</i> does not name a directory.
75522		
75523	[EPERM]	The file named by <i>path</i> is a directory, the <i>flag</i> parameter does not have the AT_REMOVEDIR bit set, and either the calling process does not have appropriate privileges or the implementation prohibits using <i>unlink()</i> on directories.
75524		
75525		
75526		

- 75527 These functions may fail and not unlink the file if:
- 75528 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during
  - 75529 resolution of the *path* argument.
  - 75530 [ENAMETOOLONG]
  - 75531 The length of a pathname exceeds {PATH\_MAX}, or pathname resolution of a
  - 75532 symbolic link produced an intermediate result with a length that exceeds
  - 75533 {PATH\_MAX}.
  - 75534 [ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared
  - 75535 text) file that is being executed.

75536 The *unlinkat()* function may fail and not unlink the file if:

- 75537 [EINVAL] The value of the *flag* argument is not valid.

## 75538 EXAMPLES

### 75539 Removing a Link to a File

75540 The following example shows how to remove a link to a file named `/home/cnd/mod1` by

75541 removing the entry named `/modules/pass1`.

```
75542 #include <unistd.h>
75543 char *path = "/modules/pass1";
75544 int status;
75545 ...
75546 status = unlink(path);
```

### 75547 Checking for an Error

75548 The following example fragment creates a temporary password lock file named `LOCKFILE`,

75549 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for

75550 writing, *unlink()* is used to remove the link between the file descriptor and `LOCKFILE`.

```
75551 #include <sys/types.h>
75552 #include <stdio.h>
75553 #include <fcntl.h>
75554 #include <errno.h>
75555 #include <unistd.h>
75556 #include <sys/stat.h>
75557 #define LOCKFILE "/etc/ptmp"
75558 int pfd; /* Integer for file descriptor returned by open call. */
75559 FILE *fpfd; /* File pointer for use in putpwent(). */
75560 ...
75561 /* Open password Lock file. If it exists, this is an error. */
75562 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
75563 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
75564     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
75565     exit(1);
75566 }
75567 /* Lock file created; proceed with fdopen of lock file so that
75568 putpwent() can be used.
```

```

75569     */
75570     if ((fpfd = fdopen(pfd, "w")) == NULL) {
75571         close(pfd);
75572         unlink(LOCKFILE);
75573         exit(1);
75574     }

```

### 75575 Replacing Files

75576 The following example fragment uses *unlink()* to discard links to files, so that they can be  
75577 replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error  
75578 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can  
75579 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```

75580     #include <sys/types.h>
75581     #include <stdio.h>
75582     #include <fcntl.h>
75583     #include <errno.h>
75584     #include <unistd.h>
75585     #include <sys/stat.h>
75586
75587     #define LOCKFILE "/etc/ptmp"
75588     #define PASSWDFILE "/etc/passwd"
75589     #define SAVEFILE "/etc/opasswd"
75590     ...
75591     /* If no change was made, assume error and leave passwd unchanged. */
75592     if (!valid_change) {
75593         fprintf(stderr, "Could not change password for user %s\n", user);
75594         unlink(LOCKFILE);
75595         exit(1);
75596     }
75597
75598     /* Change permissions on new password file. */
75599     chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);
75600
75601     /* Remove saved password file. */
75602     unlink(SAVEFILE);
75603
75604     /* Save current password file. */
75605     link(PASSWDFILE, SAVEFILE);
75606
75607     /* Remove current password file. */
75608     unlink(PASSWDFILE);
75609
75610     /* Save new password file as current password file. */
75611     link(LOCKFILE, PASSWDFILE);
75612
75613     /* Remove lock file. */
75614     unlink(LOCKFILE);
75615
75616     exit(0);

```

### 75609 APPLICATION USAGE

75610 Applications should use *rmdir()* to remove a directory.

75611 **RATIONALE**

75612 Unlinking a directory is restricted to the superuser in many historical implementations for  
75613 reasons given in *link()* (see also *rename()*).

75614 The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume  
75615 of POSIX.1-2024 does not cover the system administration concepts of mounting and  
75616 unmounting, the description of the error was changed to “resource busy”. (This meaning is used  
75617 by some device drivers when a second process tries to open an exclusive use device.) The  
75618 wording is also intended to allow implementations to refuse to remove a directory if it is the root  
75619 or current working directory of any process.

75620 The standard developers reviewed TR 24715-2006 and noted that LSB-conforming  
75621 implementations may return [EISDIR] instead of [EPERM] when unlinking a directory. A change  
75622 to permit this behavior by changing the requirement for [EPERM] to [EPERM] or [EISDIR] was  
75623 considered, but decided against since it would break existing strictly conforming and  
75624 conforming applications. Applications written for portability to both POSIX.1-2024 and the LSB  
75625 should be prepared to handle either error code.

75626 The purpose of the *unlinkat()* function is to remove directory entries in directories other than the  
75627 current working directory without exposure to race conditions. Any part of the path of a file  
75628 could be changed in parallel to a call to *unlink()*, resulting in unspecified behavior. By opening a  
75629 file descriptor for the target directory and using the *unlinkat()* function it can be guaranteed that  
75630 the removed directory entry is located relative to the desired directory.

75631 **FUTURE DIRECTIONS**

75632 None.

75633 **SEE ALSO**

75634 *close()*, *link()*, *remove()*, *rename()*, *rmdir()*, *symlink()*

75635 XBD Section 4.5 (on page 96), [<fcntl.h>](#), [<sys/stat.h>](#), [<unistd.h>](#)

75636 **CHANGE HISTORY**

75637 First released in Issue 1. Derived from Issue 1 of the SVID.

75638 **Issue 5**

75639 The [EBUSY] error is added to the optional part of the ERRORS section.

75640 **Issue 6**

75641 The following new requirements on POSIX implementations derive from alignment with the  
75642 Single UNIX Specification:

- 75643 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 75644 • The [ELOOP] mandatory error condition is added.
- 75645 • A second [ENAMETOOLONG] is added as an optional error condition.
- 75646 • The [ETXTBSY] optional error condition is added.

75647 The following changes were made to align with the IEEE P1003.1a draft standard:

- 75648 • The [ELOOP] optional error condition is added.

75649 The normative text is updated to avoid use of the term “must” for application requirements.

75650 **Issue 7**

75651 Austin Group Interpretation 1003.1-2001 #143 is applied.

75652 Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for  
75653 operations when the S\_ISVTX bit is set.

- 75654 Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of [EPERM]  
75655 and [EISDIR].
- 75656 The *unlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API  
75657 Set Part 2.
- 75658 Changes are made related to support for finegrained timestamps.
- 75659 Changes are made to allow a directory to be opened for searching.
- 75660 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a  
75661 pathname exists but is not a directory or a symbolic link to a directory.
- 75662 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0693 [461], XSH/TC1-2008/0694 [324],  
75663 XSH/TC1-2008/0695 [278], and XSH/TC1-2008/0696 [278] are applied.
- 75664 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0378 [873], XSH/TC2-2008/0379 [591],  
75665 XSH/TC2-2008/0380 [817], and XSH/TC2-2008/0381 [817] are applied.
- 75666 **Issue 8**
- 75667 Austin Group Defect 1314 is applied, clarifying that file identities become available for reuse  
75668 after the space occupied by the file has been freed.
- 75669 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 75670 Austin Group Defect 1380 is applied, changing text using the term “link” in line with its  
75671 updated definition.
- 75672 Austin Group Defect 1385 is applied, clarifying that the file contents are not removed until there  
75673 are no references to the file via open file descriptors or memory mappings.
- 75674 Austin Group Defect 1574 is applied, splitting the [EPERM] error into separate entries for  
75675 *unlink()* and *unlinkat()*.

75676 **NAME**

75677 unlockpt — unlock a pseudo-terminal manager/subsidiary pair

75678 **SYNOPSIS**

```
75679 XSI #include <stdlib.h>  
75680 int unlockpt(int fildev);
```

75681 **DESCRIPTION**75682 The *unlockpt()* function shall unlock the subsidiary pseudo-terminal device associated with the  
75683 manager device to which *fildev* refers.75684 Conforming applications shall ensure that they call *unlockpt()* before opening the subsidiary side  
75685 of a pseudo-terminal device.75686 **RETURN VALUE**75687 Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno*  
75688 to indicate the error.75689 **ERRORS**75690 The *unlockpt()* function may fail if:75691 [EBADF] The *fildev* argument is not a file descriptor open for writing.75692 [EINVAL] The *fildev* argument is not associated with a manager pseudo-terminal device.75693 **EXAMPLES**

75694 None.

75695 **APPLICATION USAGE**

75696 None.

75697 **RATIONALE**75698 See the RATIONALE section for *posix\_openpt()*.75699 **FUTURE DIRECTIONS**

75700 None.

75701 **SEE ALSO**75702 *grantpt()*, *open()*, *posix\_openpt()*, *ptsname()*

75703 XBD &lt;stdlib.h&gt;

75704 **CHANGE HISTORY**

75705 First released in Issue 4, Version 2.

75706 **Issue 5**

75707 Moved from X/OPEN UNIX extension to BASE.

75708 **Issue 6**

75709 The normative text is updated to avoid use of the term “must” for application requirements.

75710 **Issue 7**

75711 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0697 [96] is applied.

75712 **Issue 8**75713 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal  
75714 devices.

75715 **NAME**

75716           unsetenv — remove an environment variable

75717 **SYNOPSIS**

```
75718 CX       #include <stdlib.h>
75719       int unsetenv(const char *name);
```

75720 **DESCRIPTION**

75721       The *unsetenv()* function shall remove an environment variable from the environment of the  
 75722       calling process. The *name* argument points to a string, which is the name of the variable to be  
 75723       removed. The named argument shall not contain an '=' character. If the named variable does  
 75724       not exist in the current environment, the environment shall be unchanged and the function is  
 75725       considered to have completed successfully.

75726       The *unsetenv()* function shall update the list of pointers to which *environ* points.

75727       The *unsetenv()* function need not be thread-safe.

75728 **RETURN VALUE**

75729       Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to  
 75730       indicate the error, and the environment shall be unchanged.

75731 **ERRORS**

75732       The *unsetenv()* function shall fail if:

75733	[EINVAL]	The <i>name</i> argument points to an empty string, or points to a string containing
75734		an '=' character.

75735 **EXAMPLES**

75736       None.

75737 **APPLICATION USAGE**

75738       None.

75739 **RATIONALE**

75740       Refer to the RATIONALE section in *setenv()*.

75741 **FUTURE DIRECTIONS**

75742       None.

75743 **SEE ALSO**

75744       *getenv()*, *setenv()*

75745       XBD <stdlib.h>, <sys/types.h>

75746 **CHANGE HISTORY**

75747       First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

75748 **Issue 7**

75749       Austin Group Interpretation 1003.1-2001 #156 is applied.

75750       POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0698 [167] and XSH/TC1-2008/0699  
 75751       [185] are applied.

75752 **NAME**

75753           uselocale — use locale in current thread

75754 **SYNOPSIS**

```
75755 CX       #include <locale.h>
75756       locale_t uselocale(locale_t newloc);
```

75757 **DESCRIPTION**75758       The *uselocale()* function shall set or query the current locale for the calling thread.75759       The value for the *newloc* argument shall be one of the following:

- 75760           1. A value returned by the *newlocale()* or *duplocale()* functions
- 75761           2. The special locale object descriptor LC\_GLOBAL\_LOCALE
- 75762           3. **(locale\_t)0**

75763       If the *newloc* argument is **(locale\_t)0**, the current locale shall not be changed; this value can be used to query the current locale setting. If the *newloc* argument is LC\_GLOBAL\_LOCALE, any thread-local locale for the calling thread shall be uninstalled; the thread shall again use the global locale as the current locale, and changes to the global locale shall affect the thread. Otherwise, the locale represented by *newloc* shall be installed as a thread-local locale to be used as the current locale for the calling thread.

75769       Once the *uselocale()* function has been called to install a thread-local locale, the behavior of every interface using data from the current locale shall be affected for the calling thread. The current locale for other threads shall remain unchanged.

75772 **RETURN VALUE**

75773       Upon successful completion, the *uselocale()* function shall return a handle for the thread-local locale that was in use as the current locale for the calling thread on entry to the function, or LC\_GLOBAL\_LOCALE if no thread-local locale was in use. Otherwise, *uselocale()* shall return **(locale\_t)0** and set *errno* to indicate the error.

75777 **ERRORS**75778       The *uselocale()* function may fail if:75779       [EINVAL]           *newloc* is not a valid locale object and is not **(locale\_t)0**.75780 **EXAMPLES**

75781       None.

75782 **APPLICATION USAGE**

75783       Unlike the *setlocale()* function, the *uselocale()* function does not allow replacing some locale categories only. Applications that need to install a locale which differs only in a few categories must use *newlocale()* to change a locale object equivalent to the currently used locale and install it.

75787 **RATIONALE**

75788       None.

75789 **FUTURE DIRECTIONS**

75790       None.



75791 **SEE ALSO**75792 *duplocale()*, *freelocale()*, *getlocalename\_1()*, *newlocale()*, *setlocale()*

75793 XBD &lt;locale.h&gt;

75794 **CHANGE HISTORY**

75795 First released in Issue 7.

75796 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0700 [290] and XSH/TC1-2008/0701  
75797 [334] are applied.

75798 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0382 [582] is applied.

75799 **Issue 8**75800 Austin Group Defect 1220 is applied, adding *getlocalename\_1()* to the SEE ALSO section.

75801 **NAME**

75802 utimensat, utimes — set file access and modification times

75803 **SYNOPSIS**

75804 #include <sys/stat.h>

75805 int utimensat(int *fd*, const char \**path*, const struct timespec *times*[2],  
75806 int *flag*);

75807 XSI #include <sys/time.h>

75808 int utimes(const char \**path*, const struct timeval *times*[2]);

75809 **DESCRIPTION**

75810 Refer to [futimens\(\)](#).

75811 **NAME**

75812       va\_arg, va\_copy, va\_end, va\_start — handle variable argument list

75813 **SYNOPSIS**

75814       #include &lt;stdarg.h&gt;

75815       type va\_arg(va\_list ap, type);

75816       void va\_copy(va\_list dest, va\_list src);

75817       void va\_end(va\_list ap);

75818       void va\_start(va\_list ap, argN);

75819 **DESCRIPTION**75820       Refer to XBD [<stdarg.h>](#)

75821 **NAME**

75822       vasprintf — format output of a stdarg argument list

75823 **SYNOPSIS**

75824       #include <stdarg.h>

75825       #include <stdio.h>

75826       int vasprintf(char \*\*restrict ptr, const char \*restrict format,

75827           va\_list ap);

75828 **DESCRIPTION**

75829       Refer to *vfprintf()*.

75830 **NAME**

75831 vasprintf, vdprintf, vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument  
75832 list

75833 **SYNOPSIS**

```
75834 #include <stdarg.h>
75835 #include <stdio.h>
```

```
75836 CX int vasprintf(char **restrict ptr, const char *restrict format,
75837                va_list ap);
75838 int vdprintf(int fildes, const char *restrict format, va_list ap);
75839 int vfprintf(FILE *restrict stream, const char *restrict format,
75840             va_list ap);
75841 int vprintf(const char *restrict format, va_list ap);
75842 int vsnprintf(char *restrict s, size_t n, const char *restrict format,
75843              va_list ap);
75844 int vsprintf(char *restrict s, const char *restrict format, va_list ap);
```

75845 **DESCRIPTION**

75846 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
75847 conflict between the requirements described here and the ISO C standard is unintentional. This  
75848 volume of POSIX.1-2024 defers to the ISO C standard.

75849 CX The *vasprintf()*, *vdprintf()*, *vfprintf()*, *vprintf()*, *vsnprintf()*, and *vsprintf()* functions shall be  
75850 CX equivalent to the *asprintf()*, *dprintf()*, *fprintf()*, *printf()*, *snprintf()*, and *sprintf()* functions  
75851 respectively, except that instead of being called with a variable number of arguments, they are  
75852 called with an argument list as defined by **<stdarg.h>**.

75853 These functions shall not invoke the *va\_end* macro. As these functions invoke the *va\_arg* macro,  
75854 the value of *ap* after the return is unspecified.

75855 **RETURN VALUE**

75856 Refer to *fprintf()*.

75857 **ERRORS**

75858 Refer to *fprintf()*.

75859 **EXAMPLES**

75860 None.

75861 **APPLICATION USAGE**

75862 Applications using these functions should call *va\_end(ap)* afterwards to clean up.

75863 **RATIONALE**

75864 None.

75865 **FUTURE DIRECTIONS**

75866 None.

75867 **SEE ALSO**

75868 [Section 2.5](#) (on page 521), *fprintf()*

75869 XBD **<stdarg.h>**, **<stdio.h>**

75870 **CHANGE HISTORY**

75871 First released in Issue 1. Derived from Issue 1 of the SVID.

75872 **Issue 5**

75873 The *vsnprintf()* function is added.

75874 **Issue 6**

75875 The *vfprintf()*, *vprintf()*, *vsprintf()*, and *vsnprintf()* functions are updated for alignment with the  
75876 ISO/IEC 9899:1999 standard.

75877 **Issue 7**

75878 The *vdprintf()* function is added to complement the *dprintf()* function from The Open Group  
75879 Technical Standard, 2006, Extended API Set Part 1.

75880 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0703 [14] is applied.

75881 **Issue 8**

75882 Austin Group Defect 1496 is applied, adding the *vasprintf()* function.

75883 **NAME**75884 `vfscanf`, `vscanf`, `vsscanf` — format input of a `stdarg` argument list75885 **SYNOPSIS**75886 `#include <stdarg.h>`75887 `#include <stdio.h>`75888 `int vfscanf(FILE *restrict stream, const char *restrict format,`  
75889 `va_list arg);`75890 `int vscanf(const char *restrict format, va_list arg);`75891 `int vsscanf(const char *restrict s, const char *restrict format,`  
75892 `va_list arg);`75893 **DESCRIPTION**75894 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
75895 conflict between the requirements described here and the ISO C standard is unintentional. This  
75896 volume of POSIX.1-2024 defers to the ISO C standard.75897 The `vscanf()`, `vfscanf()`, and `vsscanf()` functions shall be equivalent to the `scanf()`, `fscanf()`, and  
75898 `sscanf()` functions, respectively, except that instead of being called with a variable number of  
75899 arguments, they are called with an argument list as defined in the `<stdarg.h>` header. These  
75900 functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro, the  
75901 value of `ap` after the return is unspecified.75902 **RETURN VALUE**75903 Refer to `fscanf()`.75904 **ERRORS**75905 Refer to `fscanf()`.75906 **EXAMPLES**

75907 None.

75908 **APPLICATION USAGE**75909 Applications using these functions should call `va_end(ap)` afterwards to clean up.75910 **RATIONALE**

75911 None.

75912 **FUTURE DIRECTIONS**

75913 None.

75914 **SEE ALSO**75915 [Section 2.5](#) (on page 521), `fscanf()`75916 XBD `<stdarg.h>`, `<stdio.h>`75917 **CHANGE HISTORY**

75918 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

75919 **Issue 7**

75920 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0704 [14] is applied.

75921 **NAME**

75922 vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

75923 **SYNOPSIS**

```
75924 #include <stdarg.h>
75925 #include <stdio.h>
75926 #include <wchar.h>

75927 int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
75928             va_list arg);
75929 int vswprintf(wchar_t *restrict ws, size_t n,
75930             const wchar_t *restrict format, va_list arg);
75931 int vwprintf(const wchar_t *restrict format, va_list arg);
```

75932 **DESCRIPTION**

75933 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 75934 conflict between the requirements described here and the ISO C standard is unintentional. This  
 75935 volume of POSIX.1-2024 defers to the ISO C standard.

75936 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be equivalent to *fwprintf()*, *swprintf()*,  
 75937 and *wprintf()* respectively, except that instead of being called with a variable number of  
 75938 arguments, they are called with an argument list as defined by **<stdarg.h>**.

75939 These functions shall not invoke the *va\_end* macro. However, as these functions do invoke the  
 75940 *va\_arg* macro, the value of *ap* after the return is unspecified.

75941 **RETURN VALUE**75942 Refer to *fwprintf()*.75943 **ERRORS**75944 Refer to *fwprintf()*.75945 **EXAMPLES**

75946 None.

75947 **APPLICATION USAGE**75948 Applications using these functions should call *va\_end(ap)* afterwards to clean up.75949 **RATIONALE**

75950 None.

75951 **FUTURE DIRECTIONS**

75952 None.

75953 **SEE ALSO**75954 [Section 2.5](#) (on page 521), *fwprintf()*75955 XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**75956 **CHANGE HISTORY**

75957 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 75958 (E).

75959 **Issue 6**

75960 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the  
 75961 ISO/IEC 9899:1999 standard.



75962 **Issue 7**  
75963

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0705 [14] is applied.

75964 **NAME**

75965 vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg argument list

75966 **SYNOPSIS**

```
75967 #include <stdarg.h>
75968 #include <stdio.h>
75969 #include <wchar.h>

75970 int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,
75971             va_list arg);
75972 int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
75973             va_list arg);
75974 int vwscanf(const wchar_t *restrict format, va_list arg);
```

75975 **DESCRIPTION**

75976 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 75977 conflict between the requirements described here and the ISO C standard is unintentional. This  
 75978 volume of POSIX.1-2024 defers to the ISO C standard.

75979 The *vfwscanf()*, *vswscanf()*, and *vwscanf()* functions shall be equivalent to the *fscanf()*,  
 75980 *swscanf()*, and *wscanf()* functions, respectively, except that instead of being called with a variable  
 75981 number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header.  
 75982 These functions shall not invoke the *va\_end* macro. As these functions invoke the *va\_arg* macro,  
 75983 the value of *ap* after the return is unspecified.

75984 **RETURN VALUE**75985 Refer to *fscanf()*.75986 **ERRORS**75987 Refer to *fscanf()*.75988 **EXAMPLES**

75989 None.

75990 **APPLICATION USAGE**75991 Applications using these functions should call *va\_end(ap)* afterwards to clean up.75992 **RATIONALE**

75993 None.

75994 **FUTURE DIRECTIONS**

75995 None.

75996 **SEE ALSO**75997 [Section 2.5](#) (on page 521), *fscanf()*75998 XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**75999 **CHANGE HISTORY**

76000 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

76001 **Issue 7**

76002 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0706 [14] is applied.

76003 **NAME**

76004 vprintf — format the output of a stdarg argument list

76005 **SYNOPSIS**

76006 #include &lt;stdarg.h&gt;

76007 #include &lt;stdio.h&gt;

76008 int vprintf(const char \*restrict *format*, va\_list *ap*);76009 **DESCRIPTION**76010 Refer to *vfprintf()*.

76011 **NAME**

76012           vscanf — format input of a stdarg argument list

76013 **SYNOPSIS**

76014           #include &lt;stdarg.h&gt;

76015           #include &lt;stdio.h&gt;

76016           int vscanf(const char \*restrict *format*, va\_list *arg*);76017 **DESCRIPTION**76018           Refer to *vfprintf()*.

76019 **NAME**

76020        vsprintf, vsprintf — format output of a stdarg argument list

76021 **SYNOPSIS**

76022        #include &lt;stdarg.h&gt;

76023        #include &lt;stdio.h&gt;

```
76024        int vsnprintf(char *restrict s, size_t n,
76025                    const char *restrict format, va_list ap);
76026        int vsprintf(char *restrict s, const char *restrict format,
76027                    va_list ap);
```

76028 **DESCRIPTION**76029        Refer to *fprintf()*.

76030 **NAME**

76031 vsscanf — format input of a stdarg argument list

76032 **SYNOPSIS**

76033 #include &lt;stdarg.h&gt;

76034 #include &lt;stdio.h&gt;

76035 int vsscanf(const char \*restrict *s*, const char \*restrict *format*,76036 va\_list *arg*);76037 **DESCRIPTION**76038 Refer to *vfscanf()*.

76039 **NAME**

76040 vswprintf — wide-character formatted output of a stdarg argument list

76041 **SYNOPSIS**

76042 #include &lt;stdarg.h&gt;

76043 #include &lt;stdio.h&gt;

76044 #include &lt;wchar.h&gt;

76045 int vswprintf(wchar\_t \*restrict *ws*, size\_t *n*,76046 const wchar\_t \*restrict *format*, va\_list *arg*);76047 **DESCRIPTION**76048 Refer to *vfwprintf()*.

76049 **NAME**

76050 vswscanf — wide-character formatted input of a stdarg argument list

76051 **SYNOPSIS**

76052 #include &lt;stdarg.h&gt;

76053 #include &lt;stdio.h&gt;

76054 #include &lt;wchar.h&gt;

76055 int vswscanf(const wchar\_t \*restrict *ws*, const wchar\_t \*restrict *format*,  
76056 va\_list *arg*);76057 **DESCRIPTION**76058 Refer to *vfwscanf()*.



76059 **NAME**

76060 vwprintf — wide-character formatted output of a stdarg argument list

76061 **SYNOPSIS**

76062 #include &lt;stdarg.h&gt;

76063 #include &lt;stdio.h&gt;

76064 #include &lt;wchar.h&gt;

76065 int vwprintf(const wchar\_t \*restrict *format*, va\_list *arg*);76066 **DESCRIPTION**76067 Refer to *vwprintf()*.

76068 **NAME**

76069       vwscanf — wide-character formatted input of a stdarg argument list

76070 **SYNOPSIS**

76071       #include &lt;stdarg.h&gt;

76072       #include &lt;stdio.h&gt;

76073       #include &lt;wchar.h&gt;

76074       int vwscanf(const wchar\_t \*restrict *format*, va\_list *arg*);76075 **DESCRIPTION**76076       Refer to [vwscanf\(\)](#).

76077 **NAME**

76078 wait, waitpid — wait for a child process to stop or terminate

76079 **SYNOPSIS**

76080 #include &lt;sys/wait.h&gt;

76081 pid\_t wait(int \*stat\_loc);

76082 pid\_t waitpid(pid\_t pid, int \*stat\_loc, int options);

76083 **DESCRIPTION**

76084 The *wait()* and *waitpid()* functions shall obtain status information (see [Section 2.12](#), on page 563) pertaining to one of the caller's child processes. The *wait()* function obtains status information for process termination from any child process. The *waitpid()* function obtains status information for process termination, and optionally process stop and/or continue, from a specified subset of the child processes.

76089 The *wait()* function shall cause the calling thread to become blocked until status information generated by child process termination is made available to the thread, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process, or an error occurs. If termination status information is available prior to the call to *wait()*, return shall be immediate. If termination status information is available for two or more child processes, the order in which their status is reported is unspecified.

76095 As described in [Section 2.12](#) (on page 563), the *wait()* and *waitpid()* functions consume the status information they obtain.

76097 The behavior when multiple threads are blocked in *wait()*, *waitid()*, or *waitpid()* is described in [Section 2.12](#) (on page 563).

76099 The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is **(pid\_t)-1** and the *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and *options* arguments.

76102 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()* function shall only return the status of a child process from this set:

- 76104 • If *pid* is equal to **(pid\_t)-1**, *status* is requested for any child process. In this respect, *waitpid()* is then equivalent to *wait()*.
- 76106 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is requested.
- 76108 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of the calling process.
- 76110 • If *pid* is less than **(pid\_t)-1**, *status* is requested for any child process whose process group ID is equal to the absolute value of *pid*.

76112 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the following flags, defined in the **<sys/wait.h>** header:

76114 XSI **WCONTINUED** The *waitpid()* function shall report the status of any continued child process specified by *pid* whose status has not been reported since it continued from a job control stop.

76117 **WNOHANG** The *waitpid()* function shall not suspend execution of the calling thread if *status* is not immediately available for one of the child processes specified by *pid*.

76120 WUNTRACED The status of any child processes specified by *pid* that are stopped, and whose  
 76121 status has not yet been reported since they stopped, shall also be reported to  
 76122 the requesting process.

76123 If *wait()* or *waitpid()* return because the status of a child process is available, these functions  
 76124 shall return a value equal to the process ID of the child process. In this case, if the value of the  
 76125 argument *stat\_loc* is not a null pointer, information shall be stored in the location pointed to by  
 76126 *stat\_loc*. The value stored at the location pointed to by *stat\_loc* shall be 0 if and only if the status  
 76127 returned is from a terminated child process that terminated by one of the following means:

- 76128 1. The process returned 0 from *main()*.
- 76129 2. The process called *\_exit()* or *exit()* with a *status* argument of 0.
- 76130 3. The process was terminated because the last thread in the process terminated.

76131 Regardless of its value, this information may be interpreted using the following macros, which  
 76132 are defined in **<sys/wait.h>** and evaluate to integral expressions; the *stat\_val* argument is the  
 76133 integer value pointed to by *stat\_loc*.

76134 WIFEXITED(*stat\_val*)  
 76135 Evaluates to a non-zero value if *status* was returned for a child process that terminated  
 76136 normally.

76137 WEXITSTATUS(*stat\_val*)  
 76138 If the value of WIFEXITED(*stat\_val*) is non-zero, this macro shall evaluate to the low-order 8  
 76139 bits of the *status* argument that the child process passed to *\_exit()* or *exit()*, or the value the  
 76140 child process returned from *main()*.

76141 WIFSIGNALED(*stat\_val*)  
 76142 Evaluates to a non-zero value if *status* was returned for a child process that terminated due  
 76143 to the receipt of a signal that was not caught (see **<signal.h>**).

76144 WCOREDUMP(*stat\_val*)  
 76145 If the value of WIFSIGNALED(*stat\_val*) is non-zero, this macro shall evaluate to a non-zero  
 76146 value if the creation of a core image of the terminated child was attempted.

76147 WTERMSIG(*stat\_val*)  
 76148 If the value of WIFSIGNALED(*stat\_val*) is non-zero, this macro shall evaluate to the number  
 76149 of the signal that caused the termination of the child process.

76150 WIFSTOPPED(*stat\_val*)  
 76151 Evaluates to a non-zero value if *status* was returned for a child process that stopped due to  
 76152 the receipt of a signal that was not caught (see **<signal.h>**).

76153 WSTOPSIG(*stat\_val*)  
 76154 If the value of WIFSTOPPED(*stat\_val*) is non-zero, this macro shall evaluate to the number  
 76155 of the signal that caused the child process to stop.

76156 XSI WIFCONTINUED(*stat\_val*)  
 76157 Evaluates to a non-zero value if *status* was returned for a child process that has continued  
 76158 from a job control stop.

76159 SPN It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes  
 76160 created by *posix\_spawn()* or *posix\_spawnnp()* can indicate a WIFSTOPPED(*stat\_val*) before  
 76161 subsequent calls to *wait()* or *waitpid()* indicate WIFEXITED(*stat\_val*) as the result of an error  
 76162 detected before the new process image starts executing.

76163 It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes

76164 created by `posix_spawn()` or `posix_spawnp()` can indicate a `WIFSIGNALED(stat_val)` if a signal is  
 76165 sent to the parent's process group after `posix_spawn()` or `posix_spawnp()` is called.

76166 If the information pointed to by `stat_loc` was stored by a call to `waitpid()` that specified the  
 76167 XSI `WUNTRACED` flag and did not specify the `WCONTINUED` flag, exactly one of the macros  
 76168 `WIFEXITED(*stat_loc)`, `WIFSIGNALED(*stat_loc)`, and `WIFSTOPPED(*stat_loc)` shall evaluate to a  
 76169 non-zero value.

76170 XSI If the information pointed to by `stat_loc` was stored by a call to `waitpid()` that specified the  
 76171 `WUNTRACED` and `WCONTINUED` flags, exactly one of the macros `WIFEXITED(*stat_loc)`,  
 76172 `WIFSIGNALED(*stat_loc)`, `WIFSTOPPED(*stat_loc)`, and `WIFCONTINUED(*stat_loc)` shall  
 76173 evaluate to a non-zero value.

76174 If the information pointed to by `stat_loc` was stored by a call to `waitpid()` that did not specify the  
 76175 XSI `WUNTRACED` or `WCONTINUED` flags, or by a call to the `wait()` function, exactly one of the  
 76176 macros `WIFEXITED(*stat_loc)` and `WIFSIGNALED(*stat_loc)` shall evaluate to a non-zero value.

76177 XSI If the information pointed to by `stat_loc` was stored by a call to `waitpid()` that did not specify the  
 76178 `WUNTRACED` flag and specified the `WCONTINUED` flag, exactly one of the macros  
 76179 `WIFEXITED(*stat_loc)`, `WIFSIGNALED(*stat_loc)`, and `WIFCONTINUED(*stat_loc)` shall evaluate  
 76180 to a non-zero value.

76181 If the implementation queues the `SIGCHLD` signal, then if `wait()` or `waitpid()` returns because  
 76182 the status of a child process is available, any pending `SIGCHLD` signal associated with the  
 76183 process ID of the child process shall be discarded. Any other pending `SIGCHLD` signals shall  
 76184 remain pending.

76185 Otherwise, if `SIGCHLD` is blocked, if `wait()` or `waitpid()` return because the status of a child  
 76186 process is available, any pending `SIGCHLD` signal shall be cleared unless the status of another  
 76187 child process is available.

76188 For all other conditions, it is unspecified whether child *status* will be available when a `SIGCHLD`  
 76189 signal is delivered.

76190 There may be additional implementation-defined circumstances under which `wait()` or `waitpid()`  
 76191 report *status*. This shall not occur unless the calling process or one of its child processes  
 76192 explicitly makes use of a non-standard extension. In these cases the interpretation of the  
 76193 reported *status* is implementation-defined.

76194 If a parent process terminates without waiting for all of its child processes to terminate, the  
 76195 remaining child processes shall be assigned a new parent process ID corresponding to an  
 76196 implementation-defined system process.

#### 76197 RETURN VALUE

76198 If `wait()` or `waitpid()` returns because the status of a child process is available, these functions  
 76199 shall return a value equal to the process ID of the child process for which *status* is reported. If  
 76200 `wait()` or `waitpid()` returns due to the delivery of a signal to the calling process, `-1` shall be  
 76201 returned and `errno` set to `[EINTR]`. If `waitpid()` was invoked with `WNOHANG` set in *options*, it  
 76202 has at least one child process specified by *pid* for which *status* is not available, and *status* is not  
 76203 available for any process specified by *pid*, `0` is returned. Otherwise, `-1` shall be returned, and  
 76204 `errno` set to indicate the error.

#### 76205 ERRORS

76206 The `wait()` function shall fail if:

76207 `[ECHILD]` The calling process has no existing unwaited-for child processes.

76208	[EINTR]	The function was interrupted by a signal. The value of the location pointed to by <i>stat_loc</i> is undefined.
76209		
76210		The <i>waitpid()</i> function shall fail if:
76211	[ECHILD]	The process specified by <i>pid</i> does not exist or is not a child of the calling process, or the process group specified by <i>pid</i> does not exist or does not have any member process that is a child of the calling process.
76212		
76213		
76214	[EINTR]	The function was interrupted by a signal. The value of the location pointed to by <i>stat_loc</i> is undefined.
76215		
76216	[EINVAL]	The <i>options</i> argument is not valid.

## 76217 EXAMPLES

### 76218 Waiting for a Child Process and then Checking its Status

76219 The following example demonstrates the use of *waitpid()*, *fork()*, and the macros used to  
 76220 interpret the status value returned by *waitpid()* (and *wait()*). The code segment creates a child  
 76221 process which does some unspecified work. Meanwhile the parent loops performing calls to  
 76222 *waitpid()* to monitor the status of the child. The loop terminates when child termination is  
 76223 detected.

```

76224 #include <stdio.h>
76225 #include <stdlib.h>
76226 #include <unistd.h>
76227 #include <sys/wait.h>
76228 ...
76229 pid_t child_pid, wpid;
76230 int status;
76231 child_pid = fork();
76232 if (child_pid == -1) {          /* fork() failed */
76233     perror("fork");
76234     exit(EXIT_FAILURE);
76235 }
76236 if (child_pid == 0) {          /* This is the child */
76237     /* Child does some work and then terminates */
76238     ...
76239 } else {                       /* This is the parent */
76240     do {
76241         wpid = waitpid(child_pid, &status, WUNTRACED
76242 #ifdef WCONTINUED             /* Not all implementations support this */
76243     | WCONTINUED
76244 #endif
76245     );
76246         if (wpid == -1) {
76247             perror("waitpid");
76248             exit(EXIT_FAILURE);
76249         }
76250         if (WIFEXITED(status)) {
76251             printf("child exited, status=%d\n", WEXITSTATUS(status));
76252         } else if (WIFSIGNALED(status)) {

```

```

76253         printf("child killed (signal %d)\n", WTERMSIG(status));
76254     } else if (WIFSTOPPED(status)) {
76255         printf("child stopped (signal %d)\n", WSTOPSIG(status));
76256 #ifdef WIFCONTINUED     /* Not all implementations support this */
76257     } else if (WIFCONTINUED(status)) {
76258         printf("child continued\n");
76259 #endif
76260     } else {     /* Non-standard case -- may never happen */
76261         printf("Unexpected status (0x%x)\n", status);
76262     }
76263     } while (!WIFEXITED(status) && !WIFSIGNALED(status));
76264 }

```

### 76265 **Waiting for a Child Process in a Signal Handler for SIGCHLD**

76266 The following example demonstrates how to use *waitpid()* in a signal handler for SIGCHLD  
76267 without passing *-1* as the *pid* argument. (See the APPLICATION USAGE section below for the  
76268 reasons why passing a *pid* of *-1* is not recommended.) The method used here relies on the  
76269 standard behavior of *waitpid()* when SIGCHLD is blocked. On historical non-conforming  
76270 systems, the status of some child processes might not be reported.

```

76271 #include <stdlib.h>
76272 #include <stdio.h>
76273 #include <signal.h>
76274 #include <sys/types.h>
76275 #include <sys/wait.h>
76276 #include <unistd.h>
76277 #define CHILDREN 10
76278 static void
76279 handle_sigchld(int signum, siginfo_t *sinfo, void *unused)
76280 {
76281     int sav_errno = errno;
76282     int status;
76283     /*
76284     * Obtain status information for the child which
76285     * caused the SIGCHLD signal and write its exit code
76286     * to stdout.
76287     */
76288     if (sinfo->si_code != CLD_EXITED)
76289     {
76290         static char msg[] = "wrong si_code\n";
76291         write(2, msg, sizeof msg - 1);
76292     }
76293     else if (waitpid(sinfo->si_pid, &status, 0) == -1)
76294     {
76295         static char msg[] = "waitpid() failed\n";
76296         write(2, msg, sizeof msg - 1);
76297     }
76298     else if (!WIFEXITED(status))
76299     {

```

```
76300         static char msg[] = "WIFEXITED was false\n";
76301         write(2, msg, sizeof msg - 1);
76302     }
76303     else
76304     {
76305         int code = WEXITSTATUS(status);
76306         char buf[2];
76307         buf[0] = '0' + code;
76308         buf[1] = '\n';
76309         write(1, buf, 2);
76310     }
76311     errno = sav_errno;
76312 }

76313 int
76314 main(void)
76315 {
76316     int i;
76317     pid_t pid;
76318     struct sigaction sa;

76319     sa.sa_flags = SA_SIGINFO;
76320     sa.sa_sigaction = handle_sigchld;
76321     sigemptyset(&sa.sa_mask);
76322     if (sigaction(SIGCHLD, &sa, NULL) == -1)
76323     {
76324         perror("sigaction");
76325         exit(EXIT_FAILURE);
76326     }
76327     for (i = 0; i < CHILDREN; i++)
76328     {
76329         switch (pid = fork())
76330         {
76331             case -1:
76332                 perror("fork");
76333                 exit(EXIT_FAILURE);
76334             case 0:
76335                 sleep(2);
76336                 _exit(i);
76337         }
76338     }

76339     /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */
76340     alarm(3);
76341     for (;;)
76342         pause();

76343     return 0; /* NOTREACHED */
76344 }
```



76345 **APPLICATION USAGE**

76346 Calls to *wait()* will collect information about any child process. This may result in interactions  
 76347 with other interfaces that may be waiting for their own children (such as by use of *system()*). For  
 76348 this and other reasons it is recommended that portable applications not use *wait()*, but instead  
 76349 use *waitpid()*. For these same reasons, the use of *waitpid()* with a *pid* argument of *-1*, and the use  
 76350 of *waitid()* with the *idtype* argument set to *P\_ALL*, are also not recommended for portable  
 76351 applications.

76352 XSI As specified in [Consequences of Process Termination](#) (on page 568), if the calling process has  
 76353 *SA\_NOCLDWAIT* set or has *SIGCHLD* set to *SIG\_IGN*, then the termination of a child process  
 76354 will not cause status information to become available to a thread blocked in *wait()*, *waitid()*, or  
 76355 *waitpid()*. Thus, a thread blocked in one of the wait functions will remain blocked unless some  
 76356 other condition causes the thread to resume execution (such as an [ECHILD] failure due to no  
 76357 remaining children in the set of waited-for children).

76358 **RATIONALE**

76359 A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the  
 76360 calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an  
 76361 *exec* or other function calls) from the parent. If a child produces grandchildren by further use of  
 76362 *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()*  
 76363 from the original parent process. Nothing in this volume of POSIX.1-2024 prevents an  
 76364 implementation from providing extensions that permit a process to get *status* from a grandchild  
 76365 or any other process, but a process that does not use such extensions must be guaranteed to see  
 76366 *status* from only its direct children.

76367 The *waitpid()* function is provided for three reasons:

- 76368 1. To support job control
- 76369 2. To permit a non-blocking version of the *wait()* function
- 76370 3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without  
 76371 interfering with other terminated children for which the process has not waited

76372 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The  
 76373 function uses the *options* argument, which is equivalent to an argument to *wait3()*. The  
 76374 *WUNTRACED* flag is used only in conjunction with job control on systems supporting job  
 76375 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped  
 76376 processes in that implementation: processes being traced via the *ptrace()* debugging facility and  
 76377 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of  
 76378 POSIX.1-2024, only the second type is relevant. The name *WUNTRACED* was retained because  
 76379 its usage is the same, even though the name is not intuitively meaningful in this context.

76380 The third reason for the *waitpid()* function is to permit independent sections of a process to  
 76381 spawn and wait for children without interfering with each other. For example, the following  
 76382 problem occurs in developing a portable shell, or command interpreter:

```
76383 stream = popen("/bin/true");
76384 (void) system("sleep 100");
76385 (void) pclose(stream);
```

76386 On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

76387 The status values are retrieved by macros, rather than given as specific bit encodings as they are  
 76388 in most historical implementations (and thus expected by existing programs). This was  
 76389 necessary to eliminate a limitation on the number of signals an implementation can support that  
 76390 was inherent in the traditional encodings. This volume of POSIX.1-2024 does require that a *status*

76391 value of zero corresponds to a process calling `_exit(0)`, as this is the most common encoding  
76392 expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

76393 These macros syntactically operate on an arbitrary integer value. The behavior is undefined  
76394 unless that value is one stored by a successful call to `wait()` or `waitpid()` in the location pointed to  
76395 by the `stat_loc` argument. An early proposal attempted to make this clearer by specifying each  
76396 argument as `*stat_loc` rather than `stat_val`. However, that did not follow the conventions of other  
76397 specifications in this volume of POSIX.1-2024 or traditional usage. It also could have implied  
76398 that the argument to the macro must literally be `*stat_loc`; in fact, that value can be stored or  
76399 passed as an argument to other functions before being interpreted by these macros.

76400 The extension that affects `wait()` and `waitpid()` and is common in historical implementations is  
76401 the `ptrace()` function. It is called by a child process and causes that child to stop and return a  
76402 `status` that appears identical to the `status` indicated by WIFSTOPPED. The `status` of `ptrace()`  
76403 children is traditionally returned regardless of the WUNTRACED flag (or by the `wait()`  
76404 function). Most applications do not need to concern themselves with such extensions because  
76405 they have control over what extensions they or their children use. However, applications, such  
76406 as command interpreters, that invoke arbitrary processes may see this behavior when those  
76407 arbitrary processes misuse such extensions.

76408 On implementations that support the creation of a file containing a core image on some process  
76409 terminations, the `WCOREDUMP(stat_val)` macro indicates whether creation of a core image was  
76410 attempted. If it returns a non-zero value this does not necessarily mean that the core image was  
76411 created, only that it was attempted. For example, if the `RLIMIT_CORE` limit for the process is 0,  
76412 this prevents creation of the file; `WCOREDUMP(stat_val)` returning non-zero in this case  
76413 indicates that the file would have been created if the limit had not been 0.

76414 Allowing the `wait()` family of functions to discard a pending SIGCHLD signal that is associated  
76415 with a successfully waited-for child process puts them into the `sigwait()` and `sigwaitinfo()`  
76416 category with respect to SIGCHLD.

76417 This definition allows implementations to treat a pending SIGCHLD signal as accepted by the  
76418 process in `wait()`, with the same meaning of “accepted” as when that word is applied to the  
76419 `sigwait()` family of functions.

76420 Allowing the `wait()` family of functions to behave this way permits an implementation to be able  
76421 to deal precisely with SIGCHLD signals.

76422 In particular, an implementation that does accept (discard) the SIGCHLD signal can make the  
76423 following guarantees regardless of the queuing depth of signals in general (the list of waitable  
76424 children can hold the SIGCHLD queue):

- 76425 1. If a SIGCHLD signal handler is established via `sigaction()` without the `SA_RESETHAND`  
76426 flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal  
76427 will be delivered to or accepted by the process for every child process that terminates.
- 76428 2. A single `wait()` issued from a SIGCHLD signal handler can be guaranteed to return  
76429 immediately with status information for a child process.
- 76430 3. When `SA_SIGINFO` is requested, the SIGCHLD signal handler can be guaranteed to  
76431 receive a non-null pointer to a `siginfo_t` structure that describes a child process for which  
76432 a wait via `waitpid()` or `waitid()` will not block or fail.
- 76433 4. The `system()` function will not cause the SIGCHLD handler of a process to be called as a  
76434 result of the `fork()/exec` executed within `system()` because `system()` will accept the  
76435 SIGCHLD signal when it performs a `waitpid()` for its child process. This is a desirable  
76436 behavior of `system()` so that it can be used in a library without causing side-effects to the

76437 application linked with the library.

76438 An implementation that does not permit the *wait()* family of functions to accept (discard) a  
76439 pending SIGCHLD signal associated with a successfully waited-for child, cannot make the  
76440 guarantees described above for the following reasons:

76441 **Guarantee #1**

76442 Although it might be assumed that reliable queuing of all SIGCHLD signals generated by  
76443 the system can make this guarantee, the counter-example is the case of a process that blocks  
76444 SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the  
76445 implementation supports queued signals, then eventually the system will run out of  
76446 memory for the queue. The guarantee cannot be made because there must be some limit to  
76447 the depth of queuing.

76448 **Guarantees #2 and #3**

76449 These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD  
76450 signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()*  
76451 function) will result in an invocation of the handler when SIGCHLD is unblocked, after the  
76452 process has disappeared.

76453 **Guarantee #4**

76454 Although possible to make this guarantee, *system()* would have to set the SIGCHLD  
76455 handler to SIG\_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded  
76456 (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This  
76457 would have the undesirable side-effect of discarding all SIGCHLD signals pending to the  
76458 process.

76459 **FUTURE DIRECTIONS**

76460 None.

76461 **SEE ALSO**

76462 [Section 2.12](#) (on page 563), *exec*, *exit()*, *fork()*, *system()*, *waitid()*

76463 [XBD Section 4.15.2](#) (on page 104), [<signal.h>](#), [<sys/wait.h>](#)

76464 **CHANGE HISTORY**

76465 First released in Issue 1. Derived from Issue 1 of the SVID.

76466 **Issue 5**

76467 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

76468 **Issue 6**

76469 The following new requirements on POSIX implementations derive from alignment with the  
76470 Single UNIX Specification:

- 76471 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
76472 required for conforming implementations of previous POSIX specifications, it was not  
76473 required for UNIX applications.

76474 The following changes were made to align with the IEEE P1003.1a draft standard:

- 76475 • The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

76476 The semantics of *WIFSTOPPED(stat\_val)*, *WIFEXITED(stat\_val)*, and *WIFSIGNALED(stat\_val)*  
76477 are defined with respect to *posix\_spawn()* or *posix\_spawnnp()* for alignment with IEEE Std  
76478 1003.1d-1999.

76479 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

76480 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/145 is applied, adding the example to the

- 76481           EXAMPLES section.
- 76482 **Issue 7**
- 76483           SD5-XSH-ERN-202 is applied.
- 76484           APPLICATION USAGE is added, recommending that the *wait()* function not be used.
- 76485           An additional example for *waitpid()* is added.
- 76486           POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0707 [421], XSH/TC1-2008/0708 [166],  
76487           XSH/TC1-2008/0709 [166], and XSH/TC1-2008/0710 [69] are applied.
- 76488           POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0384 [690], XSH/TC2-2008/0385 [691],  
76489           and XSH/TC2-2008/0386 [690] are applied.
- 76490 **Issue 8**
- 76491           Austin Group Defect 1116 is applied, removing text related to the Realtime Signals Extension  
76492           option that existed in earlier versions of this standard.
- 76493           Austin Group Defects 1141 and 1363 are applied, adding WCOREDUMP, changing the  
76494           description of WIFSTOPPED, and changing the RATIONALE section.
- 76495           Austin Group Defect 1570 is applied, removing extra spacing in "==".

76496 **NAME**76497 `waitid` — wait for a child process to change state76498 **SYNOPSIS**76499 `#include <sys/wait.h>`76500 `int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);`76501 **DESCRIPTION**76502 The `waitid()` function shall obtain status information (see [Section 2.12](#), on page 563) pertaining to  
76503 termination, stop, and/or continue events in one of the caller's child processes.76504 The `waitid()` function shall cause the calling thread to become blocked until an error occurs or  
76505 status information becomes available to the calling thread that satisfies all of the following  
76506 properties ("matching status information"):

- 76507 • The status information is from one of the child processes in the set of child processes  
76508 specified by the `idtype` and `id` arguments.
- 76509 • The state change in the status information matches one of the state change flags set in the  
76510 `options` argument.

76511 If matching status information is available prior to the call to `waitid()`, return shall be immediate.  
76512 If matching status information is available for two or more child processes, the order in which  
76513 their status is reported is unspecified.76514 As described in [Section 2.12](#) (on page 563), the `waitid()` function consumes the status information  
76515 it obtains unless the `WNOWAIT` flag is set in the `options` argument.76516 The behavior when multiple threads are blocked in `wait()`, `waitid()`, or `waitpid()` is described in  
76517 [Section 2.12](#) (on page 563).76518 The `waitid()` function shall record the obtained status information in the structure pointed to by  
76519 `infop`. The fields of the structure pointed to by `infop` shall be filled in as described under "Pointer  
76520 to a Function" in [Section 2.4.3](#) (on page 516).76521 The `idtype` and `id` arguments are used to specify which children `waitid()` waits for.76522 If `idtype` is `P_PID`, `waitid()` shall wait for the child with a process ID equal to `(pid_t)id`.76523 If `idtype` is `P_PGID`, `waitid()` shall wait for any child with a process group ID equal to `(pid_t)id`.76524 If `idtype` is `P_ALL`, `waitid()` shall wait for any children and `id` is ignored.76525 The `options` argument is used to specify which state changes `waitid()` shall wait for. It is formed  
76526 by OR'ing together the following flags:76527 **WCONTINUED** Status shall be returned for any continued child process whose status either  
76528 has not been reported since it continued from a job control stop or has been  
76529 reported only by calls to `waitid()` with the `WNOWAIT` flag set.76530 **WEXITED** Wait for processes that have terminated.76531 **WNOHANG** Do not hang if no status is available; return immediately.76532 **WNOWAIT** Keep the process whose status is returned in `infop` in a waitable state. This  
76533 shall not affect the state of the process; the process may be waited for again  
76534 after this call completes.76535 **WSTOPPED** Status shall be returned for any child that has stopped upon receipt of a signal,  
76536 and whose status either has not been reported since it stopped or has been  
76537 reported only by calls to `waitid()` with the `WNOWAIT` flag set.

76538 Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to  
76539 be OR'ed in with the *options* argument.

76540 The application shall ensure that the *infop* argument points to a **siginfo\_t** structure. If *waitid()*  
76541 returns because a child process was found that satisfied the conditions indicated by the  
76542 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the  
76543 system with the status of the process; the *si\_signo* member shall be set equal to SIGCHLD. If  
76544 *waitid()* returns because WNOHANG was specified and status is not available for any process  
76545 specified by *idtype* and *id*, then the *si\_signo* and *si\_pid* members of the structure pointed to by  
76546 *infop* shall be set to zero and the values of other members of the structure are unspecified.

#### 76547 RETURN VALUE

76548 If WNOHANG was specified and status is not available for any process specified by *idtype* and  
76549 *id*, 0 shall be returned. If *waitid()* returns due to the change of state of one of its children, 0 shall  
76550 be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

#### 76551 ERRORS

76552 The *waitid()* function shall fail if:

- |       |          |  |
|-------|----------|--|
| 76553 | [ECHILD] | The calling process has no existing unwaited-for child processes.  |
| 76554 | [EINTR]  | The <i>waitid()</i> function was interrupted by a signal. The values of the fields of<br>76555 the structure pointed to by <i>infop</i> are undefined. |
| 76556 | [EINVAL] | An invalid value was specified for <i>options</i> , or <i>idtype</i> and <i>id</i> specify an invalid<br>76557 set of processes.                       |

#### 76558 EXAMPLES

76559 None.

#### 76560 APPLICATION USAGE

76561 Calls to *waitid()* with *idtype* equal to P\_ALL will collect information about any child process.  
76562 This may result in interactions with other interfaces that may be waiting for their own children  
76563 (such as by use of *system()*). For this reason it is recommended that portable applications not  
76564 use *waitid()* with *idtype* of P\_ALL. See also APPLICATION USAGE for *wait()*.

76565 XSI As specified in [Consequences of Process Termination](#) (on page 568), if the calling process has  
76566 SA\_NOCLDWAIT set or has SIGCHLD set to SIG\_IGN, then the termination of a child process  
76567 will not cause status information to become available to a thread blocked in *wait()*, *waitid()*, or  
76568 *waitpid()*. Thus, a thread blocked in one of the wait functions will remain blocked unless some  
76569 other condition causes the thread to resume execution (such as an [ECHILD] failure due to no  
76570 remaining children in the set of waited-for children).

#### 76571 RATIONALE

76572 None.

#### 76573 FUTURE DIRECTIONS

76574 None.

#### 76575 SEE ALSO

76576 [Section 2.4.3](#) (on page 516), [Section 2.12](#) (on page 563), *exec*, *exit()*, *wait()*

76577 XBD [<signal.h>](#), [<sys/wait.h>](#)

#### 76578 CHANGE HISTORY

76579 First released in Issue 4, Version 2.

76580 **Issue 5**

76581 Moved from X/OPEN UNIX extension to BASE.

76582 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

76583 **Issue 6**

76584 The normative text is updated to avoid use of the term “must” for application requirements.

76585 **Issue 7**

76586 Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION.

76587 The *waitid()* function is moved from the XSI option to the Base.

76588 APPLICATION USAGE is added, recommending that the *waitid()* function not be used with *idtype* equal to P\_ALL.

76590 The description of the WNOHANG flag is updated.

76591 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0711 [154], XSH/TC1-2008/0712 [154], and XSH/TC1-2008/0713 [153] are applied.

76593 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0387 [690] is applied.

76594 **Issue 8**

76595 Austin Group Defect 1332 is applied, changing the description of WEXITED.

76596 Austin Group Defect 1547 is applied, changing the description of [EINTR].

76597 **NAME**

76598           waitpid — wait for a child process to stop or terminate

76599 **SYNOPSIS**

76600           #include <sys/wait.h>

76601           pid\_t waitpid(pid\_t *pid*, int \**stat\_loc*, int *options*);

76602 **DESCRIPTION**

76603           Refer to *wait()*.



76604 **NAME**

76605           wcpcpy — copy a wide-character string, returning a pointer to its end

76606 **SYNOPSIS**

```
76607 CX       #include <wchar.h>  
76608       wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

76609 **DESCRIPTION**76610       Refer to *wcscopy()*.

76611 **NAME**

76612           wcpncpy — copy a fixed-size wide-character string, returning a pointer to its end

76613 **SYNOPSIS**

```
76614 CX       #include <wchar.h>
76615       wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
76616                       size_t n);
```

76617 **DESCRIPTION**76618       Refer to *wcsncpy()*.

76619 **NAME**76620 `wrtomb` — convert a wide-character code to a character (restartable)76621 **SYNOPSIS**76622 `#include <wchar.h>`76623 `size_t wrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);`76624 **DESCRIPTION**76625 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
76626 conflict between the requirements described here and the ISO C standard is unintentional. This  
76627 volume of POSIX.1-2024 defers to the ISO C standard.76628 If *s* is a null pointer, the `wrtomb()` function shall be equivalent to the call:76629 `wrtomb(buf, L'\0', ps)`76630 where *buf* is an internal buffer.76631 If *s* is not a null pointer, the `wrtomb()` function shall determine the number of bytes needed to  
76632 represent the character that corresponds to the wide character given by *wc* (including any shift  
76633 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At  
76634 most {MB\_CUR\_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored,  
76635 preceded by any shift sequence needed to restore the initial shift state. The resulting state  
76636 described shall be the initial conversion state.76637 If *ps* is a null pointer, the `wrtomb()` function shall use its own internal **mbstate\_t** object, which is  
76638 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate\_t** object  
76639 pointed to by *ps* shall be used to completely describe the current conversion state of the  
76640 associated character sequence. The implementation shall behave as if no function defined in this  
76641 volume of POSIX.1-2024 calls `wrtomb()`.76642 If called with a null *ps* argument, the `wrtomb()` function need not be thread-safe; however, such  
76643 calls shall avoid data races with calls to `wrtomb()` with a non-null argument and with calls to all  
76644 other functions.76645 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.76646 The `wrtomb()` function shall not change the setting of *errno* if successful.76647 **RETURN VALUE**76648 The `wrtomb()` function shall return the number of bytes stored in the array object (including any  
76649 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this  
76650 case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size\_t**)-1;  
76651 the conversion state shall be undefined.76652 **ERRORS**76653 The `wrtomb()` function shall fail if:

76654 [EILSEQ] An invalid wide-character code is detected.

76655 The `wrtomb()` function may fail if:76656 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

76657 **EXAMPLES**

76658 None.

76659 **APPLICATION USAGE**

76660 None.

76661 **RATIONALE**

76662 None.

76663 **FUTURE DIRECTIONS**

76664 None.

76665 **SEE ALSO**76666 *mbsinit()*, *wcsrtombs()*76667 XBD <**wchar.h**>76668 **CHANGE HISTORY**76669 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
76670 (E).76671 **Issue 6**

76672 In the DESCRIPTION, a note on using this function in a threaded application is added.

76673 Extensions beyond the ISO C standard are marked.

76674 The normative text is updated to avoid use of the term “must” for application requirements.

76675 The *wcr tomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.76676 **Issue 7**76677 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcr tomb()* function  
76678 need not be thread-safe if called with a NULL *ps* argument.

76679 Austin Group Interpretation 1003.1-2001 #170 is applied.

76680 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0714 [88] and XSH/TC1-2008/0715  
76681 [105] are applied.76682 **Issue 8**76683 Austin Group Defect 1302 is applied, aligning this function with the ISO/IEC 9899:2018  
76684 standard.

76685 **NAME**

76686 wscasecmp, wscasecmp\_l, wcsncasecmp, wcsncasecmp\_l — case-insensitive wide-character  
76687 string comparison

76688 **SYNOPSIS**

```
76689 CX #include <wchar.h>
76690 int wscasecmp(const wchar_t *ws1, const wchar_t *ws2);
76691 int wscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
76692 locale_t locale);
76693 int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
76694 int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
76695 size_t n, locale_t locale);
```

76696 **DESCRIPTION**

76697 The *wscasecmp()* and *wcsncasecmp()* functions are the wide-character equivalent of the  
76698 *strcasecmp()* and *strncasecmp()* functions, respectively.

76699 The *wscasecmp()* and *wscasecmp\_l()* functions shall compare, while ignoring differences in case,  
76700 the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

76701 The *wcsncasecmp()* and *wcsncasecmp\_l()* functions shall compare, while ignoring differences in  
76702 case, not more than *n* wide-characters from the wide-character string pointed to by *ws1* to the  
76703 wide-character string pointed to by *ws2*.

76704 The *wscasecmp()* and *wcsncasecmp()* functions use the current locale to determine the case of the  
76705 wide characters.

76706 The *wscasecmp\_l()* and *wcsncasecmp\_l()* functions use the locale represented by *locale* to  
76707 determine the case of the wide characters.

76708 When the *LC\_CTYPE* category of the locale being used is from the POSIX locale, these functions  
76709 shall behave as if the wide-character strings had been converted to lowercase and then a  
76710 comparison of wide-character codes performed. Otherwise, the results are unspecified.

76711 The information for *wscasecmp\_l()* and *wcsncasecmp\_l()* about the case of the characters comes  
76712 from the locale represented by *locale*.

76713 The behavior is undefined if the *locale* argument to *wscasecmp\_l()* or *wcsncasecmp\_l()* is the  
76714 special locale object *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.

76715 **RETURN VALUE**

76716 Upon completion, the *wscasecmp()* and *wscasecmp\_l()* functions shall return an integer greater  
76717 than, equal to, or less than 0 if the wide-character string pointed to by *ws1* is, ignoring case,  
76718 greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

76719 Upon completion, the *wcsncasecmp()* and *wcsncasecmp\_l()* functions shall return an integer  
76720 greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed  
76721 to by *ws1* is, ignoring case, greater than, equal to, or less than the possibly null wide-character  
76722 terminated string pointed to by *ws2*, respectively.

76723 No return values are reserved to indicate an error.

76724 **ERRORS**

76725 No errors are defined.

76726 **EXAMPLES**

76727       None.

76728 **APPLICATION USAGE**

76729       None.

76730 **RATIONALE**

76731       None.

76732 **FUTURE DIRECTIONS**

76733       None.

76734 **SEE ALSO**76735       [strcasecmp\(\)](#), [wcscmp\(\)](#), [wcsncmp\(\)](#)76736       XBD <[wchar.h](#)>76737 **CHANGE HISTORY**

76738       First released in Issue 7.

76739       POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0716 [294], XSH/TC1-2008/0717 [283],  
76740       and XSH/TC1-2008/0718 [283] are applied.

76741 **NAME**

76742 wscat — concatenate two wide-character strings

76743 **SYNOPSIS**

76744 #include &lt;wchar.h&gt;

76745 wchar\_t \*wscat(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2);

76746 **DESCRIPTION**

76747 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
76748 conflict between the requirements described here and the ISO C standard is unintentional. This  
76749 volume of POSIX.1-2024 defers to the ISO C standard.

76750 The `wscat()` function shall append a copy of the wide-character string pointed to by `ws2`  
76751 (including the terminating null wide-character code) to the end of the wide-character string  
76752 pointed to by `ws1`. The initial wide-character code of `ws2` shall overwrite the null wide-character  
76753 code at the end of `ws1`. If copying takes place between objects that overlap, the behavior is  
76754 undefined.

76755 CX The `wscat()` function shall not change the setting of `errno` on valid input.

76756 **RETURN VALUE**76757 The `wscat()` function shall return `ws1`; no return value is reserved to indicate an error.76758 **ERRORS**

76759 No errors are defined.

76760 **EXAMPLES**

76761 None.

76762 **APPLICATION USAGE**

76763 None.

76764 **RATIONALE**

76765 None.

76766 **FUTURE DIRECTIONS**

76767 None.

76768 **SEE ALSO**76769 [wscncat\(\)](#)76770 XBD [<wchar.h>](#)76771 **CHANGE HISTORY**

76772 First released in Issue 4. Derived from the MSE working draft.

76773 **Issue 6**

76774 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, `s1` is  
76775 changed to `ws1`.

76776 The `wscat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

76777 **Issue 8**

76778 Austin Group Defect 448 is applied, adding a requirement that `wscat()` does not change the  
76779 setting of `errno` on valid input.

76780 **NAME**76781 `wcschr` — wide-character string scanning operation76782 **SYNOPSIS**76783 `#include <wchar.h>`76784 `wchar_t *wcschr(const wchar_t *ws, wchar_t wc);`76785 **DESCRIPTION**

76786 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
76787 conflict between the requirements described here and the ISO C standard is unintentional. This  
76788 volume of POSIX.1-2024 defers to the ISO C standard.

76789 The `wcschr()` function shall locate the first occurrence of `wc` in the wide-character string pointed  
76790 to by `ws`. The application shall ensure that the value of `wc` is a character representable as a type  
76791 `wchar_t` and a wide-character code corresponding to a valid character in the current locale. The  
76792 terminating null wide-character code is considered to be part of the wide-character string.

76793 CX The `wcschr()` function shall not change the setting of `errno` on valid input.

76794 **RETURN VALUE**

76795 Upon completion, `wcschr()` shall return a pointer to the wide-character code, or a null pointer if  
76796 the wide-character code is not found.

76797 **ERRORS**

76798 No errors are defined.

76799 **EXAMPLES**

76800 None.

76801 **APPLICATION USAGE**

76802 None.

76803 **RATIONALE**

76804 None.

76805 **FUTURE DIRECTIONS**

76806 None.

76807 **SEE ALSO**76808 [wcsrchr\(\)](#)76809 XBD [<wchar.h>](#)76810 **CHANGE HISTORY**

76811 First released in Issue 4. Derived from the MSE working draft.

76812 **Issue 6**

76813 The normative text is updated to avoid use of the term “must” for application requirements.

76814 **Issue 8**

76815 Austin Group Defect 448 is applied, adding a requirement that `wcschr()` does not change the  
76816 setting of `errno` on valid input.



76817 **NAME**

76818 wscmp — compare two wide-character strings

76819 **SYNOPSIS**

76820 #include &lt;wchar.h&gt;

76821 int wscmp(const wchar\_t \*ws1, const wchar\_t \*ws2);

76822 **DESCRIPTION**

76823 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
76824 conflict between the requirements described here and the ISO C standard is unintentional. This  
76825 volume of POSIX.1-2024 defers to the ISO C standard.

76826 The *wscmp()* function shall compare the wide-character string pointed to by *ws1* to the wide-  
76827 character string pointed to by *ws2*.

76828 The sign of a non-zero return value shall be determined by the sign of the difference between the  
76829 values of the first pair of wide-character codes that differ in the objects being compared.

76830 CX The *wscmp()* function shall not change the setting of *errno* on valid input.

76831 **RETURN VALUE**

76832 Upon completion, *wscmp()* shall return an integer greater than, equal to, or less than 0, if the  
76833 wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character  
76834 string pointed to by *ws2*, respectively.

76835 **ERRORS**

76836 No errors are defined.

76837 **EXAMPLES**

76838 None.

76839 **APPLICATION USAGE**

76840 None.

76841 **RATIONALE**

76842 None.

76843 **FUTURE DIRECTIONS**

76844 None.

76845 **SEE ALSO**76846 [wscasecmp\(\)](#), [wcsncmp\(\)](#)76847 XBD [<wchar.h>](#)76848 **CHANGE HISTORY**

76849 First released in Issue 4. Derived from the MSE working draft.

76850 **Issue 8**

76851 Austin Group Defect 448 is applied, adding a requirement that *wscmp()* does not change the  
76852 setting of *errno* on valid input.

76853 **NAME**

76854 wscoll, wscoll\_l — wide-character string comparison using collating information

76855 **SYNOPSIS**

```
76856 #include <wchar.h>
76857 int wscoll(const wchar_t *ws1, const wchar_t *ws2);
76858 CX int wscoll_l(const wchar_t *ws1, const wchar_t *ws2,
76859 locale_t locale);
```

76860 **DESCRIPTION**

76861 CX For `wscoll()`: The functionality described on this reference page is aligned with the ISO C  
 76862 standard. Any conflict between the requirements described here and the ISO C standard is  
 76863 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

76864 CX The `wscoll()` and `wscoll_l()` functions shall compare the wide-character string pointed to by  
 76865 `ws1` to the wide-character string pointed to by `ws2`, both interpreted as appropriate to the  
 76866 CX `LC_COLLATE` category of the current locale, or the locale represented by `locale`, respectively.

76867 CX The `wscoll()` and `wscoll_l()` functions shall not change the setting of `errno` if successful.

76868 CX An application wishing to check for error situations should set `errno` to 0 before calling `wscoll()`  
 76869 or `wscoll_l()`. If `errno` is non-zero on return, an error has occurred.

76870 CX The behavior is undefined if the `locale` argument to `wscoll_l()` is the special locale object  
 76871 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

76872 **RETURN VALUE**

76873 CX Upon successful completion, `wscoll()` and `wscoll_l()` shall return an integer greater than, equal  
 76874 to, or less than 0, according to whether the wide-character string pointed to by `ws1` is greater  
 76875 than, equal to, or less than the wide-character string pointed to by `ws2`, when both are  
 76876 CX interpreted as appropriate to the current locale, or to the locale represented by `locale`,  
 76877 CX respectively. On error, `wscoll()` and `wscoll_l()` shall set `errno`, but no return value is reserved  
 76878 to indicate an error.

76879 **ERRORS**

76880 These functions may fail if:

76881 CX [EINVAL] The `ws1` or `ws2` arguments contain wide-character codes outside the domain of  
 76882 the collating sequence.

76883 **EXAMPLES**

76884 None.

76885 **APPLICATION USAGE**

76886 The `wcsxfrm()` and `wscmp()` functions should be used for sorting large lists.

76887 **RATIONALE**

76888 None.

76889 **FUTURE DIRECTIONS**

76890 None.

76891 **SEE ALSO**

76892 [wscmp\(\)](#), [wcsxfrm\(\)](#)

76893 XBD [<wchar.h>](#)

76894 **CHANGE HISTORY**

76895 First released in Issue 4. Derived from the MSE working draft.

76896 **Issue 5**

76897 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

76898 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

76899 **Issue 7**

76900 The *wscoll\_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

76902 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0719 [302], XSH/TC1-2008/0720 [283],  
76903 and XSH/TC1-2008/0721 [283] are applied.

76904 **NAME**

76905 wpcpy, wscpy — copy a wide-character string, returning a pointer to its end

76906 **SYNOPSIS**

76907 #include &lt;wchar.h&gt;

76908 CX `wchar_t *wpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`76909 `wchar_t *wscpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`76910 **DESCRIPTION**76911 CX For `wscpy()`: The functionality described on this reference page is aligned with the ISO C  
76912 standard. Any conflict between the requirements described here and the ISO C standard is  
76913 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.76914 CX The `wpcpy()` and `wscpy()` functions shall copy the wide-character string pointed to by `ws2`  
76915 (including the terminating null wide-character code) into the array pointed to by `ws1`.76916 The application shall ensure that there is room for at least `wcslen(ws2)+1` wide characters in the  
76917 `ws1` array, and that the `ws2` and `ws1` arrays do not overlap.

76918 If copying takes place between objects that overlap, the behavior is undefined.

76919 CX The `wscpy()` and `wpcpy()` functions shall not change the setting of `errno` on valid input.76920 **RETURN VALUE**76921 CX The `wpcpy()` function shall return a pointer to the terminating null wide-character code copied  
76922 into the `ws1` buffer.76923 The `wscpy()` function shall return `ws1`.

76924 No return values are reserved to indicate an error.

76925 **ERRORS**

76926 No errors are defined.

76927 **EXAMPLES**

76928 None.

76929 **APPLICATION USAGE**

76930 None.

76931 **RATIONALE**

76932 None.

76933 **FUTURE DIRECTIONS**

76934 None.

76935 **SEE ALSO**76936 [strcpy\(\)](#), [wcsdup\(\)](#), [wscncpy\(\)](#)76937 XBD [<wchar.h>](#)76938 **CHANGE HISTORY**

76939 First released in Issue 4. Derived from the MSE working draft.

76940 **Issue 6**76941 The `wscpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.76942 **Issue 7**76943 The `wpcpy()` function is added from The Open Group Technical Standard, 2006, Extended API  
76944 Set Part 1.

76945 **Issue 8**

76946

76947

Austin Group Defect 448 is applied, adding a requirement that *wcscopy()* and *wcpcpy()* do not change the setting of *errno* on valid input.

76948 **NAME**

76949       wcscspn — get the length of a complementary wide substring

76950 **SYNOPSIS**

76951       #include &lt;wchar.h&gt;

76952       size\_t wcscspn(const wchar\_t \*ws1, const wchar\_t \*ws2);

76953 **DESCRIPTION**

76954 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
76955       conflict between the requirements described here and the ISO C standard is unintentional. This  
76956       volume of POSIX.1-2024 defers to the ISO C standard.

76957       The *wcscspn()* function shall compute the length (in wide characters) of the maximum initial  
76958       segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character  
76959       codes *not* from the wide-character string pointed to by *ws2*.

76960 CX       The *wcscspn()* function shall not change the setting of *errno* on valid input.

76961 **RETURN VALUE**

76962       The *wcscspn()* function shall return the length of the initial substring of *ws1*; no return value is  
76963       reserved to indicate an error.

76964 **ERRORS**

76965       No errors are defined.

76966 **EXAMPLES**

76967       None.

76968 **APPLICATION USAGE**

76969       None.

76970 **RATIONALE**

76971       None.

76972 **FUTURE DIRECTIONS**

76973       None.

76974 **SEE ALSO**76975       *wcsspn()*

76976       XBD &lt;wchar.h&gt;

76977 **CHANGE HISTORY**

76978       First released in Issue 4. Derived from the MSE working draft.

76979 **Issue 5**

76980       The RETURN VALUE section is updated to indicate that *wcscspn()* returns the length of *ws1*,  
76981       rather than *ws1* itself.

76982 **Issue 8**

76983       Austin Group Defect 448 is applied, adding a requirement that *wcscspn()* does not change the  
76984       setting of *errno* on valid input.

76985 **NAME**

76986           wcsdup — duplicate a wide-character string

76987 **SYNOPSIS**

```
76988 CX       #include <wchar.h>
76989       wchar_t *wcsdup(const wchar_t *string);
```

76990 **DESCRIPTION**76991           The *wcsdup()* function is the wide-character equivalent of the *strdup()* function.

76992           The *wcsdup()* function shall return a pointer to a new wide-character string, allocated as if by a call to *malloc()*, which is the duplicate of the wide-character string *string*. The returned pointer can be passed to *free()*. A null pointer is returned if the new wide-character string cannot be created.

76996 **RETURN VALUE**

76997           Upon successful completion, the *wcsdup()* function shall return a pointer to the newly allocated wide-character string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

76999 **ERRORS**77000           The *wcsdup()* function shall fail if:

77001           [ENOMEM]       Memory large enough for the duplicate string could not be allocated.

77002 **EXAMPLES**

77003           None.

77004 **APPLICATION USAGE**

77005           For functions that allocate memory as if by *malloc()*, the application should release such memory when it is no longer required by a call to *free()*. For *wcsdup()*, this is the return value.

77007 **RATIONALE**

77008           None.

77009 **FUTURE DIRECTIONS**

77010           None.

77011 **SEE ALSO**77012           *free()*, *strdup()*, *wcscpy()*

77013           XBD &lt;wchar.h&gt;

77014 **CHANGE HISTORY**

77015           First released in Issue 7.

77016 **NAME**77017 `wcsftime` — convert date and time to a wide-character string77018 **SYNOPSIS**77019 `#include <wchar.h>`77020 `size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,`  
77021 `const wchar_t *restrict format, const struct tm *restrict timeptr);`77022 **DESCRIPTION**77023 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
77024 conflict between the requirements described here and the ISO C standard is unintentional. This  
77025 volume of POSIX.1-2024 defers to the ISO C standard.77026 The `wcsftime()` function shall be equivalent to the `strftime()` function, except that:

- 77027 • The argument `wcs` points to the initial element of an array of wide characters into which  
77028 the generated output is to be placed.
- 77029 • The argument `maxsize` indicates the maximum number of wide characters to be placed in  
77030 the output array.
- 77031 • The argument `format` is a wide-character string and the conversion specifications are  
77032 replaced by corresponding sequences of wide characters. It is unspecified whether an  
77033 encoding error occurs if the format string contains `wchar_t` values that do not correspond  
77034 to members of the character set of the current locale.
- 77035 CX • Field widths specify the number of wide characters instead of the number of bytes.
- 77036 • The return value indicates the number of wide characters placed in the output array.

77037 If copying takes place between objects that overlap, the behavior is undefined.

77038 **RETURN VALUE**77039 If the total number of resulting wide-character codes including the terminating null wide-  
77040 character code is no more than `maxsize`, `wcsftime()` shall return the number of wide-character  
77041 codes placed into the array pointed to by `wcs`, not including the terminating null wide-character  
77042 code. Otherwise, zero is returned and the contents of the array are unspecified.77043 **ERRORS**

77044 No errors are defined.

77045 **EXAMPLES**

77046 None.

77047 **APPLICATION USAGE**

77048 None.

77049 **RATIONALE**

77050 None.

77051 **FUTURE DIRECTIONS**

77052 None.

77053 **SEE ALSO**77054 [strftime\(\)](#)77055 XBD [<wchar.h>](#)



77056 **CHANGE HISTORY**

77057 First released in Issue 4.

77058 **Issue 5**

77059 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

77060 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format*  
77061 argument is changed from **const char \*** to **const wchar\_t \***.77062 **Issue 6**77063 The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.77064 **Issue 7**77065 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0388 [73] and XSH/TC2-2008/0389  
77066 [740] are applied.

77067 **NAME**

77068       wcslcat, wcsncpy — size-bounded wide string concatenation and copying

77069 **SYNOPSIS**

```
77070 CX     #include <wchar.h>
77071       size_t wcslcat(wchar_t *restrict dst, const wchar_t *restrict src,
77072                    size_t dstsize);
77073       size_t wcsncpy(wchar_t *restrict dst, const wchar_t *restrict src,
77074                    size_t dstsize);
```

77075 **DESCRIPTION**

77076       The *wcsncpy()* and *wcslcat()* functions copy and concatenate wide strings, stopping when either a  
77077       terminating null wide-character code in the source wide string is encountered or the specified  
77078       full size (in wide-character codes) of the destination buffer is reached. They null terminate the  
77079       result if there is room. The application should ensure that room for the terminating null wide-  
77080       character code is included in *dstsize*.

77081       The *wcsncpy()* function shall copy not more than *dstsize* – 1 wide-character codes from the wide  
77082       string pointed to by *src* to the array pointed to by *dst*; a terminating null wide-character code in  
77083       *src* and wide-character codes that follow it shall not be copied. A terminating null wide-  
77084       character code shall be appended to the result, unless *dstsize* is 0. If copying takes place between  
77085       objects that overlap, the behavior is undefined.

77086       The *wcslcat()* function shall append not more than *dstsize* – *wcslen(dst)* – 1 wide-character codes  
77087       from the wide string pointed to by *src* to the end of the wide string pointed to by *dst*; a  
77088       terminating null wide-character code in *src* and wide-character codes that follow it shall not be  
77089       appended. The initial wide-character code of *src* shall overwrite the null wide-character code at  
77090       the end of *dst*. A terminating null wide-character code shall be appended to the result, unless its  
77091       location would be at or beyond *dst* + *dstsize*. If copying takes place between objects that overlap,  
77092       the behavior is undefined.

77093       The *wcsncpy()* and *wcslcat()* functions shall not change the setting of *errno* on valid input.

77094 **RETURN VALUE**

77095       Upon successful completion, the *wcsncpy()* function shall return the length of the wide string  
77096       pointed to by *src*; that is, the number of wide-character codes in the wide string, not including  
77097       the terminating null wide-character code.

77098       Upon successful completion, the *wcslcat()* function shall return the initial length of the wide  
77099       string pointed to by *dst* plus the length of the wide string pointed to by *src*.

77100       No return values are reserved to indicate an error.

77101 **ERRORS**

77102       No errors are defined.

77103 **EXAMPLES**

77104       None.

77105 **APPLICATION USAGE**

77106       The return value of the *wcsncpy()* and *wcslcat()* functions follows the same convention as  
77107       *snprintf()*; that is, they return the total length (in wide-character codes) of the wide string they  
77108       tried to create. If the return value is greater than or equal to *dstsize*, the output wide string has  
77109       been truncated.

- 77110 **RATIONALE**  
77111 None.
- 77112 **FUTURE DIRECTIONS**  
77113 None.
- 77114 **SEE ALSO**  
77115 *fprintf()*, *strlcat()*, *wcslen()*, *wcsncat()*, *wcsncpy()*  
77116 XBD <[wchar.h](#)>
- 77117 **CHANGE HISTORY**  
77118 First released in Issue 8.

77119 **NAME**77120 `wcslen`, `wcsnlen` — get length of a fixed-sized wide-character string77121 **SYNOPSIS**77122 `#include <wchar.h>`77123 `size_t wcslen(const wchar_t *ws);`77124 CX `size_t wcsnlen(const wchar_t *ws, size_t maxlen);`77125 **DESCRIPTION**77126 CX For `wcslen()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.77129 The `wcslen()` function shall compute the number of wide-character codes in the wide-character string to which `ws` points, not including the terminating null wide-character code.77131 CX The `wcsnlen()` function shall compute the smaller of the number of wide characters in the array to which `ws` points, not including any terminating null wide-character code, and the value of `maxlen`. The `wcsnlen()` function shall never examine more than the first `maxlen` characters of the wide-character array pointed to by `ws`.77135 CX The `wcslen()` and `wcsnlen()` functions shall not change the setting of `errno` on valid input.77136 **RETURN VALUE**77137 The `wcslen()` function shall return the length of `ws`.77138 CX The `wcsnlen()` function shall return the number of wide characters preceding the first null wide-character code in the array to which `ws` points, if `ws` contains a null wide-character code within the first `maxlen` wide characters; otherwise, it shall return `maxlen`.

77141 No return values are reserved to indicate an error.

77142 **ERRORS**

77143 No errors are defined.

77144 **EXAMPLES**

77145 None.

77146 **APPLICATION USAGE**

77147 None.

77148 **RATIONALE**

77149 None.

77150 **FUTURE DIRECTIONS**

77151 None.

77152 **SEE ALSO**77153 [`strlen\(\)`](#), [`wcslcat\(\)`](#)77154 XBD [`<wchar.h>`](#)77155 **CHANGE HISTORY**

77156 First released in Issue 4. Derived from the MSE working draft.

77157 **Issue 7**77158 The `wcsnlen()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

77160 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0390 [560] is applied.

77161 **Issue 8**

77162 Austin Group Defect 448 is applied, adding a requirement that *wcslen()* and *wcsnlen()* do not  
77163 change the setting of *errno* on valid input.

77164 Austin Group Defect 986 is applied, adding *wcslcat()* to the SEE ALSO section.

77165 **NAME**

77166        `wcsncasecmp`, `wcsncasecmp_l` — case-insensitive wide-character string comparison

77167 **SYNOPSIS**

```
77168 CX     #include <wchar.h>
77169       int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
77170       int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
77171                       size_t n, locale_t locale);
```

77172 **DESCRIPTION**

77173       Refer to [wcscasecmp\(\)](#).

77174 **NAME**

77175           wcsncat — concatenate a wide-character string with part of another

77176 **SYNOPSIS**

77177           #include &lt;wchar.h&gt;

77178           wchar\_t \*wcsncat(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
77179                           size\_t n);77180 **DESCRIPTION**77181 CX       The functionality described on this reference page is aligned with the ISO C standard. Any  
77182       conflict between the requirements described here and the ISO C standard is unintentional. This  
77183       volume of POSIX.1-2024 defers to the ISO C standard.77184       The *wcsncat()* function shall append not more than *n* wide-character codes (a null wide-  
77185       character code and wide-character codes that follow it are not appended) from the array pointed  
77186       to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character  
77187       code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null  
77188       wide-character code shall always be appended to the result. If copying takes place between  
77189       objects that overlap, the behavior is undefined.77190 CX       The *wcsncat()* function shall not change the setting of *errno* on valid input.77191 **RETURN VALUE**77192       The *wcsncat()* function shall return *ws1*; no return value is reserved to indicate an error.77193 **ERRORS**

77194       No errors are defined.

77195 **EXAMPLES**

77196       None.

77197 **APPLICATION USAGE**

77198       None.

77199 **RATIONALE**

77200       None.

77201 **FUTURE DIRECTIONS**

77202       None.

77203 **SEE ALSO**77204       *wcscat()*, *wcslcat()*

77205       XBD &lt;wchar.h&gt;

77206 **CHANGE HISTORY**

77207       First released in Issue 4. Derived from the MSE working draft.

77208 **Issue 6**77209       The *wcsncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.77210 **Issue 8**77211       Austin Group Defect 448 is applied, adding a requirement that *wcsncat()* does not change the  
77212       setting of *errno* on valid input.77213       Austin Group Defect 986 is applied, adding *wcslcat()* to the SEE ALSO section.

77214 **NAME**

77215           wcsncmp — compare part of two wide-character strings

77216 **SYNOPSIS**

77217           #include &lt;wchar.h&gt;

77218           int wcsncmp(const wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

77219 **DESCRIPTION**77220 **CX**       The functionality described on this reference page is aligned with the ISO C standard. Any  
77221 conflict between the requirements described here and the ISO C standard is unintentional. This  
77222 volume of POSIX.1-2024 defers to the ISO C standard.77223       The *wcsncmp()* function shall compare not more than *n* wide-character codes (wide-character  
77224 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*  
77225 to the array pointed to by *ws2*.77226       The sign of a non-zero return value shall be determined by the sign of the difference between the  
77227 values of the first pair of wide-character codes that differ in the objects being compared.77228 **CX**       The *wcsncmp()* function shall not change the setting of *errno* on valid input.77229 **RETURN VALUE**77230       Upon successful completion, *wcsncmp()* shall return an integer greater than, equal to, or less  
77231 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less  
77232 than the possibly null-terminated array pointed to by *ws2*, respectively.77233 **ERRORS**

77234       No errors are defined.

77235 **EXAMPLES**

77236       None.

77237 **APPLICATION USAGE**

77238       None.

77239 **RATIONALE**

77240       None.

77241 **FUTURE DIRECTIONS**

77242       None.

77243 **SEE ALSO**77244       [wscasecmp\(\)](#), [wscmp\(\)](#)77245       XBD [<wchar.h>](#)77246 **CHANGE HISTORY**

77247       First released in Issue 4. Derived from the MSE working draft.

77248 **Issue 8**77249       Austin Group Defect 448 is applied, adding a requirement that *wcsncmp()* does not change the  
77250 setting of *errno* on valid input.



77251 **NAME**

77252           wcsncpy, wcsncpy — copy a fixed-size wide-character string, returning a pointer to its end

77253 **SYNOPSIS**

77254           #include &lt;wchar.h&gt;

77255 CX        wchar\_t \*wcpncpy(wchar\_t restrict \*ws1, const wchar\_t \*restrict ws2,  
77256            size\_t n);77257        wchar\_t \*wcsncpy(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
77258            size\_t n);77259 **DESCRIPTION**77260 CX        For *wcsncpy()*: The functionality described on this reference page is aligned with the ISO C  
77261 standard. Any conflict between the requirements described here and the ISO C standard is  
77262 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.77263 CX        The *wcpncpy()* and *wcsncpy()* functions shall copy not more than *n* wide-character codes (wide-  
77264 character codes that follow a null wide-character code are not copied) from the array pointed to  
77265 by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the  
77266 behavior is undefined.77267            If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character  
77268 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*,  
77269 until *n* wide-character codes in all are written.77270 CX        The *wcsncpy()* and *wcpncpy()* functions shall not change the setting of *errno* on valid input.77271 **RETURN VALUE**77272 CX        If any null wide-character codes were written into the destination, the *wcpncpy()* function shall  
77273 return the address of the first such null wide-character code. Otherwise, it shall return *&ws1[n]*.77274            The *wcsncpy()* function shall return *ws1*.

77275            No return values are reserved to indicate an error.

77276 **ERRORS**

77277            No errors are defined.

77278 **EXAMPLES**

77279            None.

77280 **APPLICATION USAGE**77281            If there is no null wide-character code in the first *n* wide-character codes of the array pointed to  
77282 by *ws2*, the result is not null-terminated.77283 **RATIONALE**

77284            None.

77285 **FUTURE DIRECTIONS**

77286            None.

77287 **SEE ALSO**77288            [strncpy\(\)](#), [wcscpy\(\)](#), [wcsncpy\(\)](#)77289            XBD [<wchar.h>](#)77290 **CHANGE HISTORY**

77291            First released in Issue 4. Derived from the MSE working draft.

77292 **Issue 6**

77293 The *wcsncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

77294 **Issue 7**

77295 The *wcpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API  
77296 Set Part 1.

77297 **Issue 8**

77298 Austin Group Defect 448 is applied, adding a requirement that *wcsncpy()* and *wcpncpy()* do not  
77299 change the setting of *errno* on valid input.

77300 Austin Group Defect 986 is applied, adding *wslcat()* to the SEE ALSO section.

77301 **NAME**

77302       wcsnlen — get length of a fixed-sized wide-character string

77303 **SYNOPSIS**

```
77304 CX       #include <wchar.h>  
77305       size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

77306 **DESCRIPTION**77307       Refer to *wcslen()*.

77308 **NAME**

77309           wcsnrtoombs — convert wide-character string to multi-byte string

77310 **SYNOPSIS**

```
77311 CX       #include <wchar.h>
77312           size_t wcsnrtoombs(char *restrict dst, const wchar_t **restrict src,
77313                           size_t nwc, size_t len, mbstate_t *restrict ps);
```

77314 **DESCRIPTION**77315           Refer to *wcsrtoombs()*.

77316 **NAME**

77317 wcpbrk — scan a wide-character string for a wide-character code

77318 **SYNOPSIS**

77319 #include &lt;wchar.h&gt;

77320 wchar\_t \*wcpbrk(const wchar\_t \*ws1, const wchar\_t \*ws2);

77321 **DESCRIPTION**77322 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
77323 conflict between the requirements described here and the ISO C standard is unintentional. This  
77324 volume of POSIX.1-2024 defers to the ISO C standard.77325 The *wcpbrk()* function shall locate the first occurrence in the wide-character string pointed to by  
77326 *ws1* of any wide-character code from the wide-character string pointed to by *ws2*.77327 CX The *wcpbrk()* function shall not change the setting of *errno* on valid input.77328 **RETURN VALUE**77329 Upon successful completion, *wcpbrk()* shall return a pointer to the wide-character code or a null  
77330 pointer if no wide-character code from *ws2* occurs in *ws1*.77331 **ERRORS**

77332 No errors are defined.

77333 **EXAMPLES**

77334 None.

77335 **APPLICATION USAGE**

77336 None.

77337 **RATIONALE**

77338 None.

77339 **FUTURE DIRECTIONS**

77340 None.

77341 **SEE ALSO**77342 [wcschr\(\)](#), [wcsrchr\(\)](#)77343 XBD [<wchar.h>](#)77344 **CHANGE HISTORY**

77345 First released in Issue 4. Derived from the MSE working draft.

77346 **Issue 8**77347 Austin Group Defect 448 is applied, adding a requirement that *wcpbrk()* does not change the  
77348 setting of *errno* on valid input.

77349 **NAME**

77350 wcsrchr — wide-character string scanning operation

77351 **SYNOPSIS**

77352 #include &lt;wchar.h&gt;

77353 wchar\_t \*wcsrchr(const wchar\_t \*ws, wchar\_t wc);

77354 **DESCRIPTION**

77355 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
77356 conflict between the requirements described here and the ISO C standard is unintentional. This  
77357 volume of POSIX.1-2024 defers to the ISO C standard.

77358 The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed  
77359 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type  
77360 **wchar\_t** and a wide-character code corresponding to a valid character in the current locale. The  
77361 terminating null wide-character code shall be considered to be part of the wide-character string.

77362 CX The *wcsrchr()* function shall not change the setting of *errno* on valid input.

77363 **RETURN VALUE**

77364 Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null  
77365 pointer if *wc* does not occur in the wide-character string.

77366 **ERRORS**

77367 No errors are defined.

77368 **EXAMPLES**

77369 None.

77370 **APPLICATION USAGE**

77371 None.

77372 **RATIONALE**

77373 None.

77374 **FUTURE DIRECTIONS**

77375 None.

77376 **SEE ALSO**77377 [wcschr\(\)](#)77378 XBD [<wchar.h>](#)77379 **CHANGE HISTORY**

77380 First released in Issue 4. Derived from the MSE working draft.

77381 **Issue 6**

77382 The normative text is updated to avoid use of the term “must” for application requirements.

77383 **Issue 8**

77384 Austin Group Defect 448 is applied, adding a requirement that *wcsrchr()* does not change the  
77385 setting of *errno* on valid input.

77386 **NAME**77387 `wcsnrtombs`, `wcsrtombs` — convert a wide-character string to a character string (restartable)77388 **SYNOPSIS**77389 `#include <wchar.h>`

```
77390 CX   size_t wcsnrtombs(char *restrict dst, const wchar_t **restrict src,
77391         size_t nwc, size_t len, mbstate_t *restrict ps);
77392     size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,
77393         size_t len, mbstate_t *restrict ps);
```

77394 **DESCRIPTION**

77395 CX For `wcsrtombs()`: The functionality described on this reference page is aligned with the ISO C  
 77396 standard. Any conflict between the requirements described here and the ISO C standard is  
 77397 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.

77398 The `wcsrtombs()` function shall convert a sequence of wide characters from the array indirectly  
 77399 pointed to by `src` into a sequence of corresponding characters, beginning in the conversion state  
 77400 described by the object pointed to by `ps`. If `dst` is not a null pointer, the converted characters  
 77401 shall then be stored into the array pointed to by `dst`. Conversion continues up to and including a  
 77402 terminating null wide character, which shall also be stored. Conversion shall stop earlier in the  
 77403 following cases:

- 77404 • When a code is reached that does not correspond to a valid character
- 77405 • When the next character would exceed the limit of `len` total bytes to be stored in the array  
 77406 pointed to by `dst` (and `dst` is not a null pointer)

77407 Each conversion shall take place as if by a call to the `wcrtomb()` function.

77408 If `dst` is not a null pointer, the pointer object pointed to by `src` shall be assigned either a null  
 77409 pointer (if conversion stopped due to reaching a terminating null wide character) or the address  
 77410 just past the last wide character converted (if any). If conversion stopped due to reaching a  
 77411 terminating null wide character, the resulting state described shall be the initial conversion state.

77412 If `ps` is a null pointer, the `wcsrtombs()` function shall use its own internal `mbstate_t` object, which  
 77413 is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object  
 77414 pointed to by `ps` shall be used to completely describe the current conversion state of the  
 77415 associated character sequence.

77416 If called with a null `ps` argument, the `wcsrtombs()` function need not be thread-safe; however,  
 77417 such calls shall avoid data races with calls to `wcsrtombs()` with a non-null argument and with  
 77418 calls to all other functions.

77419 CX The `wcsnrtombs()` function shall be equivalent to the `wcsrtombs()` function, except that the  
 77420 conversion is limited to the first `nwc` wide characters.

77421 If called with a null `ps` argument, the `wcsnrtombs()` function need not be thread-safe; however,  
 77422 such calls shall avoid data races with calls to `wcsnrtombs()` with a non-null argument and with  
 77423 calls to all other functions.

77424 These functions shall not change the setting of `errno` if successful.

77425 The behavior of these functions shall be affected by the `LC_CTYPE` category of the current locale.

77426 The implementation shall behave as if no function defined in System Interfaces volume of  
 77427 POSIX.1-2024 calls these functions.

77428 **RETURN VALUE**

77429 If conversion stops because a code is reached that does not correspond to a valid character, an  
 77430 encoding error occurs. In this case, these functions shall store the value of the macro [EILSEQ] in  
 77431 *errno* and return (**size\_t**)-1; the conversion state is undefined. Otherwise, these functions shall  
 77432 return the number of bytes in the resulting character sequence, not including the terminating  
 77433 null (if any).

77434 **ERRORS**

77435 These functions shall fail if:

77436 [EILSEQ] A wide-character code does not correspond to a valid character.

77437 These functions may fail if:

77438 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

77439 **EXAMPLES**

77440 None.

77441 **APPLICATION USAGE**

77442 None.

77443 **RATIONALE**

77444 None.

77445 **FUTURE DIRECTIONS**

77446 None.

77447 **SEE ALSO**

77448 [\*mbsinit\(\)\*](#), [\*wcrtomb\(\)\*](#)

77449 XBD <[wchar.h](#)>

77450 **CHANGE HISTORY**

77451 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 77452 (E).

77453 **Issue 6**

77454 In the DESCRIPTION, a note on using this function in a threaded application is added.

77455 Extensions beyond the ISO C standard are marked.

77456 The normative text is updated to avoid use of the term “must” for application requirements.

77457 The *wcsrtoombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

77458 **Issue 7**

77459 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcsrtoombs()* function  
 77460 need not be thread-safe if called with a NULL *ps* argument.

77461 Austin Group Interpretation 1003.1-2001 #170 is applied.

77462 The *wcsnrtoombs()* function is added from The Open Group Technical Standard, 2006, Extended  
 77463 API Set Part 1.

77464 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0722 [109,105] is applied.

77465 **Issue 8**

77466 Austin Group Defect 1302 is applied, aligning these functions with the ISO/IEC 9899:2018  
 77467 standard.



77468 **NAME**77469 `wcsspnp` — get the length of a wide substring77470 **SYNOPSIS**77471 `#include <wchar.h>`77472 `size_t wcsspnp(const wchar_t *ws1, const wchar_t *ws2);`77473 **DESCRIPTION**77474 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
77475 conflict between the requirements described here and the ISO C standard is unintentional. This  
77476 volume of POSIX.1-2024 defers to the ISO C standard.77477 The `wcsspnp()` function shall compute the length (in wide characters) of the maximum initial  
77478 segment of the wide-character string pointed to by `ws1` which consists entirely of wide-character  
77479 codes from the wide-character string pointed to by `ws2`.77480 CX The `wcsspnp()` function shall not change the setting of `errno` on valid input.77481 **RETURN VALUE**77482 The `wcsspnp()` function shall return the length of the initial substring of `ws1`; no return value is  
77483 reserved to indicate an error.77484 **ERRORS**

77485 No errors are defined.

77486 **EXAMPLES**

77487 None.

77488 **APPLICATION USAGE**

77489 None.

77490 **RATIONALE**

77491 None.

77492 **FUTURE DIRECTIONS**

77493 None.

77494 **SEE ALSO**77495 [wcscspnp\(\)](#)77496 XBD [<wchar.h>](#)77497 **CHANGE HISTORY**

77498 First released in Issue 4. Derived from the MSE working draft.

77499 **Issue 5**77500 The RETURN VALUE section is updated to indicate that `wcsspnp()` returns the length of `ws1`  
77501 rather than `ws1` itself.77502 **Issue 8**77503 Austin Group Defect 448 is applied, adding a requirement that `wcsspnp()` does not change the  
77504 setting of `errno` on valid input.

77505 **NAME**77506 `wcsstr` — find a wide-character substring77507 **SYNOPSIS**77508 `#include <wchar.h>`77509 `wchar_t *wcsstr(const wchar_t *restrict ws1,`  
77510 `const wchar_t *restrict ws2);`77511 **DESCRIPTION**77512 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
77513 conflict between the requirements described here and the ISO C standard is unintentional. This  
77514 volume of POSIX.1-2024 defers to the ISO C standard.77515 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by  
77516 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the  
77517 wide-character string pointed to by `ws2`.77518 CX The `wcsstr()` function shall not change the setting of `errno` on valid input.77519 **RETURN VALUE**77520 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,  
77521 or a null pointer if the wide-character string is not found.77522 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.77523 **ERRORS**

77524 No errors are defined.

77525 **EXAMPLES**

77526 None.

77527 **APPLICATION USAGE**

77528 None.

77529 **RATIONALE**

77530 None.

77531 **FUTURE DIRECTIONS**

77532 None.

77533 **SEE ALSO**77534 [`wcschr\(\)`](#)77535 XBD [`<wchar.h>`](#)77536 **CHANGE HISTORY**77537 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
77538 (E).77539 **Issue 6**77540 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.77541 **Issue 8**77542 Austin Group Defect 448 is applied, adding a requirement that `wcsstr()` does not change the  
77543 setting of `errno` on valid input.

77544 **NAME**

77545 wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

77546 **SYNOPSIS**

77547 #include &lt;wchar.h&gt;

77548 double wcstod(const wchar\_t \*restrict *nptr*, wchar\_t \*\*restrict *endptr*);77549 float wcstof(const wchar\_t \*restrict *nptr*, wchar\_t \*\*restrict *endptr*);77550 long double wcstold(const wchar\_t \*restrict *nptr*,77551 wchar\_t \*\*restrict *endptr*);77552 **DESCRIPTION**

77553 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 77554 conflict between the requirements described here and the ISO C standard is unintentional. This  
 77555 volume of POSIX.1-2024 defers to the ISO C standard.

77556 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
 77557 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the  
 77558 input wide-character string into three parts:

- 77559 1. An initial, possibly empty, sequence of white-space wide characters
- 77560 2. A subject sequence interpreted as a floating-point constant or representing infinity or  
 77561 NaN
- 77562 3. A final wide-character string of one or more unrecognized wide-character codes,  
 77563 including the terminating null wide-character code of the input wide-character string

77564 Then they shall attempt to convert the subject sequence to a floating-point number, and return  
 77565 the result.

77566 The expected form of the subject sequence is an optional '+' or '-' sign, then one of the  
 77567 following:

- 77568 • A non-empty sequence of decimal digits optionally containing a radix character; then an  
 77569 optional exponent part consisting of the wide character 'e' or the wide character 'E',  
 77570 optionally followed by a '+' or '-' wide character, and then followed by one or more  
 77571 decimal digits
- 77572 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix  
 77573 character; then an optional binary exponent part consisting of the wide character 'p' or  
 77574 the wide character 'P', optionally followed by a '+' or '-' wide character, and then  
 77575 followed by one or more decimal digits
- 77576 • One of INF or INFINITY, or any other wide string equivalent except for case
- 77577 • One of NAN or NAN(*n-wchar-sequence<sub>opt</sub>*), or any other wide string ignoring case in the  
 77578 NAN part, where:

77579 n-wchar-sequence:

77580 digit

77581 nondigit

77582 n-wchar-sequence digit

77583 n-wchar-sequence nondigit

77584 The subject sequence is defined as the longest initial subsequence of the input wide string,  
 77585 starting with the first non-white-space wide character, that is of the expected form. The subject  
 77586 sequence contains no wide characters if the input wide string is not of the expected form.

77587 If the subject sequence has the expected form for a floating-point number, the sequence of wide  
 77588 characters starting with the first digit or the radix character (whichever occurs first) shall be

77589 interpreted as a floating constant according to the rules of the C language, except that the radix  
 77590 character shall be used in place of a period, and that if neither an exponent part nor a radix  
 77591 character appears in a decimal floating-point number, or if a binary exponent part does not  
 77592 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with  
 77593 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins  
 77594 with a <hyphen-minus>, the sequence shall be interpreted as negated. A wide-character  
 77595 sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type,  
 77596 else as if it were a floating constant that is too large for the range of the return type. A wide-  
 77597 character sequence NAN or NAN(*n-wchar-sequence<sub>opt</sub>*) shall be interpreted as a quiet NaN, if  
 77598 supported in the return type, else as if it were a subject sequence part that does not have the  
 77599 expected form; the meaning of the *n-wchar* sequences is implementation-defined. A pointer to  
 77600 the final wide string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not  
 77601 a null pointer.

77602 If the subject sequence has the hexadecimal form and FLT\_RADIX is a power of 2, the  
 77603 conversion shall be rounded in an implementation-defined manner.

77604 CX The radix character shall be as defined in the current locale (category *LC\_NUMERIC*). In the  
 77605 POSIX locale, or in a locale where the radix character is not defined, the radix character shall  
 77606 default to a <period> ( ' . ' ).

77607 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be  
 77608 accepted.

77609 If the subject sequence is empty or does not have the expected form, no conversion shall be  
 77610 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
 77611 *endptr* is not a null pointer.

77612 These functions shall not change the setting of *errno* if successful.

77613 Since 0 is returned on error and is also a valid return on success, an application wishing to check  
 77614 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check  
 77615 *errno*.

#### 77616 RETURN VALUE

77617 Upon successful completion, these functions shall return the converted value. If no conversion  
 77618 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

77619 If the correct value would cause an overflow and default rounding is in effect, ±HUGE\_VAL,  
 77620 ±HUGE\_VALF, or ±HUGE\_VALL shall be returned (according to the sign of the value), and *errno*  
 77621 shall be set to [ERANGE].

77622 If the correct value would cause underflow, a value whose magnitude is no greater than the  
 77623 CX smallest normalized positive number in the return type shall be returned and *errno* set to  
 77624 [ERANGE].

#### 77625 ERRORS

77626 The *wcstod()* function shall fail if:

77627 [ERANGE] The value to be returned would cause overflow and default rounding is in  
 77628 CX effect or the value to be returned would cause underflow.

77629 The *wcstod()* function may fail if:

77630 CX [EINVAL] No conversion could be performed.

77631 **EXAMPLES**

77632 None.

77633 **APPLICATION USAGE**

77634 If the subject sequence has the hexadecimal form and FLT\_RADIX is not a power of 2, and the  
 77635 result is not exactly representable, the result should be one of the two numbers in the  
 77636 appropriate internal format that are adjacent to the hexadecimal floating source value, with the  
 77637 extra stipulation that the error should have a correct sign for the current rounding direction.

77638 If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in `<float.h>`)  
 77639 significant digits, the result should be correctly rounded. If the subject sequence *D* has the  
 77640 decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding,  
 77641 adjacent decimal strings *L* and *U*, both having DECIMAL\_DIG significant digits, such that the  
 77642 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or  
 77643 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current  
 77644 rounding direction, with the extra stipulation that the error with respect to *D* should have a  
 77645 correct sign for the current rounding direction.

77646 **RATIONALE**

77647 None.

77648 **FUTURE DIRECTIONS**

77649 None.

77650 **SEE ALSO**77651 [\*fscanf\(\)\*](#), [\*iswspace\(\)\*](#), [\*localeconv\(\)\*](#), [\*setlocale\(\)\*](#), [\*wcstol\(\)\*](#)77652 XBD [Chapter 7](#) (on page 127), `<float.h>`, `<wchar.h>`77653 **CHANGE HISTORY**

77654 First released in Issue 4. Derived from the MSE working draft.

77655 **Issue 5**77656 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.77657 **Issue 6**

77658 Extensions beyond the ISO C standard are marked.

77659 The following new requirements on POSIX implementations derive from alignment with the  
 77660 Single UNIX Specification:

- 77661 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
 77662 added if no conversion could be performed.

77663 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 77664 • The *wcstod()* prototype is updated.
- 77665 • The *wcstof()* and *wcstold()* functions are added.
- 77666 • If the correct value for *wcstod()* would cause underflow, the return value changed from 0  
 77667 (as specified in Issue 5) to the smallest normalized positive number.
- 77668 • The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are  
 77669 extensively updated.

77670 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

77671 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second  
 77672 paragraph in the RETURN VALUE section.

77673 **Issue 7**

77674 Austin Group Interpretation 1003.1-2001 #015 is applied.

77675 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0723 [302] and XSH/TC1-2008/0724  
77676 [105] are applied.77677 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0391 [584] and XSH/TC2-2008/0392  
77678 [796] are applied.77679 **Issue 8**

77680 Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

77681 Austin Group Defect 1686 is applied, addressing some inconsistencies with *strtod*().

77682 **NAME**77683 `wcstoimax`, `wcstoumax` — convert a wide-character string to an integer type77684 **SYNOPSIS**

```
77685 #include <stddef.h>
77686 #include <inttypes.h>
77687
77687 intmax_t wcstoimax(const wchar_t *restrict nptr,
77688                   wchar_t **restrict endptr, int base);
77689
77689 uintmax_t wcstoumax(const wchar_t *restrict nptr,
77690                    wchar_t **restrict endptr, int base);
```

77691 **DESCRIPTION**

77692 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 77693 conflict between the requirements described here and the ISO C standard is unintentional. This  
 77694 volume of POSIX.1-2024 defers to the ISO C standard.

77695 These functions shall be equivalent to the `wcstol()`, `wcstoll()`, `wcstoul()`, and `wcstoull()` functions,  
 77696 respectively, except that the initial portion of the wide string shall be converted to `intmax_t` and  
 77697 `uintmax_t` representation, respectively.

77698 **RETURN VALUE**

77699 These functions shall return the converted value, if any.

77700 If no conversion could be performed, zero shall be returned. If the correct value is outside the  
 77701 range of representable values, `{INTMAX_MAX}`, `{INTMAX_MIN}`, or `{UINTMAX_MAX}` shall  
 77702 be returned (according to the return type and sign of the value, if any), and `errno` shall be set to  
 77703 `[ERANGE]`.

77704 **ERRORS**

77705 These functions shall fail if:

77706 `[EINVAL]` The value of `base` is not supported.77707 `[ERANGE]` The value to be returned is not representable.

77708 These functions may fail if:

77709 `[EINVAL]` No conversion could be performed.77710 **EXAMPLES**

77711 None.

77712 **APPLICATION USAGE**

77713 None.

77714 **RATIONALE**

77715 None.

77716 **FUTURE DIRECTIONS**

77717 None.

77718 **SEE ALSO**77719 `wcstol()`, `wcstoul()`77720 XBD `<inttypes.h>`, `<stddef.h>`77721 **CHANGE HISTORY**

77722 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

77723 **NAME**

77724 wcstok — split a wide-character string into tokens

77725 **SYNOPSIS**

77726 #include &lt;wchar.h&gt;

77727 wchar\_t \*wcstok(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
77728 wchar\_t \*\*restrict ptr);77729 **DESCRIPTION**77730 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
77731 conflict between the requirements described here and the ISO C standard is unintentional. This  
77732 volume of POSIX.1-2024 defers to the ISO C standard.77733 A sequence of calls to *wcstok()* shall break the wide-character string pointed to by *ws1* into a  
77734 sequence of tokens, each of which shall be delimited by a wide-character code from the wide-  
77735 character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar\_t** pointer  
77736 into which the *wcstok()* function shall store information necessary for it to continue scanning the  
77737 same wide-character string.77738 The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null  
77739 pointer as their first argument. The separator string pointed to by *ws2* may be different from call  
77740 to call.77741 The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first  
77742 wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no  
77743 such wide-character code is found, then there are no tokens in the wide-character string pointed  
77744 to by *ws1* and *wcstok()* shall return a null pointer. If such a wide-character code is found, it shall  
77745 be the start of the first token.77746 The *wcstok()* function shall then search from there for a wide-character code that *is* contained in  
77747 the current separator string. If no such wide-character code is found, the current token extends  
77748 to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token  
77749 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a  
77750 null wide character, which terminates the current token. The *wcstok()* function shall save a  
77751 pointer to the following wide-character code, from which the next search for a token shall start.77752 Each subsequent call, with a null pointer as the value of the first argument, shall start searching  
77753 from the saved pointer and behave as described above.77754 The implementation shall behave as if no function calls *wcstok()*.77755 CX The *wcstok()* function shall not change the setting of *errno* on valid input.77756 **RETURN VALUE**77757 Upon successful completion, the *wcstok()* function shall return a pointer to the first wide-  
77758 character code of a token. Otherwise, if there is no token, *wcstok()* shall return a null pointer.77759 **ERRORS**

77760 No errors are defined.



77761 **EXAMPLES**

77762 None.

77763 **APPLICATION USAGE**

77764 None.

77765 **RATIONALE**

77766 None.

77767 **FUTURE DIRECTIONS**

77768 None.

77769 **SEE ALSO**77770 XBD [<wchar.h>](#)77771 **CHANGE HISTORY**

77772 First released in Issue 4.

77773 **Issue 5**77774 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is  
77775 added to the definition of *wcstok()* in the SYNOPSIS.77776 **Issue 6**77777 The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.77778 **Issue 8**77779 Austin Group Defect 448 is applied, adding a requirement that *wcstok()* does not change the  
77780 setting of *errno* on valid input.

77781 **NAME**

77782           wcstol, wcstoll — convert a wide-character string to a long integer

77783 **SYNOPSIS**

```
77784     #include <wchar.h>
77785     long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
77786                int base);
77787     long long wcstoll(const wchar_t *restrict nptr,
77788                      wchar_t **restrict endptr, int base);
```

77789 **DESCRIPTION**

77790 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
 77791 conflict between the requirements described here and the ISO C standard is unintentional. This  
 77792 volume of POSIX.1-2024 defers to the ISO C standard.

77793 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to  
 77794 **long** and **long long**, respectively. First, they shall decompose the input string into three parts:

- 77795       1. An initial, possibly empty, sequence of white-space wide characters
- 77796       2. A subject sequence interpreted as an integer represented in some radix determined by the  
 77797       value of *base*
- 77798       3. A final wide-character string of one or more unrecognized wide-character codes,  
 77799       including the terminating null wide-character code of the input wide-character string

77800 Then they shall attempt to convert the subject sequence to an integer, and return the result.

77801 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
 77802 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
 77803 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
 77804 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
 77805 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 77806 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

77807 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence  
 77808 of letters and digits representing an integer with the radix specified by *base*, optionally preceded  
 77809 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
 77810 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
 77811 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code  
 77812 representations of 0x or 0X may optionally precede the sequence of letters and digits, following  
 77813 the sign if present.

77814 The subject sequence is defined as the longest initial subsequence of the input wide-character  
 77815 string, starting with the first non-white-space wide character, that is of the expected form. The  
 77816 subject sequence contains no wide-character codes if the input wide-character string is empty or  
 77817 consists entirely of white-space wide characters, or if the first non-white-space wide character is  
 77818 other than a sign or a permissible letter or digit.

77819 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes  
 77820 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has  
 77821 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for  
 77822 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a  
 77823 <hyphen-minus>, the resulting value shall be the negative of the converted value. A pointer to  
 77824 the final wide-character string shall be stored in the object pointed to by *endptr*, provided that  
 77825 *endptr* is not a null pointer.

77826 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be  
77827 accepted.

77828 If the subject sequence is empty or does not have the expected form, no conversion shall be  
77829 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
77830 *endptr* is not a null pointer.

77831 These functions shall not change the setting of *errno* if successful.

77832 Since 0, {LONG\_MIN} or {LLONG\_MIN} and {LONG\_MAX} or {LLONG\_MAX} are returned on  
77833 error and are also valid returns on success, an application wishing to check for error situations  
77834 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.

#### 77835 RETURN VALUE

77836 Upon successful completion, these functions shall return the converted value, if any. If no  
77837 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If  
77838 the correct value is outside the range of representable values, {LONG\_MIN}, {LONG\_MAX},  
77839 {LLONG\_MIN}, or {LLONG\_MAX} shall be returned (according to the sign of the value), and  
77840 *errno* set to [ERANGE].

#### 77841 ERRORS

77842 These functions shall fail if:

77843 CX [EINVAL] The value of *base* is not supported.

77844 [ERANGE] The value to be returned is not representable.

77845 These functions may fail if:

77846 CX [EINVAL] No conversion could be performed.

#### 77847 EXAMPLES

77848 None.

#### 77849 APPLICATION USAGE

77850 None.

#### 77851 RATIONALE

77852 None.

#### 77853 FUTURE DIRECTIONS

77854 None.

#### 77855 SEE ALSO

77856 *fscanf()*, *iswalpha()*, *wcstod()*

77857 XBD <wchar.h>

#### 77858 CHANGE HISTORY

77859 First released in Issue 4. Derived from the MSE working draft.

#### 77860 Issue 5

77861 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 77862 Issue 6

77863 Extensions beyond the ISO C standard are marked.

77864 The following new requirements on POSIX implementations derive from alignment with the  
77865 Single UNIX Specification:

- 77866 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
77867 added if no conversion could be performed.

77868 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 77869 • The *wcstol()* prototype is updated.
- 77870 • The *wcstoll()* function is added.

77871 **Issue 7**

77872 SD5-XSH-ERN-56 is applied, removing the reference to **unsigned long** and **unsigned long long**  
77873 from the DESCRIPTION.

77874 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0725 [105] is applied.

77875 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0393 [584] and XSH/TC2-2008/0394  
77876 [796] are applied.

77877 **Issue 8**

77878 Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-  
77879 minus> is converted.

77880 Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

77881 **NAME**

77882           wcstold — convert a wide-character string to a double-precision number

77883 **SYNOPSIS**

77884           #include &lt;wchar.h&gt;

77885           long double wcstold(const wchar\_t \*restrict *nptr*,77886                            wchar\_t \*\*restrict *endptr*);77887 **DESCRIPTION**77888           Refer to *wcstod()*.

77889 **NAME**

77890           wcstoll — convert a wide-character string to a long integer

77891 **SYNOPSIS**

77892           #include <wchar.h>

77893           long long wcstoll(const wchar\_t \*restrict *nptr*,  
77894                            wchar\_t \*\*restrict *endptr*, int *base*);

77895 **DESCRIPTION**

77896           Refer to *wcstol()*.

77897 **NAME**

77898 wcstombs — convert a wide-character string to a character string

77899 **SYNOPSIS**

77900 #include &lt;stdlib.h&gt;

77901 size\_t wcstombs(char \*restrict s, const wchar\_t \*restrict pwcs,  
77902 size\_t n);77903 **DESCRIPTION**77904 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
77905 conflict between the requirements described here and the ISO C standard is unintentional. This  
77906 volume of POSIX.1-2024 defers to the ISO C standard.77907 The *wcstombs()* function shall convert the sequence of wide-character codes that are in the array  
77908 pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store  
77909 these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n*  
77910 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to  
77911 *wctomb()*, except that the shift state of *wctomb()* shall not be affected.77912 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.77913 No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place  
77914 CX between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs()* shall  
77915 return the length required to convert the entire array regardless of the value of *n*, but no values  
77916 are stored.77917 **RETURN VALUE**77918 If a wide-character code is encountered that does not correspond to a valid character (of one or  
77919 more bytes each), *wcstombs()* shall return (**size\_t**)−1. Otherwise, *wcstombs()* shall return the  
77920 number of bytes stored in the character array, not including any terminating null byte. The array  
77921 shall not be null-terminated if the value returned is *n*.77922 **ERRORS**77923 The *wcstombs()* function shall fail if:

77924 CX [EILSEQ] A wide-character code does not correspond to a valid character.

77925 **EXAMPLES**

77926 None.

77927 **APPLICATION USAGE**

77928 None.

77929 **RATIONALE**

77930 None.

77931 **FUTURE DIRECTIONS**

77932 None.

77933 **SEE ALSO**77934 [mblen\(\)](#), [mbtowc\(\)](#), [mbstowcs\(\)](#), [wctomb\(\)](#)77935 XBD [<stdlib.h>](#)77936 **CHANGE HISTORY**

77937 First released in Issue 4. Derived from the ISO C standard.

77938 **Issue 6**

77939 The following new requirements on POSIX implementations derive from alignment with the  
77940 Single UNIX Specification:

- 77941 • The DESCRIPTION states the effect of when *s* is a null pointer.
- 77942 • The [EILSEQ] error condition is added.

77943 The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

77944 **Issue 7**

77945 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

77946 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0726 [109] is applied.



77947 **NAME**77948 `wcstoul, wcstoull` — convert a wide-character string to an unsigned long77949 **SYNOPSIS**

```
77950 #include <wchar.h>
77951 unsigned long wcstoul(const wchar_t *restrict nptr,
77952     wchar_t **restrict endptr, int base);
77953 unsigned long long wcstoull(const wchar_t *restrict nptr,
77954     wchar_t **restrict endptr, int base);
```

77955 **DESCRIPTION**

77956 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 77957 conflict between the requirements described here and the ISO C standard is unintentional. This  
 77958 volume of POSIX.1-2024 defers to the ISO C standard.

77959 The `wcstoul()` and `wcstoull()` functions shall convert the initial portion of the wide-character  
 77960 string pointed to by `nptr` to **unsigned long** and **unsigned long long** representation, respectively.  
 77961 First, they shall decompose the input wide-character string into three parts:

- 77962 1. An initial, possibly empty, sequence of white-space wide characters
- 77963 2. A subject sequence interpreted as an integer represented in some radix determined by the  
 77964 value of `base`
- 77965 3. A final wide-character string of one or more unrecognized wide-character codes,  
 77966 including the terminating null wide-character code of the input wide-character string

77967 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the  
 77968 result.

77969 If `base` is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,  
 77970 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal  
 77971 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal  
 77972 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'  
 77973 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the  
 77974 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

77975 If the value of `base` is between 2 and 36, the expected form of the subject sequence is a sequence  
 77976 of letters and digits representing an integer with the radix specified by `base`, optionally preceded  
 77977 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'  
 77978 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less  
 77979 than that of `base` shall be permitted. If the value of `base` is 16, the wide-character codes 0x or 0X  
 77980 may optionally precede the sequence of letters and digits, following the sign if present.

77981 The subject sequence is defined as the longest initial subsequence of the input wide-character  
 77982 string, starting with the first non-white-space wide character, that is of the expected form. The  
 77983 subject sequence contains no wide-character codes if the input wide-character string is empty or  
 77984 consists entirely of white-space wide characters, or if the first non-white-space wide character is  
 77985 other than a sign or a permissible letter or digit.

77986 If the subject sequence has the expected form and `base` is 0, the sequence of wide-character codes  
 77987 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has  
 77988 the expected form and the value of `base` is between 2 and 36, it shall be used as the base for  
 77989 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a  
 77990 <hyphen-minus>, the resulting value shall be the negative of the converted value; this action  
 77991 shall be performed in the return type. A pointer to the final wide-character string shall be stored  
 77992 in the object pointed to by `endptr`, provided that `endptr` is not a null pointer.

77993 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be  
77994 accepted.

77995 If the subject sequence is empty or does not have the expected form, no conversion shall be  
77996 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that  
77997 *endptr* is not a null pointer.

77998 These functions shall not change the setting of *errno* if successful.

77999 Since 0, {ULONG\_MAX}, and {ULLONG\_MAX} are returned on error and 0 is also a valid return  
78000 on success, an application wishing to check for error situations should set *errno* to 0, then call  
78001 *wcstoul()* or *wcstoull()*, then check *errno*.

#### 78002 RETURN VALUE

78003 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted  
78004 CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to  
78005 indicate the error. If the correct value is outside the range of representable values,  
78006 {ULONG\_MAX} or {ULLONG\_MAX} respectively shall be returned and *errno* set to [ERANGE].

#### 78007 ERRORS

78008 These functions shall fail if:

78009 CX [EINVAL] The value of *base* is not supported.

78010 [ERANGE] The value to be returned is not representable.

78011 These functions may fail if:

78012 CX [EINVAL] No conversion could be performed.

#### 78013 EXAMPLES

78014 None.

#### 78015 APPLICATION USAGE

78016 None.

#### 78017 RATIONALE

78018 None.

#### 78019 FUTURE DIRECTIONS

78020 None.

#### 78021 SEE ALSO

78022 *fscanf()*, *iswalph()*, *wcstod()*, *wcstol()*

78023 XBD <[wchar.h](#)>

#### 78024 CHANGE HISTORY

78025 First released in Issue 4. Derived from the MSE working draft.

#### 78026 Issue 5

78027 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

#### 78028 Issue 6

78029 Extensions beyond the ISO C standard are marked.

78030 The following new requirements on POSIX implementations derive from alignment with the  
78031 Single UNIX Specification:

- 78032 • The [EINVAL] error condition is added for when the value of *base* is not supported.

78033 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
78034 added if no conversion could be performed.

78035 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 78036 • The *wcstoul()* prototype is updated.
- 78037 • The *wcstoull()* function is added.

78038 **Issue 7**

78039 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0727 [105] is applied.

78040 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0395 [584] and XSH/TC2-2008/0396  
78041 [796] are applied.

78042 **Issue 8**

78043 Austin Group Defect 700 is applied, clarifying how a subject sequence beginning with <hyphen-  
78044 minus> is converted.

78045 Austin Group Defect 1163 is applied, clarifying the handling of white space in the input string.

78046 **NAME**

78047       wcstoumax — convert a wide-character string to an integer type

78048 **SYNOPSIS**

78049       #include &lt;stddef.h&gt;

78050       #include &lt;inttypes.h&gt;

78051       uintmax\_t wcstoumax(const wchar\_t \*restrict *nptr*,78052                    wchar\_t \*\*restrict *endptr*, int *base*);78053 **DESCRIPTION**78054       Refer to *wcstoimax()*.

78055 **NAME**78056 `wcswidth` — number of column positions of a wide-character string78057 **SYNOPSIS**

```
78058 XSI #include <wchar.h>
78059 int wcswidth(const wchar_t *pwcs, size_t n);
```

78060 **DESCRIPTION**

78061 The `wcswidth()` function shall determine the number of column positions required for  $n$  wide-  
78062 character codes (or fewer than  $n$  wide-character codes if a null wide-character code is  
78063 encountered before  $n$  wide-character codes are exhausted) in the string pointed to by `pwcs`.

78064 The `wcswidth()` function shall not change the setting of `errno` on valid input.

78065 **RETURN VALUE**

78066 The `wcswidth()` function either shall return 0 (if `pwcs` points to a null wide-character code), or  
78067 return the number of column positions to be occupied by the wide-character string pointed to by  
78068 `pwcs`, or return -1 (if any of the first  $n$  wide-character codes in the wide-character string pointed  
78069 to by `pwcs` is not a printable wide-character code).

78070 **ERRORS**

78071 No errors are defined.

78072 **EXAMPLES**

78073 None.

78074 **APPLICATION USAGE**

78075 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the  
78076 return value for a non-printable wide character is not specified.

78077 **RATIONALE**

78078 None.

78079 **FUTURE DIRECTIONS**

78080 None.

78081 **SEE ALSO**

78082 [wctype\(\)](#)

78083 [XBD Section 3.75](#) (on page 42), [<wchar.h>](#)

78084 **CHANGE HISTORY**

78085 First released in Issue 4. Derived from the MSE working draft.

78086 **Issue 6**

78087 The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

78088 **Issue 8**

78089 Austin Group Defect 448 is applied, adding a requirement that `wcswidth()` does not change the  
78090 setting of `errno` on valid input.

78091 **NAME**

78092           wcsxfrm, wcsxfrm\_l — wide-character string transformation

78093 **SYNOPSIS**

78094           #include &lt;wchar.h&gt;

78095           size\_t wcsxfrm(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
78096                           size\_t n);78097 CX        size\_t wcsxfrm\_l(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
78098                           size\_t n, locale\_t locale);78099 **DESCRIPTION**78100 CX        For `wcsxfrm()`: The functionality described on this reference page is aligned with the ISO C  
78101 standard. Any conflict between the requirements described here and the ISO C standard is  
78102 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.78103 CX        The `wcsxfrm()` and `wcsxfrm_l()` functions shall transform the wide-character string pointed to  
78104 by `ws2` and place the resulting wide-character string into the array pointed to by `ws1`. The  
78105 transformation shall be such that if `wcscmp()` is applied to two transformed wide strings, it shall  
78106 CX        return a value greater than, equal to, or less than 0, corresponding to the result of `wcscoll()` and  
78107 `wcscoll_l()` applied to the same two original wide-character strings, and the same `LC_COLLATE`  
78108 CX        category of the current locale or the locale object `locale`, respectively. No more than `n` wide-  
78109 character codes shall be placed into the resulting array pointed to by `ws1`, including the  
78110 terminating null wide-character code. If `n` is 0, `ws1` is permitted to be a null pointer. If copying  
78111 takes place between objects that overlap, the behavior is undefined.78112 CX        The `wcsxfrm()` and `wcsxfrm_l()` functions shall not change the setting of `errno` if successful.78113        Since no return value is reserved to indicate an error, an application wishing to check for error  
78114 situations should set `errno` to 0, then call `wcsxfrm()` or `wcsxfrm_l()`, then check `errno`.78115        The behavior is undefined if the `locale` argument to `wcsxfrm_l()` is the special locale object  
78116 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.78117 **RETURN VALUE**78118 CX        The `wcsxfrm()` and `wcsxfrm_l()` functions shall return the length of the transformed wide-  
78119 character string (not including the terminating null wide-character code). If the value returned is  
78120 `n` or more, the contents of the array pointed to by `ws1` are unspecified.78121 CX        On error, the `wcsxfrm()` and `wcsxfrm_l()` functions may set `errno`, but no return value is reserved  
78122 to indicate an error.78123 **ERRORS**

78124        These functions may fail if:

78125 CX        [EINVAL]        The wide-character string pointed to by `ws2` contains wide-character codes  
78126                           outside the domain of the collating sequence.

78127 **EXAMPLES**

78128 None.

78129 **APPLICATION USAGE**

78130 The transformation function is such that two transformed wide-character strings can be ordered  
78131 by *wscmp()* as appropriate to collating sequence information in the current locale (category  
78132 *LC\_COLLATE*).

78133 The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of  
78134 the *ws1* array prior to making the transformation.

78135 **RATIONALE**

78136 None.

78137 **FUTURE DIRECTIONS**

78138 None.

78139 **SEE ALSO**78140 *wscmp()*, *wscoll()*78141 XBD <[wchar.h](#)>78142 **CHANGE HISTORY**

78143 First released in Issue 4. Derived from the MSE working draft.

78144 **Issue 5**

78145 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

78146 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.78147 **Issue 6**

78148 In earlier versions, this function was required to return -1 on error.

78149 Extensions beyond the ISO C standard are marked.

78150 The following new requirements on POSIX implementations derive from alignment with the  
78151 Single UNIX Specification:

- 78152 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is  
78153 added if no conversion could be performed.

78154 The *wcsxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.78155 **Issue 7**

78156 The *wcsxfrm\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
78157 API Set Part 4.

78158 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0728 [302], XSH/TC1-2008/0729 [283],  
78159 XSH/TC1-2008/0730 [283], and XSH/TC1-2008/0731 [302] are applied.

78160 **NAME**

78161 wctob — wide-character to single-byte conversion

78162 **SYNOPSIS**

78163 #include &lt;stdio.h&gt;

78164 #include &lt;wchar.h&gt;

78165 int wctob(wint\_t c);

78166 **DESCRIPTION**

78167 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
78168 conflict between the requirements described here and the ISO C standard is unintentional. This  
78169 volume of POSIX.1-2024 defers to the ISO C standard.

78170 The *wctob()* function shall determine whether *c* corresponds to a member of the extended  
78171 character set whose character representation is a single byte when in the initial shift state.

78172 The behavior of this function shall be affected by the *LC\_CTYPE* category of the current locale.

78173 CX The *wctob()* function shall not change the setting of *errno* on valid input.

78174 **RETURN VALUE**

78175 The *wctob()* function shall return EOF if *c* does not correspond to a character with length one in  
78176 the initial shift state. Otherwise, it shall return the single-byte representation of that character as  
78177 an **unsigned char** converted to **int**.

78178 **ERRORS**

78179 No errors are defined.

78180 **EXAMPLES**

78181 None.

78182 **APPLICATION USAGE**

78183 None.

78184 **RATIONALE**

78185 None.

78186 **FUTURE DIRECTIONS**

78187 None.

78188 **SEE ALSO**78189 [btowc\(\)](#)78190 XBD [<stdio.h>](#), [<wchar.h>](#)78191 **CHANGE HISTORY**

78192 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
78193 (E).

78194 **Issue 8**

78195 Austin Group Defect 448 is applied, adding a requirement that *wctob()* does not change the  
78196 setting of *errno* on valid input.



78197 **NAME**

78198 wctomb — convert a wide-character code to a character

78199 **SYNOPSIS**

78200 #include &lt;stdlib.h&gt;

78201 int wctomb(char \*s, wchar\_t wchar);

78202 **DESCRIPTION**

78203 CX Except for requirements relating to data races, the functionality described on this reference page  
 78204 is aligned with the ISO C standard. Any other conflict between the requirements described here  
 78205 and the ISO C standard is unintentional. This volume of POSIX.1-2024 defers to the ISO C  
 78206 standard for all *wctomb()* functionality except in relation to data races.

78207 The *wctomb()* function shall determine the number of bytes needed to represent the character  
 78208 corresponding to the wide-character code whose value is *wchar* (including any change in the  
 78209 shift state). It shall store the character representation (possibly multiple bytes and any special  
 78210 bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most  
 78211 {MB\_CUR\_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any  
 78212 shift sequence needed to restore the initial shift state, and *wctomb()* shall be left in the initial shift  
 78213 state.

78214 The behavior of this function is affected by the *LC\_CTYPE* category of the current locale. For a  
 78215 state-dependent encoding, this function shall be placed into its initial state by a call for which its  
 78216 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null  
 78217 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a  
 78218 null pointer shall cause this function to return a non-zero value if encodings have state  
 78219 dependency, and 0 otherwise. Changing the *LC\_CTYPE* category causes the shift state of this  
 78220 function to be unspecified.

78221 CX The *wctomb()* function need not be thread-safe; however, it shall avoid data races with all other  
 78222 functions.

78223 The implementation shall behave as if no function defined in this volume of POSIX.1-2024 calls  
 78224 *wctomb()*.

78225 **RETURN VALUE**

78226 If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings,  
 78227 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()*  
 78228 shall return -1 if the value of *wchar* does not correspond to a valid character, or return the  
 78229 number of bytes that constitute the character corresponding to the value of *wchar*.

78230 In no case shall the value returned be greater than the value of the {MB\_CUR\_MAX} macro.

78231 **ERRORS**78232 The *wctomb()* function shall fail if:

78233 CX [EILSEQ] An invalid wide-character code is detected.

78234 **EXAMPLES**

78235 None.

78236 **APPLICATION USAGE**

78237 None.

78238 **RATIONALE**

78239 When the ISO C standard introduced threads in C11, it required *wctomb()* to avoid data races  
78240 (with itself as well as with other functions), whereas POSIX.1-2008 did not require it to be  
78241 thread-safe, and in many implementations it did not avoid data races with itself and still does  
78242 not. The ISO C committee intend to change the requirements in a future version of the ISO C  
78243 standard, but since POSIX.1 currently refers to C17 it is necessary for it not to defer to the ISO C  
78244 standard regarding data races in order to continue to allow this function not to avoid data races  
78245 with itself.

78246 **FUTURE DIRECTIONS**

78247 It is expected that a change in a future version of the ISO C standard will allow a future version  
78248 of this standard to remove the data race exception from the statement that it defers to the ISO C  
78249 standard.

78250 **SEE ALSO**78251 [mblen\(\)](#), [mbtowc\(\)](#), [mbstowcs\(\)](#), [wcstombs\(\)](#)78252 XBD <[stdlib.h](#)>78253 **CHANGE HISTORY**

78254 First released in Issue 4. Derived from the ANSI C standard.

78255 **Issue 6**

78256 Extensions beyond the ISO C standard are marked.

78257 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

78258 **Issue 7**

78259 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

78260 **Issue 8**

78261 Austin Group Defects 708 and 1302 are applied, aligning this function with the  
78262 ISO/IEC 9899:2018 standard, except in relation to data races.

78263 Austin Group Defect 1572 is applied, removing CX shading from some text derived from the  
78264 ISO C standard.

78265 **NAME**

78266 wctrans, wctrans\_l — define character mapping

78267 **SYNOPSIS**

78268 #include &lt;wctype.h&gt;

78269 wctrans\_t wctrans(const char \*charclass);

78270 CX wctrans\_t wctrans\_l(const char \*charclass, locale\_t locale);

78271 **DESCRIPTION**78272 CX For *wctrans()*: The functionality described on this reference page is aligned with the ISO C  
78273 standard. Any conflict between the requirements described here and the ISO C standard is  
78274 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.78275 CX The *wctrans()* and *wctrans\_l()* functions are defined for valid character mapping names  
78276 identified in the current locale. The *charclass* is a string identifying a generic character mapping  
78277 name for which codeset-specific information is required. The following character mapping  
78278 names are defined in all locales: **tolower** and **toupper**.78279 These functions shall return a value of type **wctrans\_t**, which can be used as the second  
78280 CX argument to subsequent calls of *towctrans()* and *towctrans\_l()*.78281 CX The *wctrans()* and *wctrans\_l()* functions shall determine values of **wctrans\_t** according to the  
78282 CX rules of the coded character set defined by character mapping information in the current locale  
78283 or in the locale represented by *locale*, respectively (category *LC\_CTYPE*).78284 The values returned by *wctrans()* shall be valid until a call to *setlocale()* that modifies the  
78285 category *LC\_CTYPE*.78286 CX The values returned by *wctrans\_l()* shall be valid only in calls to *towctrans\_l()* with a locale  
78287 represented by *locale* with the same *LC\_CTYPE* category value.78288 The behavior is undefined if the *locale* argument to *wctrans\_l()* is the special locale object  
78289 *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.78290 **RETURN VALUE**78291 CX The *wctrans()* and *wctrans\_l()* functions shall return 0 and may set *errno* to indicate the error if  
78292 the given character mapping name is not valid for the current locale (category *LC\_CTYPE*);  
78293 otherwise, they shall return a non-zero object of type **wctrans\_t** that can be used in calls to  
78294 CX *towctrans()* and *towctrans\_l()*.78295 **ERRORS**

78296 These functions may fail if:

78297 CX [EINVAL] The character mapping name pointed to by *charclass* is not valid in the current  
78298 locale.78299 **EXAMPLES**

78300 None.

78301 **APPLICATION USAGE**

78302 None.

78303 **RATIONALE**

78304 None.

78305 **FUTURE DIRECTIONS**

78306 None.

78307 **SEE ALSO**78308 [towctrans\(\)](#)78309 XBD [<wctype.h>](#)78310 **CHANGE HISTORY**

78311 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

78312 **Issue 7**78313 The *wctrans\_l()* function is added from The Open Group Technical Standard, 2006, Extended  
78314 API Set Part 4.78315 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0732 [302], XSH/TC1-2008/0733 [289],  
78316 XSH/TC1-2008/0734 [283], and XSH/TC1-2008/0735 [283] are applied.

78317 **NAME**

78318 wctype, wctype\_l — define character class

78319 **SYNOPSIS**

78320 #include &lt;wctype.h&gt;

78321 wctype\_t wctype(const char \*property);

78322 CX wctype\_t wctype\_l(const char \*property, locale\_t locale);

78323 **DESCRIPTION**78324 CX For *wctype()*: The functionality described on this reference page is aligned with the ISO C  
78325 standard. Any conflict between the requirements described here and the ISO C standard is  
78326 unintentional. This volume of POSIX.1-2024 defers to the ISO C standard.78327 CX The *wctype()* and *wctype\_l()* functions are defined for valid character class names as defined in  
78328 CX the current locale or in the locale represented by *locale*, respectively.78329 The *property* argument is a string identifying a generic character class for which codeset-specific  
78330 type information is required. The following character class names shall be defined in all locales:

78331	<b>alnum</b>	<b>digit</b>	<b>punct</b>
78332	<b>alpha</b>	<b>graph</b>	<b>space</b>
78333	<b>blank</b>	<b>lower</b>	<b>upper</b>
78334	<b>cntrl</b>	<b>print</b>	<b>xdigit</b>

78335 Additional character class names defined in the locale definition file (category *LC\_CTYPE*) can  
78336 also be specified.78337 These functions shall return a value of type **wctype\_t**, which can be used as the second  
78338 CX argument to subsequent calls of *iswctype()* and *iswctype\_l()*.78339 CX The *wctype()* and *wctype\_l()* functions shall determine values of **wctype\_t** according to the  
78340 CX rules of the coded character set defined by character type information in the current locale or in  
78341 the locale represented by *locale*, respectively (category *LC\_CTYPE*).78342 The values returned by *wctype()* shall be valid until a call to *setlocale()* that modifies the category  
78343 *LC\_CTYPE*.78344 CX The values returned by *wctype\_l()* shall be valid only in calls to *iswctype\_l()* with a locale  
78345 represented by *locale* with the same *LC\_CTYPE* category value.78346 The behavior is undefined if the *locale* argument to *wctype\_l()* is the special locale object  
78347 *LC\_GLOBAL\_LOCALE* or is not a valid locale object handle.78348 **RETURN VALUE**78349 CX The *wctype()* and *wctype\_l()* functions shall return 0 if the given character class name is not  
78350 valid for the current locale (category *LC\_CTYPE*); otherwise, they shall return an object of type78351 CX **wctype\_t** that can be used in calls to *iswctype()* and *iswctype\_l()*.78352 **ERRORS**

78353 No errors are defined.

78354 **EXAMPLES**

78355 None.

78356 **APPLICATION USAGE**

78357 None.

78358 **RATIONALE**

78359 None.

78360 **FUTURE DIRECTIONS**

78361 None.

78362 **SEE ALSO**78363 [\*iswctype\(\)\*](#)78364 XBD [\*\*<wctype.h>\*\*](#)78365 **CHANGE HISTORY**

78366 First released in Issue 4.

78367 **Issue 5**78368 The following change has been made in this version for alignment with  
78369 ISO/IEC 9899:1990/Amendment 1:1995 (E):

- 78370
- The SYNOPSIS has been changed to indicate that this function and associated data types  
78371 are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

78372 **Issue 7**78373 The *wctype\_l()* function is added from The Open Group Technical Standard, 2006, Extended API  
78374 Set Part 4.78375 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0736 [302], XSH/TC1-2008/0737 [283],  
78376 and XSH/TC1-2008/0738 [283] are applied.

78377 **NAME**

78378           wctype — number of column positions of a wide-character code

78379 **SYNOPSIS**

```
78380 XSI       #include <wctype.h>
78381       int wctype(wchar_t wc);
```

78382 **DESCRIPTION**

78383       The *wctype()* function shall determine the number of column positions required for the wide  
78384 character *wc*. The application shall ensure that the value of *wc* is a character representable as a  
78385 **wchar\_t**, and is a wide-character code corresponding to a valid character in the current locale.

78386       The *wctype()* function shall not change the setting of *errno* on valid input.

78387 **RETURN VALUE**

78388       The *wctype()* function shall either return 0 (if *wc* is a null wide-character code), or return the  
78389 number of column positions to be occupied by the wide-character code *wc*, or return -1 (if *wc*  
78390 does not correspond to a printable wide-character code).

78391 **ERRORS**

78392       No errors are defined.

78393 **EXAMPLES**

78394       None.

78395 **APPLICATION USAGE**

78396       This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the  
78397 return value for a non-printable wide character is not specified.

78398 **RATIONALE**

78399       None.

78400 **FUTURE DIRECTIONS**

78401       None.

78402 **SEE ALSO**

78403       *wcsnwidth()*

78404       XBD <[wctype.h](#)>

78405 **CHANGE HISTORY**

78406       First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working  
78407 draft.

78408 **Issue 6**

78409       The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

78410       The normative text is updated to avoid use of the term “must” for application requirements.

78411 **Issue 8**

78412       Austin Group Defect 448 is applied, adding a requirement that *wctype()* does not change the  
78413 setting of *errno* on valid input.

78414 **NAME**

78415 wmemchr — find a wide character in memory

78416 **SYNOPSIS**

78417 #include &lt;wchar.h&gt;

78418 wchar\_t \*wmemchr(const wchar\_t \*ws, wchar\_t wc, size\_t n);

78419 **DESCRIPTION**

78420 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
78421 conflict between the requirements described here and the ISO C standard is unintentional. This  
78422 volume of POSIX.1-2024 defers to the ISO C standard.

78423 The *wmemchr()* function shall locate the first occurrence of *wc* in the initial *n* wide characters of  
78424 the object pointed to by *ws*. This function shall not be affected by locale and all **wchar\_t** values  
78425 shall be treated identically. The null wide character and **wchar\_t** values not corresponding to  
78426 valid characters shall not be treated specially.

78427 If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if  
78428 no valid occurrence of *wc* is found.

78429 CX The *wmemchr()* function shall not change the setting of *errno* on valid input.

78430 **RETURN VALUE**

78431 The *wmemchr()* function shall return a pointer to the located wide character, or a null pointer if  
78432 the wide character does not occur in the object.

78433 **ERRORS**

78434 No errors are defined.

78435 **EXAMPLES**

78436 None.

78437 **APPLICATION USAGE**

78438 None.

78439 **RATIONALE**

78440 None.

78441 **FUTURE DIRECTIONS**

78442 None.

78443 **SEE ALSO**78444 [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#), [wmemset\(\)](#)78445 XBD [<wchar.h>](#)78446 **CHANGE HISTORY**

78447 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
78448 (E).

78449 **Issue 6**

78450 The normative text is updated to avoid use of the term “must” for application requirements.

78451 **Issue 8**

78452 Austin Group Defect 448 is applied, adding a requirement that *wmemchr()* does not change the  
78453 setting of *errno* on valid input.



78454 **NAME**

78455 wmemcmp — compare wide characters in memory

78456 **SYNOPSIS**

78457 #include &lt;wchar.h&gt;

78458 int wmemcmp(const wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

78459 **DESCRIPTION**

78460 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
78461 conflict between the requirements described here and the ISO C standard is unintentional. This  
78462 volume of POSIX.1-2024 defers to the ISO C standard.

78463 The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by  
78464 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be  
78465 affected by locale and all **wchar\_t** values shall be treated identically. The null wide character and  
78466 **wchar\_t** values not corresponding to valid characters shall not be treated specially.

78467 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
78468 shall behave as if the two objects compare equal.

78469 CX The *wmemcmp()* function shall not change the setting of *errno* on valid input.

78470 **RETURN VALUE**

78471 The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,  
78472 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object  
78473 pointed to by *ws2*.

78474 **ERRORS**

78475 No errors are defined.

78476 **EXAMPLES**

78477 None.

78478 **APPLICATION USAGE**

78479 None.

78480 **RATIONALE**

78481 None.

78482 **FUTURE DIRECTIONS**

78483 None.

78484 **SEE ALSO**78485 [wmemchr\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#), [wmemset\(\)](#)78486 XBD [<wchar.h>](#)78487 **CHANGE HISTORY**

78488 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
78489 (E).

78490 **Issue 6**

78491 The normative text is updated to avoid use of the term “must” for application requirements.

78492 **Issue 8**

78493 Austin Group Defect 448 is applied, adding a requirement that *wmemcmp()* does not change the  
78494 setting of *errno* on valid input.

78495 **NAME**

78496 wmemcpy — copy wide characters in memory

78497 **SYNOPSIS**

78498 #include &lt;wchar.h&gt;

78499 wchar\_t \*wmemcpy(wchar\_t \*restrict ws1, const wchar\_t \*restrict ws2,  
78500 size\_t n);78501 **DESCRIPTION**78502 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
78503 conflict between the requirements described here and the ISO C standard is unintentional. This  
78504 volume of POSIX.1-2024 defers to the ISO C standard.78505 The *wmemcpy()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
78506 object pointed to by *ws1*. This function shall not be affected by locale and all **wchar\_t** values  
78507 shall be treated identically. The null wide character and **wchar\_t** values not corresponding to  
78508 valid characters shall not be treated specially.78509 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
78510 shall copy zero wide characters.78511 CX The *wmemcpy()* function shall not change the setting of *errno* on valid input.78512 **RETURN VALUE**78513 The *wmemcpy()* function shall return the value of *ws1*.78514 **ERRORS**

78515 No errors are defined.

78516 **EXAMPLES**

78517 None.

78518 **APPLICATION USAGE**

78519 None.

78520 **RATIONALE**

78521 None.

78522 **FUTURE DIRECTIONS**

78523 None.

78524 **SEE ALSO**78525 [wmemchr\(\)](#), [wmemcmp\(\)](#), [wmemmove\(\)](#), [wmemset\(\)](#)78526 XBD [<wchar.h>](#)78527 **CHANGE HISTORY**78528 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
78529 (E).78530 **Issue 6**

78531 The normative text is updated to avoid use of the term “must” for application requirements.

78532 The *wmemcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.78533 **Issue 8**78534 Austin Group Defect 448 is applied, adding a requirement that *wmemcpy()* does not change the  
78535 setting of *errno* on valid input.

78536 **NAME**

78537 wmemmove — copy wide characters in memory with overlapping areas

78538 **SYNOPSIS**

78539 #include &lt;wchar.h&gt;

78540 wchar\_t \*wmemmove(wchar\_t \*ws1, const wchar\_t \*ws2, size\_t n);

78541 **DESCRIPTION**

78542 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
78543 conflict between the requirements described here and the ISO C standard is unintentional. This  
78544 volume of POSIX.1-2024 defers to the ISO C standard.

78545 The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the  
78546 object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object  
78547 pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not  
78548 overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary  
78549 array are copied into the object pointed to by *ws1*.

78550 This function shall not be affected by locale and all **wchar\_t** values shall be treated identically.  
78551 The null wide character and **wchar\_t** values not corresponding to valid characters shall not be  
78552 treated specially.

78553 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function  
78554 shall copy zero wide characters.

78555 CX The *wmemmove()* function shall not change the setting of *errno* on valid input.

78556 **RETURN VALUE**78557 The *wmemmove()* function shall return the value of *ws1*.78558 **ERRORS**

78559 No errors are defined

78560 **EXAMPLES**

78561 None.

78562 **APPLICATION USAGE**

78563 None.

78564 **RATIONALE**

78565 None.

78566 **FUTURE DIRECTIONS**

78567 None.

78568 **SEE ALSO**78569 [wmemchr\(\)](#), [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemset\(\)](#)78570 XBD [<wchar.h>](#)78571 **CHANGE HISTORY**

78572 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
78573 (E).

78574 **Issue 6**

78575 The normative text is updated to avoid use of the term “must” for application requirements.

78576 **Issue 8**

78577

78578

Austin Group Defect 448 is applied, adding a requirement that *wmemmove()* does not change the setting of *errno* on valid input.

78579 **NAME**

78580 wmemset — set wide characters in memory

78581 **SYNOPSIS**

78582 #include &lt;wchar.h&gt;

78583 wchar\_t \*wmemset(wchar\_t \*ws, wchar\_t wc, size\_t n);

78584 **DESCRIPTION**

78585 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 78586 conflict between the requirements described here and the ISO C standard is unintentional. This  
 78587 volume of POSIX.1-2024 defers to the ISO C standard.

78588 The *wmemset()* function shall copy the value of *wc* into each of the first *n* wide characters of the  
 78589 object pointed to by *ws*. This function shall not be affected by locale and all **wchar\_t** values shall  
 78590 be treated identically. The null wide character and **wchar\_t** values not corresponding to valid  
 78591 characters shall not be treated specially.

78592 If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy  
 78593 zero wide characters.

78594 CX The *wmemset()* function shall not change the setting of *errno* on valid input.

78595 **RETURN VALUE**78596 The *wmemset()* functions shall return the value of *ws*.78597 **ERRORS**

78598 No errors are defined.

78599 **EXAMPLES**

78600 None.

78601 **APPLICATION USAGE**

78602 None.

78603 **RATIONALE**

78604 None.

78605 **FUTURE DIRECTIONS**

78606 None.

78607 **SEE ALSO**78608 [wmemchr\(\)](#), [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#)78609 XBD <[wchar.h](#)>78610 **CHANGE HISTORY**

78611 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
 78612 (E).

78613 **Issue 6**

78614 The normative text is updated to avoid use of the term “must” for application requirements.

78615 **Issue 8**

78616 Austin Group Defect 448 is applied, adding a requirement that *wmemset()* does not change the  
 78617 setting of *errno* on valid input.

78618 **NAME**  
 78619 wordexp, wordfree — perform word expansions

78620 **SYNOPSIS**

```
78621 #include <wordexp.h>
78622 int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
78623             int flags);
78624 void wordfree(wordexp_t *pwordexp);
```

78625 **DESCRIPTION**

78626 The *wordexp()* function shall perform word expansions as described in XCU Section 2.6 (on page  
 78627 2483), subject to quoting as described in XCU Section 2.2 (on page 2472), and place the list of  
 78628 expanded words into the structure pointed to by *pwordexp*.

78629 The *words* argument is a pointer to a string containing one or more words to be expanded. The  
 78630 expansions shall be the same as would be performed by the command line interpreter if *words*  
 78631 were the part of a command line representing the arguments to a utility. Therefore, the  
 78632 application shall ensure that *words* does not contain an unquoted <newline> character or any of  
 78633 the unquoted shell special characters '|', '&', ';', '<', '>' except in the context of command  
 78634 substitution as specified in XCU Section 2.6.3 (on page 2489). It also shall not contain unquoted  
 78635 parentheses or braces, except in the context of command or variable substitution. The  
 78636 application shall ensure that every member of *words* which it expects to have expanded by  
 78637 *wordexp()* does not contain an unquoted initial comment character. The application shall also  
 78638 ensure that any words which it intends to be ignored (because they begin or continue a  
 78639 comment) are deleted from *words*. If the argument *words* contains an unquoted comment  
 78640 character (<number-sign>) that is the beginning of a token, *wordexp()* shall either treat the  
 78641 comment character as a regular character, or interpret it as a comment indicator and ignore the  
 78642 remainder of *words*.

78643 The structure type **wordexp\_t** is defined in the <**wordexp.h**> header and includes at least the  
 78644 following members:

Member Type	Member Name	Description
size_t	<i>we_wordc</i>	Count of words matched by <i>words</i> .
char **	<i>we_wordv</i>	Pointer to list of expanded words.
size_t	<i>we_offs</i>	Slots to reserve at the beginning of <i>pwordexp-&gt;we_wordv</i> .

78650 The *wordexp()* function shall store the number of generated words into *pwordexp->we\_wordc* and  
 78651 a pointer to a list of pointers to words in *pwordexp->we\_wordv*. Each individual field created  
 78652 during field splitting (see XCU Section 2.6.5, on page 2491) or pathname expansion (see XCU  
 78653 Section 2.6.6, on page 2493) shall be a separate word in the *pwordexp->we\_wordv* list. The words  
 78654 shall be in order as described in XCU Section 2.6 (on page 2483). The first pointer after the last  
 78655 word pointer shall be a null pointer. The expansion of special parameters described in XCU  
 78656 Section 2.5.2 (on page 2479) is unspecified.

78657 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()*  
 78658 function shall allocate other space as needed, including memory pointed to by  
 78659 *pwordexp->we\_wordv*. The *wordfree()* function shall free any memory associated with *pwordexp*  
 78660 from a previous call to *wordexp()*. The *wordfree()* function shall not modify *errno* if *pwordexp* was  
 78661 previously modified by *wordexp()* and not yet freed.

78662 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the bitwise-  
 78663 inclusive OR of zero or more of the following constants, which are defined in <**wordexp.h**>:

78664	WRDE_APPEND	Append words generated to the ones from a previous call to <i>wordexp()</i> .
78665	WRDE_DOOFFS	Make use of <i>pwordexp-&gt;we_offs</i> . If this flag is set, <i>pwordexp-&gt;we_offs</i> is used to specify how many null pointers to add to the beginning of <i>pwordexp-&gt;we_wordv</i> . In other words, <i>pwordexp-&gt;we_wordv</i> shall point to <i>pwordexp-&gt;we_offs</i> null pointers, followed by <i>pwordexp-&gt;we_wordc</i> word pointers, followed by a null pointer.
78666		
78667		
78668		
78669		
78670	WRDE_NOCMD	If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2024, fail if command substitution, as specified in XCU Section 2.6.3 (on page 2489), is requested.
78671		
78672		
78673	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.
78674		
78675		
78676		
78677	WRDE_SHOWERR	Do not redirect <i>stderr</i> to <b>/dev/null</b> .
78678	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.
78679		
78680		The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to <i>wordexp()</i> . The following rules apply to applications when two or more calls to <i>wordexp()</i> are made with the same value of <i>pwordexp</i> and without intervening calls to <i>wordfree()</i> :
78681		
78682		1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.
78683		2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.
78684		3. After the second and each subsequent call, <i>pwordexp-&gt;we_wordv</i> shall point to a list containing the following:
78685		
78686		a. Zero or more null pointers, as specified by WRDE_DOOFFS and <i>pwordexp-&gt;we_offs</i>
78687		
78688		b. Pointers to the words that were in the <i>pwordexp-&gt;we_wordv</i> list before the call, in the same order as before
78689		
78690		c. Pointers to the new words generated by the latest call, in the specified order
78691		4. The count returned in <i>pwordexp-&gt;we_wordc</i> shall be the total number of words from all of the calls.
78692		
78693		5. The application can change any of the fields after a call to <i>wordexp()</i> , but if it does it shall reset them to the original value before a subsequent call, using the same <i>pwordexp</i> value, to <i>wordfree()</i> or <i>wordexp()</i> with the WRDE_APPEND or WRDE_REUSE flag.
78694		
78695		
78696		If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2024, and <i>words</i> contains an unquoted character—<newline>, ' ', '&', ';', '<', '>', '(', ')', '{', '}'—in an inappropriate context, <i>wordexp()</i> shall fail, and the number of expanded words shall be 0.
78697		
78698		
78699		
78700		Unless WRDE_SHOWERR is set in <i>flags</i> , <i>wordexp()</i> shall redirect <i>stderr</i> to <b>/dev/null</b> for any utilities executed as a result of command substitution while expanding <i>words</i> . If WRDE_SHOWERR is set, <i>wordexp()</i> may write messages to <i>stderr</i> if syntax errors are detected while expanding <i>words</i> , unless the <i>stderr</i> stream has wide orientation in which case the behavior is undefined. It is unspecified whether any write errors encountered while outputting such messages will affect the <i>stderr</i> error indicator or the value of <i>errno</i> .
78701		
78702		
78703		
78704		
78705		
78706		The application shall ensure that if WRDE_DOOFFS is set, then <i>pwordexp-&gt;we_offs</i> has the same

78707 value for each *wordexp()* call and *wordfree()* call using a given *pwordexp*.

78708 The results are unspecified if WRDE\_APPEND and WRDE\_REUSE are both specified.

78709 The following constants are defined as error return values:

78710 WRDE\_BADCHAR One of the unquoted characters—<newline>, '|', '&', ';', '<', '>',  
78711 '(', ')', '{', '}'—appears in *words* in an inappropriate context.

78712 WRDE\_BADVAL Reference to undefined shell variable when WRDE\_UNDEF is set in *flags*.

78713 WRDE\_CMDSUB Command substitution requested when WRDE\_NOCMD was set in *flags*.

78714 WRDE\_NOSPACE Attempt to allocate memory failed.

78715 WRDE\_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated  
78716 string.

#### 78717 RETURN VALUE

78718 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described  
78719 in <*wordexp.h*>, shall be returned to indicate an error. If *wordexp()* returns the value  
78720 WRDE\_NOSPACE, then *pwordexp->we\_wordc* and *pwordexp->we\_wordv* shall be updated to  
78721 reflect any words that were successfully expanded. In other error cases, if the WRDE\_APPEND  
78722 flag was specified, *pwordexp->we\_wordc* and *pwordexp->we\_wordv* shall not be modified.

78723 The *wordfree()* function shall not return a value.

#### 78724 ERRORS

78725 No errors are defined.

#### 78726 EXAMPLES

78727 None.

#### 78728 APPLICATION USAGE

78729 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's  
78730 expansions on a word or words obtained from a user. For example, if the application prompts  
78731 for a pathname (or list of pathnames) and then uses *wordexp()* to process the input, the user  
78732 could respond with anything that would be valid as input to the shell.

78733 The WRDE\_NOCMD flag is provided for applications that, for security or other reasons, want to  
78734 prevent a user from executing shell commands. Disallowing unquoted shell special characters  
78735 also prevents unwanted side-effects, such as executing a command or writing a file.

78736 POSIX.1-2024 does not require the *wordexp()* function to be thread-safe if passed an expression  
78737 referencing an environment variable while any other thread is concurrently modifying any  
78738 environment variable; see *exec* (on page 866).

78739 Even though the WRDE\_SHOWERR flag allows the implementation to write messages to *stderr*  
78740 during command substitution or syntax errors, this standard does not provide any way to detect  
78741 write failures during the output of such messages.

78742 Applications which use wide-character output functions with *stderr* should ensure that any calls  
78743 to *wordexp()* do not write to *stderr*, by avoiding use of the WRDE\_SHOWERR flag.

#### 78744 RATIONALE

78745 This function was included as an alternative to *glob()*. There had been continuing controversy  
78746 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*  
78747 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*  
78748 (which is faster, but which only performs pathname expansion, without tilde or parameter  
78749 expansion) this will satisfy the majority of applications.



78750 While *wordexp()* could be implemented entirely as a library routine, it is expected that most  
78751 implementations run a shell in a subprocess to do the expansion.

78752 Two different approaches have been proposed for how the required information might be  
78753 presented to the shell and the results returned. They are presented here as examples.

78754 One proposal is to extend the *echo* utility by adding a `-q` option. This option would cause *echo* to  
78755 add a `<backslash>` before each `<backslash>` and `<blank>` that occurs within an argument. The  
78756 *wordexp()* function could then invoke the shell as follows:

```
78757 (void) strcpy(buffer, "echo -q");
78758 (void) strcat(buffer, words);
78759 if ((flags & WRDE_SHOWERR) == 0)
78760     (void) strcat(buffer, "2>/dev/null");
78761 f = popen(buffer, "r");
```

78762 The *wordexp()* function would read the resulting output, remove unquoted `<backslash>`  
78763 characters, and break into words at unquoted `<blank>` characters. If the `WRDE_NOCMD` flag  
78764 was set, *wordexp()* would have to scan *words* before starting the subshell to make sure that there  
78765 would be no command substitution. In any case, it would have to scan *words* for unquoted  
78766 special characters.

78767 Another proposal is to add the following options to *sh*:

78768 `-w wordlist`

78769 This option provides a wordlist expansion service to applications. The words in *wordlist*  
78770 shall be expanded and the following written to standard output:

- 78771 1. The count of the number of words after expansion, in decimal, followed by a null  
78772 byte
- 78773 2. The number of bytes needed to represent the expanded words (not including null  
78774 separators), in decimal, followed by a null byte
- 78775 3. The expanded words, each terminated by a null byte

78776 If an error is encountered during word expansion, *sh* exits with a non-zero status after  
78777 writing the former to report any words successfully expanded

78778 `-P` Run in “protected” mode. If specified with the `-w` option, no command substitution shall  
78779 be performed.

78780 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess  
78781 using *fork()* and executing *sh* using the line:

```
78782 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

78783 after directing standard error to `/dev/null`.

78784 It seemed objectionable for a library routine to write messages to standard error, unless explicitly  
78785 requested, so *wordexp()* is required to redirect standard error to `/dev/null` to ensure that no  
78786 messages are generated, even for commands executed for command substitution. The  
78787 `WRDE_SHOWERR` flag can be specified to request that error messages be written.

78788 The `WRDE_REUSE` flag allows the implementation to avoid the expense of freeing and  
78789 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when  
78790 `WRDE_REUSE` is set.

78791 **FUTURE DIRECTIONS**

78792 None.

78793 **SEE ALSO**78794 *exec*, *fnmatch()*, *glob()*78795 XBD <[wordexp.h](#)>78796 XCU [Chapter 2](#) (on page 2472)78797 **CHANGE HISTORY**

78798 First released in Issue 4. Derived from the ISO POSIX-2 standard.

78799 **Issue 5**

78800 Moved from POSIX2 C-language Binding to BASE.

78801 **Issue 6**

78802 The normative text is updated to avoid use of the term “must” for application requirements.

78803 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the  
78804 ISO/IEC 9899:1999 standard.78805 **Issue 7**

78806 Austin Group Interpretation 1003.1-2001 #148 is applied, adding APPLICATION USAGE.

78807 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0739 [460], XSH/TC1-2008/0740 [291],  
78808 and XSH/TC1-2008/0741 [460] are applied.78809 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0397 [608], XSH/TC2-2008/0398 [704],  
78810 XSH/TC2-2008/0399 [704], and XSH/TC2-2008/0400 [608] are applied.78811 **Issue 8**78812 Austin Group Defect 385 is applied, adding a requirement that *wordfree()* does not modify *errno*  
78813 when passed a pointer to a **wordexp\_t** that can be freed.

78814 **NAME**

78815           wprintf — print formatted wide-character output

78816 **SYNOPSIS**

78817           #include &lt;stdio.h&gt;

78818           #include &lt;wchar.h&gt;

78819           int wprintf(const wchar\_t \*restrict *format*, ...);78820 **DESCRIPTION**78821           Refer to *fwprintf()*.

78822 **NAME**

78823 pwrite, write — write on a file

78824 **SYNOPSIS**

```
78825 #include <unistd.h>
78826 ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
78827               off_t offset);
78828 ssize_t write(int fildes, const void *buf, size_t nbyte);
```

78829 **DESCRIPTION**

78830 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the  
78831 file associated with the open file descriptor, *fildes*.

78832 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the  
78833 *write()* function may detect and return errors as described below. In the absence of errors, or if  
78834 error detection is not performed, the *write()* function shall return zero and have no other results.  
78835 If *nbyte* is zero and the file is not a regular file, the results are unspecified.

78836 On a regular file or other file capable of seeking, the actual writing of data shall proceed from  
78837 the position in the file indicated by the file offset associated with *fildes*. Before successful return  
78838 from *write()*, the file offset shall be incremented by the number of bytes actually written. On a  
78839 regular file, if the position of the last byte written is greater than or equal to the length of the file,  
78840 the length of the file shall be set to this position plus one.

78841 On a file not capable of seeking, writing shall always take place starting at the current position.  
78842 The value of a file offset associated with such a device is undefined.

78843 If the O\_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file  
78844 prior to each write and no intervening file modification operation shall occur between changing  
78845 the file offset and the write operation.

78846 If a *write()* requests that more bytes be written than there is room for (for example, the file size  
78847 limit of the process or the physical end of a medium), only as many bytes as there is room for  
78848 shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a  
78849 limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would  
78850 give a failure return (except as noted below).

78851 If the request would cause the file size to exceed the soft file size limit for the process and there  
78852 XSI is no room for any bytes to be written, the request shall fail and the implementation shall  
78853 generate a SIGXFSZ signal for the thread.

78854 If *write()* is interrupted by a signal before it writes any data, it shall return  $-1$  with *errno* set to  
78855 [EINTR].

78856 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the  
78857 number of bytes written.

78858 If the value of *nbyte* is greater than {SSIZE\_MAX}, the result is implementation-defined.

78859 After a *write()* to a regular file has successfully returned:

- 78860 • Any successful *read()* from each byte position in the file that was modified by that write  
78861 shall return the data specified by the *write()* for that position until such byte positions are  
78862 again modified.
- 78863 • Any subsequent successful *write()* to the same byte position in the file shall overwrite that  
78864 file data.

78865 Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the  
78866 following exceptions:

- 78867 • There is no file offset associated with a pipe or FIFO, hence each write request shall  
78868 append to the end of the pipe or FIFO.
- 78869 • Write requests of {PIPE\_BUF} bytes or less shall not be interleaved with data from other  
78870 threads performing write operations on the same pipe or FIFO. Writes of greater than  
78871 {PIPE\_BUF} bytes may have data interleaved, on arbitrary boundaries, with write  
78872 operations by other threads, whether or not the O\_NONBLOCK flag of the file status flags  
78873 is set.
- 78874 • If the O\_NONBLOCK flag is clear, a write request may cause the thread to block, but on  
78875 normal completion it shall return *nbyte*.
- 78876 • If the O\_NONBLOCK flag is set, *write()* requests shall be handled differently, in the  
78877 following ways:
  - 78878 — The *write()* function shall not block the thread.
  - 78879 — A write request for {PIPE\_BUF} or fewer bytes shall have the following effect: if there  
78880 is sufficient space available in the pipe or FIFO, *write()* shall transfer all the data and  
78881 return the number of bytes requested. Otherwise, *write()* shall transfer no data and  
78882 return  $-1$  with *errno* set to [EAGAIN].
  - 78883 — A write request for more than {PIPE\_BUF} bytes shall cause one of the following:
    - 78884 — When at least one byte can be written, transfer what it can and return the  
78885 number of bytes written. When all data previously written to the pipe or FIFO  
78886 is read, it shall transfer at least {PIPE\_BUF} bytes.
    - 78887 — When no data can be written, transfer no data, and return  $-1$  with *errno* set to  
78888 [EAGAIN].

78889 When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-  
78890 blocking writes and cannot accept the data immediately:

- 78891 • If the O\_NONBLOCK flag is clear, *write()* shall block the calling thread until the data can  
78892 be accepted.
- 78893 • If the O\_NONBLOCK flag is set, *write()* shall not block the thread. If some data can be  
78894 written without blocking the thread, *write()* shall write what it can and return the number  
78895 of bytes written. Otherwise, it shall return  $-1$  and set *errno* to [EAGAIN].

78896 Upon successful completion, where *nbyte* is greater than 0, *write()* shall mark for update the last  
78897 data modification and last file status change timestamps of the file, and if the file is a regular file,  
78898 the S\_ISUID and S\_ISGID bits of the file mode may be cleared.

78899 For regular files, no data transfer shall occur past the offset maximum established in the open  
78900 file description associated with *fildev*.

78901 If *fildev* refers to a socket, *write()* shall be equivalent to *send()* with no flags set.

78902 SIO If the O\_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as  
78903 defined by synchronized I/O data integrity completion.

78904 If the O\_SYNC bit has been set, write I/O operations on the file descriptor shall complete as  
78905 defined by synchronized I/O file integrity completion.

78906 SHM If *fildev* refers to a shared memory object, the result of the *write()* function is unspecified.

78907	TYM	If <i>fildest</i> refers to a typed memory object, the result of the <i>write()</i> function is unspecified.
78908		The <i>pwrite()</i> function shall be equivalent to <i>write()</i> , except that it writes into a given position and does not change the file offset (regardless of whether <code>O_APPEND</code> is set). The first three arguments to <i>pwrite()</i> are the same as <i>write()</i> with the addition of a fourth argument <i>offset</i> for the desired position inside the file. An attempt to perform a <i>pwrite()</i> on a file that is incapable of seeking shall result in an error.
78909		
78910		
78911		
78912		
78913		<b>RETURN VALUE</b>
78914		Upon successful completion, these functions shall return the number of bytes actually written to the file associated with <i>fildest</i> . This number shall never be greater than <i>nbyte</i> . Otherwise, <code>-1</code> shall be returned and <i>errno</i> set to indicate the error.
78915		
78916		
78917		<b>ERRORS</b>
78918		These functions shall fail if:
78919		[EAGAIN] The file is neither a pipe, nor a FIFO, nor a socket, the <code>O_NONBLOCK</code> flag is set for the file descriptor, and the thread would be delayed in the <i>write()</i> operation.
78920		
78921		
78922		[EBADF] The <i>fildest</i> argument is not a valid file descriptor open for writing.
78923		[EFBIG] An attempt was made to write a file that exceeds the implementation-defined maximum file size and there was no room for any bytes to be written.
78924		
78925		[EFBIG] An attempt was made to write a file that exceeds the file size limit of the process, and there was no room for any bytes to be written. A <code>SIGXFSZ</code> signal shall also be generated for the thread.
78926	XSI	
78927		
78928		[EFBIG] The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fildest</i> .
78929		
78930		
78931		[EINTR] The write operation was terminated due to the receipt of a signal, and no data was transferred.
78932		
78933		[EIO] The process is a member of a background process group attempting to write to its controlling terminal, <code>TOSTOP</code> is set, the calling thread is not blocking <code>SIGTTOU</code> , the process is not ignoring <code>SIGTTOU</code> , and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
78934		
78935		
78936		
78937		
78938		[ENOSPC] There was no free space remaining on the device containing the file.
78939		The <i>pwrite()</i> function shall fail if:
78940		[EINVAL] The file is a regular file or block special file, and the <i>offset</i> argument is negative. The file offset shall remain unchanged.
78941		
78942		[ESPIPE] The file is incapable of seeking.
78943		The <i>write()</i> function shall fail if:
78944		[EAGAIN] The file is a pipe or FIFO, the <code>O_NONBLOCK</code> flag is set for the file descriptor, and the thread would be delayed in the write operation.
78945		
78946		[EAGAIN] or [EWOULDBLOCK]
78947		The file is a socket, the <code>O_NONBLOCK</code> flag is set for the file descriptor, and the thread would be delayed in the write operation.
78948		

- 78949 [ECONNRESET] A write was attempted on a socket that is not connected.
- 78950 [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
78951 any process, or that only has one end open. A SIGPIPE signal shall also be sent  
78952 to the thread.
- 78953 [EPIPE] A write was attempted on a socket that is shut down for writing, or is no  
78954 longer connected. In the latter case, if the socket is of type SOCK\_STREAM, a  
78955 SIGPIPE signal shall also be sent to the thread.
- 78956 These functions may fail if:
- 78957 [EIO] A physical I/O error has occurred.
- 78958 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 78959 [ENXIO] A request was made of a nonexistent device, or the request was outside the  
78960 capabilities of the device.
- 78961 The *write()* function may fail if:
- 78962 [EACCES] A write was attempted on a socket and the calling process does not have  
78963 appropriate privileges.
- 78964 [ENETDOWN] A write was attempted on a socket and the local network interface used to  
78965 reach the destination is down.
- 78966 [ENETUNREACH]  
78967 A write was attempted on a socket and no route to the network is present.

## 78968 EXAMPLES

### 78969 Writing from a Buffer

78970 The following example writes data from the buffer pointed to by *buf* to the file associated with  
78971 the file descriptor *fd*.

```
78972 #include <sys/types.h>
78973 #include <string.h>
78974 ...
78975 char buf[20];
78976 size_t nbytes;
78977 ssize_t bytes_written;
78978 int fd;
78979 ...
78980 strcpy(buf, "This is a test\n");
78981 nbytes = strlen(buf);
78982 bytes_written = write(fd, buf, nbytes);
78983 ...
```

## 78984 APPLICATION USAGE

78985 None.

## 78986 RATIONALE

78987 See also the RATIONALE section in *read()*.

78988 An attempt to write to a pipe or FIFO has several major characteristics:

78989 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not  
 78990 interleaved with data from any other thread. This is useful when there are multiple  
 78991 writers sending data to a single reader. Applications need to know how large a write  
 78992 request can be expected to be performed atomically. This maximum is called {PIPE\_BUF}.  
 78993 This volume of POSIX.1-2024 does not say whether write requests for more than  
 78994 {PIPE\_BUF} bytes are atomic, but requires that writes of {PIPE\_BUF} or fewer bytes shall  
 78995 be atomic.

78996 • *Blocking/immediate*: Blocking is only possible with O\_NONBLOCK clear. If there is enough  
 78997 space for all the data requested to be written immediately, the implementation should do  
 78998 so. Otherwise, the calling thread may block; that is, pause until enough space is available  
 78999 for writing. The effective size of a pipe or FIFO (the maximum amount that can be written  
 79000 in one operation without blocking) may vary dynamically, depending on the  
 79001 implementation, so it is not possible to specify a fixed value for it.

79002 • *Complete/partial/deferred*: A write request:

```
79003 int fildes;  
79004 size_t nbyte;  
79005 ssize_t ret;  
79006 char *buf;  
  
79007 ret = write(fildes, buf, nbyte);
```

79008 may return:

79009 Complete *ret=nbyte*

79010 Partial *ret<nbyte*

79011 This shall never happen if *nbyte* ≤ {PIPE\_BUF}. If it does happen (with  
 79012 *nbyte* > {PIPE\_BUF}), this volume of POSIX.1-2024 does not guarantee  
 79013 atomicity, even if *ret* ≤ {PIPE\_BUF}, because atomicity is guaranteed according  
 79014 to the amount *requested*, not the amount *written*.

79015 Deferred: *ret=-1, errno=[EAGAIN]*

79016 This error indicates that a later request may succeed. It does not indicate that  
 79017 it *shall* succeed, even if *nbyte* ≤ {PIPE\_BUF}, because if no process reads from  
 79018 the pipe or FIFO, the write never succeeds. An application could usefully  
 79019 count the number of times [EAGAIN] is caused by a particular value of  
 79020 *nbyte* > {PIPE\_BUF} and perhaps do later writes with a smaller value, on the  
 79021 assumption that the effective size of the pipe or FIFO may have decreased.

79022 Partial and deferred writes are only possible with O\_NONBLOCK set.

79023 The relations of these properties are shown in the following tables:

Write to a Pipe or FIFO with O_NONBLOCK clear			
Immediately Writable:	None	Some	<i>nbyte</i>
<i>nbyte</i> ≤ {PIPE_BUF}	Atomic blocking <i>nbyte</i>	Atomic blocking <i>nbyte</i>	Atomic immediate <i>nbyte</i>
<i>nbyte</i> > {PIPE_BUF}	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>

79029 If the O\_NONBLOCK flag is clear, a write request shall block if the amount writable  
 79030 immediately is less than that requested. If the flag is set (by *fcntl()*), a write request shall never  
 79031 block.



79032  
79033  
79034  
79035  
79036

Write to a Pipe or FIFO with O_NONBLOCK set			
Immediately Writable:	None	Some	<i>nbyte</i>
<i>nbyte</i> ≤ {PIPE_BUF}	-1, [EAGAIN]	-1, [EAGAIN]	Atomic <i>nbyte</i>
<i>nbyte</i> > {PIPE_BUF}	-1, [EAGAIN]	< <i>nbyte</i> or -1, [EAGAIN]	≤ <i>nbyte</i> or -1, [EAGAIN]

79037  
79038  
79039  
79040  
79041  
79042

There is no exception regarding partial writes when O\_NONBLOCK is set. With the exception of writing to an empty pipe or FIFO, this volume of POSIX.1-2024 does not specify exactly when a partial write is performed since that would require specifying internal details of the implementation. Every application should be prepared to handle partial writes when O\_NONBLOCK is set and the requested amount is greater than {PIPE\_BUF}, just as every application should be prepared to handle partial writes on other kinds of file descriptors.

79043  
79044  
79045

The intent of forcing writing at least one byte if any can be written is to assure that each write makes progress if there is any room in the pipe or FIFO. If the pipe or FIFO is empty, {PIPE\_BUF} bytes must be written; if not, at least some progress must have been made.

79046  
79047  
79048  
79049  
79050  
79051  
79052  
79053  
79054

Where this volume of POSIX.1-2024 requires -1 to be returned and *errno* set to [EAGAIN], most historical implementations return zero (with the O\_NDELAY flag set, which is the historical predecessor of O\_NONBLOCK, but is not itself in this volume of POSIX.1-2024). The error indications in this volume of POSIX.1-2024 were chosen so that an application can distinguish these cases from end-of-file. While *write()* cannot receive an indication of end-of-file, *read()* can, and the two functions have similar return values. Also, some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file indication; for those systems, a return value of zero from *write()* indicates a successful write of an end-of-file indication.

79055  
79056

Implementations are allowed, but not required, to perform error checking for *write()* requests of zero bytes.

79057  
79058  
79059

The concept of a {PIPE\_MAX} limit (indicating the maximum number of bytes that can be written to a pipe or FIFO in a single operation) was considered, but rejected, because this concept would unnecessarily limit application writing.

79060

See also the discussion of O\_NONBLOCK in *read()*.

79061  
79062  
79063  
79064  
79065  
79066

Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the calls are made by different threads. A similar requirement applies to multiple write operations to the same file position. This is needed to guarantee the propagation of data from *write()* calls to subsequent *read()* calls. This requirement is particularly significant for networked file systems, where some caching schemes violate these semantics.

79067  
79068  
79069  
79070  
79071

Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writew()* also obey these semantics. A new “high-performance” write analog that did not follow these serialization requirements would also be permitted by this wording. This volume of POSIX.1-2024 is also silent about any effects of application-level caching (such as that done by *stdio*).

79072  
79073  
79074  
79075  
79076

This volume of POSIX.1-2024 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

79077

This volume of POSIX.1-2024 does not specify the behavior of concurrent writes to a regular file

79078 from multiple threads, except that each write is atomic (see [Section 2.9.7](#), on page 547).  
79079 Applications should use some form of concurrency control.

79080 This volume of POSIX.1-2024 intentionally does not specify any *pwrite()* errors related to pipes,  
79081 FIFOs, and sockets other than [ESPIPE].

#### 79082 FUTURE DIRECTIONS

79083 None.

#### 79084 SEE ALSO

79085 *chmod()*, *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *read()*, *writev()*

79086 XBD [<limits.h>](#), [<sys/uio.h>](#), [<unistd.h>](#)

#### 79087 CHANGE HISTORY

79088 First released in Issue 1. Derived from Issue 1 of the SVID.

#### 79089 Issue 5

79090 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX  
79091 Threads Extension.

79092 Large File Summit extensions are added.

79093 The *pwrite()* function is added.

#### 79094 Issue 6

79095 The DESCRIPTION states that the *write()* function does not block the thread. Previously this  
79096 said “process” rather than “thread”.

79097 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are  
79098 marked as part of the XSI STREAMS Option Group.

79099 The following new requirements on POSIX implementations derive from alignment with the  
79100 Single UNIX Specification:

79101 • The DESCRIPTION now states that if *write()* is interrupted by a signal after it has  
79102 successfully written some data, it returns the number of bytes written. In the POSIX.1-1988  
79103 standard, it was optional whether *write()* returned the number of bytes written, or whether  
79104 it returned  $-1$  with *errno* set to [EINTR]. This is a FIPS requirement.

79105 • The following changes are made to support large files:

79106 — For regular files, no data transfer occurs past the offset maximum established in the  
79107 open file description associated with the *fildev*.

79108 — A second [EFBIG] error condition is added.

79109 • The [EIO] error condition is added.

79110 • The [EPIPE] error condition is added for when a pipe has only one end open.

79111 • The [ENXIO] optional error condition is added.

79112 Text referring to sockets is added to the DESCRIPTION.

79113 The following changes were made to align with the IEEE P1003.1a draft standard:

79114 • The effect of reading zero bytes is clarified.

79115 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
79116 *write()* results are unspecified for typed memory objects.

79117 The following error conditions are added for operations on sockets: [EAGAIN],

- 79118 [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].
- 79119 The [EIO] error is made optional.
- 79120 The [ENOBUFS] error is added for sockets.
- 79121 The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN],  
79122 and [ENETUNREACH].
- 79123 The *writenv()* function is split out into a separate reference page.
- 79124 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/146 is applied, updating text in the  
79125 ERRORS section from “a SIGPIPE signal is generated to the calling process” to “a SIGPIPE  
79126 signal shall also be sent to the thread”.
- 79127 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/147 is applied, making a correction to the  
79128 RATIONALE.
- 79129 **Issue 7**
- 79130 The *pwrite()* function is moved from the XSI option to the Base.
- 79131 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 79132 SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the  
79133 *pwrite()* function, and to change the use of the phrase “file pointer” to “file offset”.
- 79134 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0742 [219], XSH/TC1-2008/0743 [215],  
79135 XSH/TC1-2008/0744 [79], and XSH/TC1-2008/0745 [215] are applied.
- 79136 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0401 [676,710] and  
79137 XSH/TC2-2008/0402 [966] are applied.
- 79138 **Issue 8**
- 79139 Austin Group Defect 308 is applied, clarifying the handling of [EFBIG] errors.
- 79140 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 79141 Austin Group Defect 1430 is applied, clarifying that requirements relating to data interleaving  
79142 on pipes and FIFOs apply to write operations in other threads, not just other processes, and  
79143 changing some uses of “pipe” to “pipe or FIFO”.
- 79144 Austin Group Defect 1669 is applied, removing XSI shading from part of the [EFBIG] error  
79145 relating to the file size limit for the process.

79146 **NAME**

79147 writev — write a vector

79148 **SYNOPSIS**

```
79149 XSI #include <sys/uio.h>
79150      ssize_t writev(int fildev, const struct iovec *iov, int iovcnt);
```

79151 **DESCRIPTION**

79152 The *writev()* function shall be equivalent to *write()*, except as described below. The *writev()*  
 79153 function shall gather output data from the *iovcnt* buffers specified by the members of the *iov*  
 79154 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than  
 79155 or equal to {IOV\_MAX}, as defined in <limits.h>.

79156 Each *iovec* entry specifies the base address and length of an area in memory from which data  
 79157 should be written. The *writev()* function shall always write a complete area before proceeding to  
 79158 the next.

79159 If *fildev* refers to a regular file and all of the *iov\_len* members in the array pointed to by *iov* are 0,  
 79160 *writev()* shall return 0 and have no other effect. For other file types, the behavior is unspecified.

79161 If the sum of the *iov\_len* values is greater than {SSIZE\_MAX}, the operation shall fail and no data  
 79162 shall be transferred.

79163 **RETURN VALUE**

79164 Upon successful completion, *writev()* shall return the number of bytes actually written.  
 79165 Otherwise, it shall return a value of -1, the file-pointer shall remain unchanged, and *errno* shall  
 79166 be set to indicate an error.

79167 **ERRORS**79168 Refer to *write()*.79169 In addition, the *writev()* function shall fail if:79170 [EINVAL] The sum of the *iov\_len* values in the *iov* array would overflow an *ssize\_t*.79171 The *writev()* function may fail and set *errno* to:79172 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV\_MAX}.79173 **EXAMPLES**79174 **Writing Data from an Array**

79175 The following example writes data from the buffers specified by members of the *iov* array to the  
 79176 file associated with the file descriptor *fd*.

```
79177 #include <sys/types.h>
79178 #include <sys/uio.h>
79179 #include <unistd.h>
79180 ...
79181 ssize_t bytes_written;
79182 int fd;
79183 char *buf0 = "short string\n";
79184 char *buf1 = "This is a longer string\n";
79185 char *buf2 = "This is the longest string in this example\n";
79186 int iocnt;
79187 struct iovec iov[3];
```

```
79188     iov[0].iov_base = buf0;
79189     iov[0].iov_len = strlen(buf0);
79190     iov[1].iov_base = buf1;
79191     iov[1].iov_len = strlen(buf1);
79192     iov[2].iov_base = buf2;
79193     iov[2].iov_len = strlen(buf2);
79194     ...
79195     iovcnt = sizeof(iov) / sizeof(struct iovec);
79196     bytes_written = writev(fd, iov, iovcnt);
79197     ...
```

**79198 APPLICATION USAGE**

79199 None.

**79200 RATIONALE**

79201 Refer to *write()*.

**79202 FUTURE DIRECTIONS**

79203 None.

**79204 SEE ALSO**

79205 *readv()*, *write()*

79206 XBD [<limits.h>](#), [<sys/uio.h>](#)

**79207 CHANGE HISTORY**

79208 First released in Issue 4, Version 2.

**79209 Issue 6**

79210 Split out from the *write()* reference page.

79211 **NAME**

79212           wscanf — convert formatted wide-character input

79213 **SYNOPSIS**

79214           #include &lt;stdio.h&gt;

79215           #include &lt;wchar.h&gt;

79216           int wscanf(const wchar\_t \*restrict *format*, ...);79217 **DESCRIPTION**79218           Refer to *fwscanf()*.

79219 **NAME**79220 `y0, y1, yn` — Bessel functions of the second kind79221 **SYNOPSIS**

```
79222 XSI      #include <math.h>
79223         double y0(double x);
79224         double y1(double x);
79225         double yn(int n, double x);
```

79226 **DESCRIPTION**

79227 The `y0()`, `y1()`, and `yn()` functions shall compute Bessel functions of  $x$  of the second kind of  
 79228 orders 0, 1, and  $n$ , respectively.  $y0(x)$  shall be equivalent to  $yn(0, x)$ , and  $y1(x)$  shall be equivalent  
 79229 to  $yn(1, x)$ .

79230 An application wishing to check for error situations should set `errno` to zero and call  
 79231 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or  
 79232 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-  
 79233 zero, an error has occurred.

79234 **RETURN VALUE**

79235 Upon successful completion, these functions shall return the relevant Bessel value of  $x$  of the  
 79236 second kind.

79237 MXX **If  $x$  is NaN, NaN shall be returned.**

79238 MXX If the  $x$  argument to these functions is negative, **either NaN (if supported) or** the same return  
 79239 value as when  $x$  is 0.0 (see below) shall be returned, and a domain error may occur.

79240 If  $x$  is 0.0, `y0()` and `y1()` shall return `-HUGE_VAL` and a pole error may occur. If  $x$  is 0.0 and  $n$  is  
 79241 not both negative and odd, `yn()` shall return `-HUGE_VAL` and a pole error may occur. If  $x$  is 0.0  
 79242 and  $n$  is negative and odd, `yn()` shall return `+HUGE_VAL` and a pole error may occur.

79243 MXX **If  $x$  is +Inf, +0 shall be returned.**

79244 MXX If the correct result would cause underflow **and is not representable,** a range error may occur,  
 79245 MXX and the function shall return `0.0`, **or** (if the IEC 60559 Floating-Point option is not supported) an  
 79246 implementation-defined value no greater in magnitude than `DBL_MIN`.

79247 MXX **If the correct result would cause underflow, and is representable, a range error may occur and**  
 79248 **the correct value shall be returned.**

79249 If the correct result of calling `y1()` would cause overflow, `-HUGE_VAL` shall be returned and a  
 79250 range error may occur. If  $n$  is not both negative and odd, and the correct result of calling `yn()`  
 79251 would cause overflow, `-HUGE_VAL` shall be returned and a range error may occur. If  $n$  is  
 79252 negative and odd, and the correct result of calling `yn()` would cause overflow, `+HUGE_VAL`  
 79253 shall be returned and a range error may occur.

79254 **ERRORS**

79255 These functions may fail if:

79256 Domain Error The value of  $x$  is negative.

79257 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,  
 79258 then `errno` shall be set to `[EDOM]`. If the integer expression (`math_errhandling`  
 79259 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception  
 79260 shall be raised.

79261	Pole Error	The value of $x$ is zero.
79262		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
79263		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
79264		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
79265		floating-point exception shall be raised.
79266	Range Error	The value of $x$ is too large in magnitude, or the correct result would cause
79267		underflow.
79268		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
79269		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
79270		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
79271		floating-point exception shall be raised.
79272		The <i>y1()</i> and <i>yn()</i> functions may fail if:
79273	Range Error	The correct result would cause overflow.
79274		If the integer expression ( <i>math_errhandling</i> & MATH_ERRNO) is non-zero,
79275		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
79276		( <i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
79277		floating-point exception shall be raised.
79278	<b>EXAMPLES</b>	
79279		None.
79280	<b>APPLICATION USAGE</b>	
79281		On error, the expressions ( <i>math_errhandling</i> & MATH_ERRNO) and ( <i>math_errhandling</i> &
79282		MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.
79283	<b>RATIONALE</b>	
79284		None.
79285	<b>FUTURE DIRECTIONS</b>	
79286		None.
79287	<b>SEE ALSO</b>	
79288		<i>feclearexcept()</i> , <i>fetestexcept()</i> , <i>isnan()</i> , <i>j0()</i>
79289		XBD Section 4.23 (on page 109), < <b>math.h</b> >
79290	<b>CHANGE HISTORY</b>	
79291		First released in Issue 1. Derived from Issue 1 of the SVID.
79292	<b>Issue 5</b>	
79293		The DESCRIPTION is updated to indicate how an application should check for an error. This
79294		text was previously published in the APPLICATION USAGE section.
79295	<b>Issue 6</b>	
79296		The normative text is updated to avoid use of the term “must” for application requirements.
79297		The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
79298		with the ISO/IEC 9899:1999 standard.
79299		IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/148 is applied, updating the RETURN
79300		VALUE and ERRORS sections. The changes are made for consistency with the general rules
79301		stated in “Treatment of Error Conditions for Mathematical Functions” in the Base Definitions
79302		volume of POSIX.1-2024.



79303 **Issue 7**

79304 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0746 [68] is applied.

79305 **Issue 8**

79306 Austin Group Defect 714 is applied, changing the behavior of these functions for special cases to  
79307 be a better match for their mathematical behavior.



79308

 *The Open Group Standard*

79309

**Vol. 3:**

79310

**Shell and Utilities, Issue 8**

79311

*The Open Group*

79312

*The Institute of Electrical and Electronics Engineers, Inc.*



79313

79314

# Introduction

79315

The Shell and Utilities volume of POSIX.1-2024 describes the commands and utilities offered to application programs by POSIX-conformant systems.

79316

79317

## 1.1 Relationship to Other Documents

79318

### 1.1.1 System Interfaces

79319

This subsection describes some of the features provided by the System Interfaces volume of POSIX.1-2024 that are assumed to be globally available on all systems conforming to this volume of POSIX.1-2024. This subsection does not attempt to detail all of the features defined in the System Interfaces volume of POSIX.1-2024 that are required by all of the utilities defined in this volume of POSIX.1-2024; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

79320

79321

79322

79323

79324

79325

The following subsections describe frequently used concepts. Many of these concepts are described in the Base Definitions volume of POSIX.1-2024. Utility and function description statements override these defaults when appropriate.

79326

79327

79328

#### 1.1.1.1 Process Attributes

79329

The following process attributes, as described in the System Interfaces volume of POSIX.1-2024, are assumed to be supported for all processes in this volume of POSIX.1-2024:

79330

79331

Controlling Terminal            Real Group ID

79332

Current Working Directory    Real User ID

79333

Effective Group ID            Root Directory

79334

Effective User ID            Saved Set-Group-ID

79335

File Descriptors            Saved Set-User-ID

79336

File Mode Creation Mask    Session Membership

79337

Process Group ID            Supplementary Group IDs

79338

Process ID

79339

A conforming implementation may include additional process attributes.

79340

#### 1.1.1.2 Concurrent Execution of Processes

79341

The following functionality of the *fork()* function defined in the System Interfaces volume of POSIX.1-2024 shall be available on all systems conforming to this volume of POSIX.1-2024:

79342

79343

1. Independent processes shall be capable of executing independently without either process terminating.

79344

79345 2. A process shall be able to create a new process with all of the attributes referenced in  
 79346 [Section 1.1.1.1](#) (on page 2453), determined according to the semantics of a call to the *fork()*  
 79347 function defined in the System Interfaces volume of POSIX.1-2024 followed by a call in  
 79348 the child process to one of the *exec* functions defined in the System Interfaces volume of  
 79349 POSIX.1-2024.

79350 1.1.1.3 *File Access Permissions*

79351 The file access control mechanism described by XBD [Section 4.7](#) (on page 97) shall apply to all  
 79352 files on an implementation conforming to this volume of POSIX.1-2024.

79353 1.1.1.4 *File Read, Write, and Creation*

79354 If a file that does not exist is to be written, it shall be created as described below, unless the  
 79355 utility description states otherwise.

79356 When a file that does not exist is created, the following features defined in the System Interfaces  
 79357 volume of POSIX.1-2024 shall apply unless the utility or function description states otherwise:

- 79358 1. The user ID of the file shall be set to the effective user ID of the calling process.
- 79359 2. The group ID of the file shall be set to the effective group ID of the calling process or the  
 79360 group ID of the directory in which the file is being created.
- 79361 3. If the file is a regular file, the permission bits of the file shall be set to:

79362 S\_IROTH | S\_IWOTH | S\_IRGRP | S\_IWGRP | S\_IRUSR | S\_IWUSR

79363 (see the description of *File Modes* in XBD [Chapter 14](#) (on page 221), `<sys/stat.h>`) except  
 79364 that the bits specified by the file mode creation mask of the process shall be cleared. If the  
 79365 file is a directory, the permission bits shall be set to:

79366 S\_IRWXU | S\_IRWXG | S\_IRWXO

79367 except that the bits specified by the file mode creation mask of the process shall be  
 79368 cleared.

- 79369 4. The last data access, last data modification, and last file status change timestamps of the  
 79370 file shall be updated as specified in XBD [Section 4.12](#) (on page 98).
- 79371 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length  
 79372 zero.
- 79373 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2\_SYMLINKS}  
 79374 variable is in effect for the directory in which the symbolic link would be created.
- 79375 7. Unless otherwise specified, the file created shall be a regular file.

79376 When an attempt is made to create a file that already exists, the utility shall take the action  
 79377 indicated in [Table 1-1](#) (on page 2455) corresponding to the type of the file the utility is trying to  
 79378 create and the type of the existing file, unless the utility description states otherwise.

79379

**Table 1-1** Actions when Creating a File that Already Exists

79380

79381

79382

79383

79384

79385

79386

79387

79388

79389

79390

79391

79392

Existing Type	New Type											Function Creating New
	B	C	D	F	L	M	P	Q	R	S	T	
B Block Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
C Character Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
D Directory	F	F	F	F	F	—	—	—	F	—	U	<i>mkdir()</i>
F FIFO Special File	F	F	F	F	F	—	—	—	O	—	U	<i>mkfifo()</i>
L Symbolic Link	F	F	F	F	F	—	—	—	FL	—	U	<i>symlink()</i>
M Shared Memory	F	F	F	F	F	—	—	—	—	—	U	<i>shm_open()</i>
P Semaphore	F	F	F	F	F	—	—	—	—	—	U	<i>sem_open()</i>
Q Message Queue	F	F	F	F	F	—	—	—	—	—	U	<i>mq_open()</i>
R Regular File	F	F	F	F	F	—	—	—	RF	—	U	<i>open()</i>
S Socket	F	F	F	F	F	—	—	—	—	—	U	<i>bind()</i>
T Typed Memory	F	F	F	F	F	U	U	U	U	U	U	*

79393

The following codes are used in [Table 1-1](#):

79394

79395

79396

**F** Fail. The attempt to create the new file shall fail and the utility shall either continue with its operation or exit immediately with an exit status that indicates an error occurred, depending on the description of the utility.

79397

79398

79399

79400

**FL** Follow link. Unless otherwise specified, the symbolic link shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

79401

79402

**O** Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

79403

79404

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.

79405

79406

2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

79407

**OF** The named file shall be opened with the consequences defined for that file type.

79408

79409

79410

**RF** Regular file. When attempting to create a regular file, and the existing file is a regular file:

1. The user ID, group ID, and permission bits of the file shall not be changed.

2. The file shall be truncated to zero length.

3. The last data modification and last file status change timestamps shall be marked for update.

79413

— The effect is implementation-defined unless specified by the utility description.

79414

**U** The effect is unspecified unless specified by the utility description.

79415

\* There is no portable way to create a file of this type.

79416

\*\* Not portable.

79417

79418

79419

When a file is to be appended, the file shall be opened in a manner equivalent to using the `O_APPEND` flag, without the `O_TRUNC` flag, in the `open()` function defined in the System Interfaces volume of POSIX.1-2024.

79420

When a file is to be read or written, the file shall be opened with an access mode corresponding

79421 to the operation to be performed. If file access permissions deny access, the requested operation  
79422 shall fail.

#### 79423 1.1.1.5 File Removal

79424 When a directory that is the root directory or current working directory of any process is  
79425 removed, the effect is implementation-defined. If file access permissions deny access, the  
79426 requested operation shall fail. Otherwise, when a file is removed:

- 79427 1. Its directory entry shall be removed from the file system.
- 79428 2. The link count of the file shall be decremented.
- 79429 3. If the file is an empty directory (see XBD [Section 3.119](#), on page 48):
  - 79430 a. If no process has the directory open, the space occupied by the directory shall be  
79431 freed and the directory shall no longer be accessible.
  - 79432 b. If one or more processes have the directory open, the directory contents shall be  
79433 preserved until all references to the file have been closed.
- 79434 4. If the file is a directory that is not empty, the last file status change timestamp shall be  
79435 marked for update.
- 79436 5. If the file is not a directory:
  - 79437 a. If the link count becomes zero:
    - 79438 i. If no process has the file open, the space occupied by the file shall be freed  
79439 and the file shall no longer be accessible.
    - 79440 ii. If one or more processes have the file open, the file contents shall be  
79441 preserved until all references to the file have been closed.
  - 79442 b. If the link count is not reduced to zero, the last file status change timestamp shall  
79443 be marked for update.
- 79444 6. The last data modification and last file status change timestamps of the containing  
79445 directory shall be marked for update.

#### 79446 1.1.1.6 File Time Values

79447 All files shall have the three time values described by XBD [Section 4.12](#) (on page 98).

#### 79448 1.1.1.7 File Contents

79449 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of  
79450 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the  
79451 following operations defined in the System Interfaces volume of POSIX.1-2024:

```
79452 while (read (fildes, buf, nbytes) > 0)
79453     ;
```

79454 If the file is indicated by a pathname *pathname*, the file descriptor shall be determined by the  
79455 equivalent of the following operation defined in the System Interfaces volume of POSIX.1-2024:

```
79456 fildes = open (pathname, O_RDONLY);
```

79457 The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data  
79458 returned by *read()* would vary with different values, the value shall be one that results in the



79459 most data being returned.

79460 If the `read()` function calls would return an error, it is unspecified whether the contents of the file  
79461 are considered to include any data from offsets in the file beyond where the error would be  
79462 returned.

#### 79463 1.1.1.8 Pathname Resolution

79464 The pathname resolution algorithm, described by XBD Section 4.16 (on page 105), shall be used  
79465 by implementations conforming to this volume of POSIX.1-2024; see also XBD Section 4.8 (on  
79466 page 97).

#### 79467 1.1.1.9 Changing the Current Working Directory

79468 When the current working directory (see XBD Section 3.94, on page 45) is to be changed, unless  
79469 the utility or function description states otherwise, the operation shall succeed unless a call to  
79470 the `chdir()` function defined in the System Interfaces volume of POSIX.1-2024 would fail when  
79471 invoked with the new working directory pathname as its argument.

#### 79472 1.1.1.10 Establish the Locale

79473 The functionality of the `setlocale()` function defined in the System Interfaces volume of  
79474 POSIX.1-2024 shall be available on all systems conforming to this volume of POSIX.1-2024; that  
79475 is, utilities that require the capability of establishing an international operating environment  
79476 shall be permitted to set the specified category of the international environment.

#### 79477 1.1.1.11 Actions Equivalent to Functions

79478 Some utility descriptions specify that a utility performs actions equivalent to a function defined  
79479 in the System Interfaces volume of POSIX.1-2024. Such specifications require only that the  
79480 external effects be equivalent, not that any effect within the utility and visible only to the utility  
79481 be equivalent.

### 79482 1.1.2 Concepts Derived from the ISO C Standard

79483 Some of the standard utilities perform complex data manipulation using their own procedure  
79484 and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS  
79485 sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type  
79486 conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard,  
79487 as described in the following sections. Note that there is no requirement that the standard  
79488 utilities be implemented in any particular programming language.

#### 79489 1.1.2.1 Arithmetic Precision and Operations

79490 Integer variables and constants, including the values of operands and option-arguments, used  
79491 by the standard utilities listed in this volume of POSIX.1-2024 shall be implemented as  
79492 equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as  
79493 equivalent to the ISO C standard **double** type. Conversions between types shall be as described  
79494 in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned  
79495 by the input to the application.

79496 Arithmetic operators and control flow keywords shall be implemented as equivalent to those in

79497 the cited ISO C standard section, as listed in [Table 1-2](#).

79498 **Note:** The comma operator (section 6.5.17 of the ISO C standard) is intentionally not included in the  
79499 table. It need not be supported by implementations.

79500 **Table 1-2** Selected ISO C Standard Operators and Control Flow Keywords

79501	Operation	ISO C Standard Equivalent Reference
79502	()	Section 6.5.1, Primary Expressions
79503	postfix ++	Section 6.5.2, Postfix Operators
79504	postfix --	
79505	unary +	Section 6.5.3, Unary Operators
79506	unary -	
79507	prefix ++	
79508	prefix --	
79509	~	
79510	!	
79511	sizeof()	
79512	*	Section 6.5.5, Multiplicative Operators
79513	/	
79514	%	
79515	+	Section 6.5.6, Additive Operators
79516	-	
79517	<<	Section 6.5.7, Bitwise Shift Operators
79518	>>	
79519	<, <=	Section 6.5.8, Relational Operators
79520	>, >=	
79521	==	Section 6.5.9, Equality Operators
79522	!=	
79523	&	Section 6.5.10, Bitwise AND Operator
79524	^	Section 6.5.11, Bitwise Exclusive OR Operator
79525		Section 6.5.12, Bitwise Inclusive OR Operator
79526	&&	Section 6.5.13, Logical AND Operator
79527		Section 6.5.14, Logical OR Operator
79528	expr?expr:expr	Section 6.5.15, Conditional Operator
79529	=, *=, /=, %=, +=, -=	Section 6.5.16, Assignment Operators
79530	<<=, >>=, &=, ^=,  =	
79531	if ()	Section 6.8.4, Selection Statements
79532	if () ... else	
79533	switch ()	
79534	while ()	Section 6.8.5, Iteration Statements
79535	do ... while ()	
79536	for ()	
79537	goto	Section 6.8.6, Jump Statements
79538	continue	
79539	break	
79540	return	

79541 The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5,  
79542 Expressions, of the ISO C standard.

### 79543 1.1.2.2 Mathematical Functions

79544 Any mathematical functions with the same names as those in the following sections of the ISO C  
79545 standard:

- 79546 • Section 7.12, Mathematics, `<math.h>`
- 79547 • Section 7.22.2, Pseudo-Random Sequence Generation Functions

79548 shall be implemented to return the results equivalent to those returned from a call to the  
79549 corresponding function described in the ISO C standard.

## 79550 1.2 Utility Limits

79551 This section lists magnitude limitations imposed by a specific implementation. The braces  
79552 notation, {LIMIT}, is used in this volume of POSIX.1-2024 to indicate these values, but the braces  
79553 are not part of the name.

79554 **Table 1-3** Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	2 048
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	1 000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> <b>order</b> keyword in the locale definition file; see the <b>border_start</b> keyword in XBD <a href="#">Section 7.3.2</a> (on page 139).	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <code>&lt;newline&gt;</code> .	2 048
{POSIX_RE_DUP_MAX}	Maximum number of repeated occurrences of a BRE or ERE interval expression; see XBD <a href="#">Section 9.3.6</a> (on page 185) and <a href="#">Section 9.4.6</a> (on page 189).	255

79580 The values specified in [Table 1-3](#) represent the lowest values conforming implementations shall  
79581 provide and, consequently, the largest values on which an application can rely without further  
79582 enquiries, as described below. These values shall be accessible to applications via the *getconf*

79583 utility (see *getconf*, on page 2973).

79584 Implementations may provide more liberal, or less restrictive, values than shown in Table 1-3  
79585 (on page 2459). These possibly more liberal values are accessible using the symbols in Table 1-4.

79586 The *sysconf()* function defined in the System Interfaces volume of POSIX.1-2024 or the *getconf*  
79587 utility return the value of each symbol on each specific implementation. The value so retrieved is  
79588 the largest, or most liberal, value that is available throughout the session lifetime, as determined  
79589 at session creation. The literal names shown in the table apply only to the *getconf* utility; the  
79590 high-level language binding describes the exact form of each name to be used by the interfaces  
79591 in that binding.

79592 All numeric limits defined by the System Interfaces volume of POSIX.1-2024, such as  
79593 {PATH\_MAX}, shall also apply to this volume of POSIX.1-2024. All the utilities defined by this  
79594 volume of POSIX.1-2024 are implicitly limited by these values, unless otherwise noted in the  
79595 utility descriptions.

79596 It is not guaranteed that the application can actually reach the specified limit of an  
79597 implementation in any given case, or at all, as a lack of virtual memory or other resources may  
79598 prevent this. The limit value indicates only that the implementation does not specifically impose  
79599 any arbitrary, more restrictive limit.

79600 **Table 1-4** Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE order</i> keyword in the locale definition file; see the <i>order_start</i> keyword in XBD Section 7.3.2 (on page 139).	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	{POSIX2_EXPR_NEST_MAX}

Name	Description	Minimum Value
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	{POSIX2_LINE_MAX}
{RE_DUP_MAX}	Maximum number of repeated occurrences of a BRE or ERE interval expression; see XBD <a href="#">Section 9.3.6</a> (on page 185) and <a href="#">Section 9.4.6</a> (on page 189).	{POSIX_RE_DUP_MAX}

The following value may be a constant within an implementation or may vary from one pathname to another.

{POSIX2\_SYMLINKS}

When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2\_SYMLINKS} is undefined.

### 1.3 Grammar Conventions

Portions of this volume of POSIX.1-2024 are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input, suitable for program use; the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial *yacc* code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores; for example, **NEWLINE**, **ASSIGN\_OP**, **NAME**.
- The identifiers for non-terminals are all lowercase.

## 79663 1.4 Utility Description Defaults

79664 This section describes all of the subsections used within the utility descriptions, including:

- 79665 • Intended usage of the section
- 79666 • Global defaults that affect all the standard utilities
- 79667 • The meanings of notations used in this volume of POSIX.1-2024 that are specific to
- 79668 individual utility sections

### 79669 NAME

79670 This section gives the name or names of the utility and briefly states its purpose.

### 79671 SYNOPSIS

79672 The SYNOPSIS section summarizes the syntax of the calling sequence for the utility,  
79673 including options, option-arguments, and operands. Standards for utility naming are  
79674 described in XBD [Section 12.2](#) (on page 215); for describing the utility's arguments in  
79675 XBD [Section 12.1](#) (on page 213).

### 79676 DESCRIPTION

79677 The DESCRIPTION section describes the actions of the utility. If the utility has a very  
79678 complex set of subcommands or its own procedural language, an EXTENDED  
79679 DESCRIPTION section is also provided. Most explanations of optional functionality are  
79680 omitted here, as they are usually explained in the OPTIONS section.

79681 As stated in [Section 1.1.1.11](#) (on page 2457), some functions are described in terms of  
79682 equivalent functionality. When specific functions are cited, the implementation shall  
79683 provide equivalent functionality including side-effects associated with successful  
79684 execution of the function. The treatment of errors and intermediate results from the  
79685 individual functions cited is generally not specified by this volume of POSIX.1-2024.  
79686 See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all  
79687 actions associated with errors encountered by the utility.

79688 A standard utility shall not be treated as a declaration utility unless explicitly stated in  
79689 this section.

### 79690 OPTIONS

79691 The OPTIONS section describes the utility options and option-arguments, and how  
79692 they modify the actions of the utility. Standard utilities that have options either fully  
79693 comply with XBD [Section 12.2](#) (on page 215) or describe all deviations. Apparent  
79694 disagreements between functionality descriptions in the OPTIONS and DESCRIPTION  
79695 (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS  
79696 section.

79697 Each OPTIONS section that uses the phrase “The ... utility shall conform to the Utility  
79698 Syntax Guidelines ...” refers only to the use of the utility as specified by this volume of  
79699 POSIX.1-2024; implementation extensions should also conform to the guidelines, but  
79700 may allow exceptions for historical practice.

79701 Unless otherwise stated in the utility description, when given an option unrecognized  
79702 by the implementation, or when a required option-argument is not provided, standard  
79703 utilities shall issue a diagnostic message to standard error and exit with an exit status  
79704 that indicates an error occurred.

79705 All utilities in this volume of POSIX.1-2024 shall be capable of processing arguments  
79706 using eight-bit transparency.

79707 **Default Behavior:** When this section is listed as “None.”, it means that the  
79708 implementation need not support any options. Standard utilities that do not accept

79709 options, but that do accept operands, shall recognize "--" as a first argument to be  
79710 discarded.

79711 The requirement for recognizing "--" is because conforming applications need a way  
79712 to shield their operands from any arbitrary options that the implementation may  
79713 provide as an extension. For example, if the standard utility *foo* is listed as taking no  
79714 options, and the application needed to give it a pathname with a leading <hyphen-  
79715 minus>, it could safely do it as:

79716 `foo -- -myfile`

79717 and avoid any problems with `-m` used as an extension.

## 79718 OPERANDS

79719 The OPERANDS section describes the utility operands, and how they affect the actions  
79720 of the utility. Apparent disagreements between functionality descriptions in the  
79721 OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be  
79722 resolved in favor of the OPERANDS section.

79723 If an operand naming a file can be specified as '-', which means to use the standard  
79724 input instead of a named file, this is explicitly stated in this section. Unless otherwise  
79725 stated, the use of multiple instances of '-' to mean standard input in a single  
79726 command produces unspecified results.

79727 Unless otherwise stated, the standard utilities that accept operands shall process those  
79728 operands in the order specified in the command line.

79729 **Default Behavior:** When this section is listed as "None.", it means that the  
79730 implementation need not support any operands.

## 79731 STDIN

79732 The STDIN section describes the standard input of the utility. This section is frequently  
79733 merely a reference to the following section, as many utilities treat standard input and  
79734 input files in the same manner. Unless otherwise stated, all restrictions described in the  
79735 INPUT FILES section shall apply to this section as well.

79736 Use of a terminal for standard input can cause any of the standard utilities that read  
79737 standard input to stop when used in the background. For this reason, applications  
79738 should not use interactive features in scripts to be placed in the background.

79739 The specified standard input format of the standard utilities shall not depend on the  
79740 existence or value of the environment variables defined in this volume of POSIX.1-2024,  
79741 except as provided by this volume of POSIX.1-2024.

79742 **Default Behavior:** When this section is listed as "Not used.", it means that the standard  
79743 input shall not be read when the utility is used as described by this volume of  
79744 POSIX.1-2024.

## 79745 INPUT FILES

79746 The INPUT FILES section describes the files, other than the standard input, used as  
79747 input by the utility. It includes files named as operands and option-arguments as well  
79748 as other files that are referred to, such as start-up and initialization files, databases, and  
79749 so on. Commonly-used files are generally described in one place and cross-referenced  
79750 by other utilities.

79751 All utilities in this volume of POSIX.1-2024 shall be capable of processing input files  
79752 using eight-bit transparency.

79753 When a standard utility reads a seekable input file and terminates without an error  
79754 before it reaches end-of-file, the utility shall ensure that the file offset in the open file

79755 description is properly positioned just past the last byte processed by the utility. For  
 79756 files that are not seekable, the state of the file offset in the open file description for that  
 79757 file is unspecified. A conforming application shall not assume that the following three  
 79758 commands are equivalent:

```
79759 tail -n +2 file
79760 (sed -n 1q; cat) < file
79761 cat file | (sed -n 1q; cat)
```

79762 The second command is equivalent to the first only when the file is seekable. The third  
 79763 command leaves the file offset in the open file description in an unspecified state. Other  
 79764 utilities, such as *head*, *read*, and *sh*, have similar properties.

79765 Some of the standard utilities, such as filters, process input files a line or a block at a  
 79766 time and have no restrictions on the maximum input file size. Some utilities may have  
 79767 size limitations that are not as obvious as file space or memory limitations. Such  
 79768 limitations should reflect resource limitations of some sort, not arbitrary limits set by  
 79769 implementors. Implementations shall document those utilities that are limited by  
 79770 constraints other than file system space, available memory, and other limits specifically  
 79771 cited by this volume of POSIX.1-2024, and identify what the constraint is and indicate a  
 79772 way of estimating when the constraint would be reached. Similarly, some utilities  
 79773 descend the directory tree (recursively). Implementations shall also document any  
 79774 limits that they may have in descending the directory tree that are beyond limits cited  
 79775 by this volume of POSIX.1-2024.

79776 When an input file is described as a “text file”, the utility produces undefined results if  
 79777 given input that is not from a text file, unless otherwise stated. Some utilities (for  
 79778 example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline>  
 79779 convention; unless otherwise stated, the utility need not be able to accumulate more  
 79780 than {LINE\_MAX} bytes from a set of multiple, continued input lines. Thus, for a  
 79781 conforming application the total of all the continued lines in a set cannot exceed  
 79782 {LINE\_MAX}. If a utility using the escaped <newline> convention detects an end-of-  
 79783 file condition immediately after an escaped <newline>, the results are unspecified.

79784 Record formats are described in a notation similar to that used by the C-language  
 79785 function, *printf()*. See XBD Chapter 5 (on page 113) for a description of this notation.  
 79786 The format description is intended to be sufficiently rigorous to allow other  
 79787 applications to generate these input files. However, since <blank>s can legitimately be  
 79788 included in some of the fields described by the standard utilities, particularly in locales  
 79789 other than the POSIX locale, this intent is not always realized.

79790 **Default Behavior:** When this section is listed as “None.”, it means that no input files  
 79791 are required to be supplied when the utility is used as described by this volume of  
 79792 POSIX.1-2024.

## 79793 ENVIRONMENT VARIABLES

79794 The ENVIRONMENT VARIABLES section lists what variables affect the utility’s  
 79795 execution.

79796 The entire manner in which environment variables described in this volume of  
 79797 POSIX.1-2024 affect the behavior of each utility is described in the ENVIRONMENT  
 79798 VARIABLES section for that utility, in conjunction with the global effects of the *LANG*,  
 79799 XSI *LC\_ALL*, and *NLSPATH* environment variables described in XBD Chapter 8 (on page  
 79800 167). The existence or value of environment variables described in this volume of  
 79801 POSIX.1-2024 shall not otherwise affect the specified behavior of the standard utilities.  
 79802 Any effects of the existence or value of environment variables not described by this  
 79803 volume of POSIX.1-2024 upon the standard utilities are unspecified.



79804 For those standard utilities that use environment variables as a means for selecting a  
 79805 utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to  
 79806 the path search described for *PATH* in XBD [Chapter 8](#) (on page 167).

79807 All utilities in this volume of POSIX.1-2024 shall be capable of processing environment  
 79808 variable names and values using eight-bit transparency.

79809 **Default Behavior:** When this section is listed as “None.”, it means that the behavior of  
 79810 the utility is not directly affected by environment variables described by this volume of  
 79811 POSIX.1-2024 when the utility is used as described by this volume of POSIX.1-2024.

## 79812 ASYNCHRONOUS EVENTS

79813 The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as  
 79814 signals and what signals are caught.

79815 **Default Behavior:** When this section is listed as “Default.”, or it refers to “the standard  
 79816 action” for any signal, it means that the action taken as a result of the signal shall be as  
 79817 follows:

- 79818 • If the action inherited from the invoking process, according to the rules of  
 79819 inheritance of signal actions defined in the System Interfaces volume of  
 79820 POSIX.1-2024, is for the signal to be ignored, the utility shall ignore the signal.
- 79821 • If the action inherited from the invoking process, according to the rules of  
 79822 inheritance of signal actions defined in System Interfaces volume of  
 79823 POSIX.1-2024, is the default signal action, the result of the utility’s execution shall  
 79824 be as if the default signal action had been taken.

79825 When the required action is for the signal to terminate the utility, the utility may catch  
 79826 the signal, perform some additional processing (such as deleting temporary files),  
 79827 restore the default signal action, and resignal itself.

## 79828 STDOUT

79829 The STDOUT section completely describes the standard output of the utility. This  
 79830 section is frequently merely a reference to the following section, OUTPUT FILES,  
 79831 because many utilities treat standard output and output files in the same manner.

79832 Use of a terminal for standard output may cause any of the standard utilities that write  
 79833 standard output to stop when used in the background. For this reason, applications  
 79834 should not use interactive features in scripts to be placed in the background.

79835 Record formats are described in a notation similar to that used by the C-language  
 79836 function, *printf()*. See XBD [Chapter 5](#) (on page 113) for a description of this notation.

79837 The specified standard output of the standard utilities shall not depend on the  
 79838 existence or value of the environment variables defined in this volume of POSIX.1-2024,  
 79839 except as provided by this volume of POSIX.1-2024.

79840 Some of the standard utilities describe their output using the verb *display*, defined in  
 79841 XBD [Section 3.107](#) (on page 46). Output described in the STDOUT sections of such  
 79842 utilities may be produced using means other than standard output. When standard  
 79843 output is directed to a terminal, the output described shall be written directly to the  
 79844 terminal. Otherwise, the results are undefined.

79845 **Default Behavior:** When this section is listed as “Not used.”, it means that the standard  
 79846 output shall not be written when the utility is used as described by this volume of  
 79847 POSIX.1-2024.

79848 **STDERR**

79849 The STDERR section describes the standard error output of the utility. Only those  
79850 messages that are purposely sent by the utility are described.

79851 Use of a terminal for standard error may cause any of the standard utilities that write  
79852 standard error output to stop when used in the background. For this reason,  
79853 applications should not use interactive features in scripts to be placed in the  
79854 background.

79855 The format of diagnostic messages for most utilities is unspecified, but the language  
79856 and cultural conventions of diagnostic and informative messages whose format is  
79857 unspecified by this volume of POSIX.1-2024 should be affected by the setting of  
79858 XSI *LC\_MESSAGES* and *NLSPATH*.

79859 The specified standard error output of standard utilities shall not depend on the  
79860 existence or value of the environment variables defined in this volume of POSIX.1-2024,  
79861 except as provided by this volume of POSIX.1-2024.

79862 **Default Behavior:** When this section is listed as “The standard error shall be used only  
79863 for diagnostic messages.”, it means that, unless otherwise stated, the diagnostic  
79864 messages shall be sent to the standard error only when the exit status indicates that an  
79865 error occurred and the utility is used as described by this volume of POSIX.1-2024.

79866 When this section is listed as “Not used.”, it means that the standard error shall not be  
79867 used when the utility is used as described in this volume of POSIX.1-2024.

79868 **OUTPUT FILES**

79869 The OUTPUT FILES section completely describes the files created or modified by the  
79870 utility. Temporary or system files that are created for internal usage by this utility or  
79871 other parts of the implementation (for example, spool, log, and audit files) are not  
79872 described in this, or any, section. The utilities creating such files and the names of such  
79873 files are unspecified. If applications are written to use temporary or intermediate files,  
79874 they should use the *TMPDIR* environment variable, if it is set and represents an  
79875 accessible directory, to select the location of temporary files.

79876 Implementations shall ensure that temporary files, when used by the standard utilities,  
79877 are named so that different utilities or multiple instances of the same utility can operate  
79878 simultaneously without regard to their working directories, or any other process  
79879 characteristic other than process ID. There are two exceptions to this rule:

- 79880 1. Resources for temporary files other than the name space (for example, disk  
79881 space, available directory entries, or number of processes allowed) are not  
79882 guaranteed.
- 79883 2. Certain standard utilities generate output files that are intended as input for  
79884 other utilities (for example, *lex* generates *lex.yy.c*), and these cannot have unique  
79885 names. These cases are explicitly identified in the descriptions of the respective  
79886 utilities.

79887 Any temporary file created by the implementation shall be removed by the  
79888 implementation upon a utility’s successful exit, exit because of errors, or before  
79889 termination by any of the SIGHUP, SIGINT, or SIGTERM signals, unless specified  
79890 otherwise by the utility description.

79891 Receipt of the SIGQUIT signal should generally cause termination (unless in some  
79892 debugging mode) that would bypass any attempted recovery actions.

79893 Record formats are described in a notation similar to that used by the C-language  
79894 function, *printf()*; see XBD Chapter 5 (on page 113) for a description of this notation.

79895 **Default Behavior:** When this section is listed as “None.”, it means that no files are  
 79896 created or modified as a consequence of direct action on the part of the utility when the  
 79897 utility is used as described by this volume of POSIX.1-2024. However, the utility may  
 79898 create or modify system files, such as log files, that are outside the utility’s normal  
 79899 execution environment.

#### 79900 EXTENDED DESCRIPTION

79901 The EXTENDED DESCRIPTION section provides a place for describing the actions of  
 79902 very complicated utilities, such as text editors or language processors, which typically  
 79903 have elaborate command languages.

79904 **Default Behavior:** When this section is listed as “None.”, no further description is  
 79905 necessary.

#### 79906 EXIT STATUS

79907 The EXIT STATUS section describes the values the utility shall return to the calling  
 79908 program, or shell, and the conditions that cause these values to be returned. Usually,  
 79909 utilities return zero for successful completion and values greater than zero for various  
 79910 error conditions. If specific numeric values are listed in this section, the system shall  
 79911 use those values for the errors described. In some cases, status values are listed more  
 79912 loosely, such as >0. A strictly conforming application shall not rely on any specific  
 79913 value in the range shown and shall be prepared to receive any value in the range.

79914 For example, a utility may list zero as a successful return, 1 as a failure for a specific  
 79915 reason, and >1 as “an error occurred”. In this case, unspecified conditions may cause a  
 79916 2 or 3, or other value, to be returned. A conforming application should be written so  
 79917 that it tests for successful exit status values (zero in this case), rather than relying upon  
 79918 the single specific error value listed in this volume of POSIX.1-2024. In that way, it has  
 79919 maximum portability, even on implementations with extensions.

79920 Unspecified error conditions may be represented by specific values not listed in this  
 79921 volume of POSIX.1-2024.

79922 **Default Behavior:** When the description of exit status 0 is “Successful completion”, it  
 79923 means that exit status 0 shall indicate that all of the actions the utility is required to  
 79924 perform were completed successfully.

#### 79925 CONSEQUENCES OF ERRORS

79926 The CONSEQUENCES OF ERRORS section describes the effects on the environment,  
 79927 file systems, process state, and so on, when error conditions occur. It does not describe  
 79928 error messages produced or exit status values used.

79929 The many reasons for failure of a utility are generally not specified by the utility  
 79930 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of  
 79931 options, arguments, or environment variables; invalid usage of the complex syntaxes  
 79932 expressed in EXTENDED DESCRIPTION sections; resource exhaustion; difficulties  
 79933 accessing, creating, reading, or writing files; or difficulties associated with the  
 79934 privileges of the process.

79935 The following shall apply to each utility, unless otherwise stated:

- 79936 • If the requested action cannot be performed on an operand representing a file,  
 79937 directory, user, process, and so on, the utility shall issue a diagnostic message to  
 79938 standard error and continue processing the next operand in sequence, but the  
 79939 final exit status shall be one that indicates an error occurred.

79940 For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if  
 79941 the requested action cannot be performed on a file or directory encountered in the

79942 hierarchy, the utility shall issue a diagnostic message to standard error and  
 79943 continue processing the remaining files in the hierarchy, but the final exit status  
 79944 shall be one that indicates an error occurred.

79945 **Note:** If the requested action is to write one or more pathnames in a format that has  
 79946 <newline> as a terminator or separator, and a pathname to be written contains  
 79947 any bytes that have the encoded value of a <newline> character, this should be  
 79948 treated as an action that cannot be performed. A future version of this standard  
 79949 may require that utilities treat this as an error.

79950 • If the requested action characterized by an option or option-argument cannot be  
 79951 performed, the utility shall issue a diagnostic message to standard error and the  
 79952 exit status returned shall be one that indicates an error occurred.

79953 • When an unrecoverable error condition is encountered, the utility shall exit with  
 79954 an exit status that indicates an error occurred.

79955 • A diagnostic message shall be written to standard error whenever an error  
 79956 condition occurs.

79957 When a utility encounters an error condition several actions are possible, depending on  
 79958 the severity of the error and the state of the utility. Included in the possible actions of  
 79959 various utilities are: deletion of temporary or intermediate work files; deletion of  
 79960 incomplete files; validity checking of the file system or directory.

79961 **Default Behavior:** When this section is listed as “Default.”, it means that any changes  
 79962 to the environment, file systems, process state, and so on are unspecified.

#### 79963 APPLICATION USAGE

79964 This section is informative.

79965 The APPLICATION USAGE section gives advice to the application programmer or  
 79966 user about the way the utility should be used.

#### 79967 EXAMPLES

79968 This section is informative.

79969 The EXAMPLES section gives one or more examples of usage, where appropriate. In  
 79970 the event of conflict between an example and a normative part of the specification, the  
 79971 normative material is to be taken as correct.

79972 In all examples, quoting has been used, showing how sample commands (utility names  
 79973 combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to  
 79974 the *system()* function defined in the System Interfaces volume of POSIX.1-2024. Such  
 79975 quoting would not be used if the utility is invoked using one of the *exec* functions  
 79976 defined in the System Interfaces volume of POSIX.1-2024.

#### 79977 RATIONALE

79978 This section is informative.

79979 This section contains historical information concerning the contents of this volume of  
 79980 POSIX.1-2024 and why features were included or discarded by the standard  
 79981 developers.

#### 79982 FUTURE DIRECTIONS

79983 This section is informative.

79984 The FUTURE DIRECTIONS section should be used as a guide to current thinking; there  
 79985 is not necessarily a commitment to implement all of these future directions in their  
 79986 entirety.

79987           **SEE ALSO**  
 79988           This section is informative.  
 79989           The SEE ALSO section lists related entries.

79990           **CHANGE HISTORY**  
 79991           This section is informative.  
 79992           This section shows the derivation of the entry and any significant changes that have  
 79993           been made to it.

79994           Certain of the standard utilities describe how they can invoke other utilities or applications, such  
 79995           as by passing a command string to the command interpreter. The external influences (STDIN,  
 79996           ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF  
 79997           ERRORS, and so on) of such invoked utilities are not described in the section concerning the  
 79998           standard utility that invokes them.

## 79999 1.5 Considerations for Utilities in Support of Files of Arbitrary Size

80000           The following utilities support files of any size up to the maximum that can be created by the  
 80001           implementation. This support includes correct writing of file size-related values (such as file  
 80002           sizes and offsets, line numbers, and block counts) and correct interpretation of command line  
 80003           arguments that contain such values.

80004	<i>basename</i>	Return non-directory portion of pathname.
80005	<i>cat</i>	Concatenate and print files.
80006	<i>cd</i>	Change working directory.
80007	<i>chgrp</i>	Change file group ownership.
80008	<i>chmod</i>	Change file modes.
80009	<i>chown</i>	Change file ownership.
80010	<i>cksum</i>	Write file checksums and sizes.
80011	<i>cmp</i>	Compare two files.
80012	<i>cp</i>	Copy files.
80013	<i>dd</i>	Convert and copy a file.
80014	<i>df</i>	Report free disk space.
80015	<i>dirname</i>	Return directory portion of pathname.
80016	<i>du</i>	Estimate file space usage.
80017	<i>find</i>	Find files.
80018	<i>ln</i>	Link files.
80019	<i>ls</i>	List directory contents.
80020	<i>mkdir</i>	Make directories.
80021	<i>mv</i>	Move files.
80022	<i>pathchk</i>	Check pathnames.





# Shell Command Language

80061 This chapter contains the definition of the Shell Command Language.

## 80062 2.1 Shell Introduction

80063 The shell is a command language interpreter. This chapter describes the syntax of that command  
80064 language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the  
80065 System Interfaces volume of POSIX.1-2024.

80066 The shell operates according to the following general overview of operations. The specific  
80067 details are included in the cited sections of this chapter.

- 80068 1. The shell reads its input from a file (see *sh*), from the `-c` option or from the *system()* and  
80069 *popen()* functions defined in the System Interfaces volume of POSIX.1-2024. If the first  
80070 line of a file of shell commands starts with the characters "#!", the results are  
80071 unspecified.
- 80072 2. The shell breaks the input into tokens: words and operators; see [Section 2.3](#).
- 80073 3. The shell parses the input into simple commands (see [Section 2.9.1](#)) and compound  
80074 commands (see [Section 2.9.4](#)).
- 80075 4. For each word within a command, the shell processes <backslash>-escape sequences  
80076 inside dollar-single-quotes (see [Section 2.2.4](#)) and then performs various word expansions  
80077 (see [Section 2.6](#)). In the case of a simple command, the results usually include a list of  
80078 pathnames and fields to be treated as a command name and arguments; see [Section 2.9](#).
- 80079 5. The shell performs redirection (see [Section 2.7](#)) and removes redirection operators and  
80080 their operands from the parameter list.
- 80081 6. The shell executes a function (see [Section 2.9.5](#)), built-in (see [Section 2.15](#)), executable file,  
80082 or script, giving the names of the arguments as positional parameters numbered 1 to *n*,  
80083 and the name of the command (or in the case of a function within a script, the name of the  
80084 script) as special parameter 0 (see [Section 2.9.1.4](#)).
- 80085 7. The shell optionally waits for the command to complete and collects the exit status (see  
80086 [Section 2.8.2](#)).

## 80087 2.2 Quoting

80088 Quoting is used to remove the special meaning of certain characters or words to the shell.  
80089 Quoting can be used to preserve the literal meaning of the special characters in the next  
80090 paragraph, prevent reserved words from being recognized as such, and prevent parameter  
80091 expansion and command substitution within here-document processing (see [Section 2.7.4](#)).

80092 The application shall quote the following characters if they are to represent themselves:

80093 | & ; < > ( ) \$ ` \ " ' <space> <tab> <newline>

80094 and the following might need to be quoted under certain circumstances. That is, these



80095 characters are sometimes special depending on conditions described elsewhere in this volume of  
80096 POSIX.1-2024:

80097 \* ? [ ] ^ - ! # ~ = % { , }

80098 **Note:** A future version of this standard may extend the conditions under which these characters are  
80099 special. Therefore applications should quote them whenever they are intended to represent  
80100 themselves. This does not apply to <hyphen-minus> ('-') since it is in the portable filename  
80101 character set.

80102 The various quoting mechanisms are the escape character, single-quotes, double-quotes, and  
80103 dollar-single-quotes. The here-document represents another form of quoting; see [Section 2.7.4](#).

### 80104 2.2.1 Escape Character (Backslash)

80105 A <backslash> that is not quoted shall preserve the literal value of the following character, with  
80106 the exception of a <newline>. If a <newline> immediately follows the <backslash>, the shell  
80107 shall interpret this as line continuation. The <backslash> and <newline> shall be removed before  
80108 splitting the input into tokens. Since the escaped <newline> is removed entirely from the input  
80109 and is not replaced by any white space, it cannot serve as a token separator.

### 80110 2.2.2 Single-Quotes

80111 Enclosing characters in single-quotes (' ') shall preserve the literal value of each character  
80112 within the single-quotes. A single-quote cannot occur within single-quotes.

### 80113 2.2.3 Double-Quotes

80114 Enclosing characters in double-quotes (" ") shall preserve the literal value of all characters  
80115 within the double-quotes, with the exception of the characters backquote, <dollar-sign>, and  
80116 <backslash>, as follows:

80117 § The <dollar-sign> shall retain its special meaning introducing parameter expansion (see  
80118 [Section 2.6.2](#)), a form of command substitution (see [Section 2.6.3](#)), and arithmetic expansion  
80119 (see [Section 2.6.4](#)), but shall not retain its special meaning introducing the dollar-single-  
80120 quotes form of quoting (see [Section 2.2.4](#)).

80121 The input characters within the quoted string that are also enclosed between "\$ (" and the  
80122 matching ') ' shall not be affected by the double-quotes, but rather shall define the  
80123 command(s) whose output replaces the "\$ (...)" when the word is expanded. The  
80124 tokenizing rules in [Section 2.3](#) shall be applied recursively to find the matching ') '.

80125 For the four varieties of parameter expansion that provide for substring processing (see  
80126 [Section 2.6.2](#)), within the string of characters from an enclosed "\$ {" to the matching ' } ' ,  
80127 the double-quotes within which the expansion occurs shall have no effect on the handling  
80128 of any special characters.

80129 For parameter expansions other than the four varieties that provide for substring  
80130 processing, within the string of characters from an enclosed "\$ {" to the matching ' } ' , the  
80131 double-quotes within which the expansion occurs shall preserve the literal value of all  
80132 characters, with the exception of the characters double-quote, backquote, <dollar-sign>, and  
80133 <backslash>. If any unescaped double-quote characters occur within the string, other than  
80134 in embedded command substitutions, the behavior is unspecified. The backquote and  
80135 <dollar-sign> characters shall follow the same rules as for characters in double-quotes  
80136 described in this section. The <backslash> character shall follow the same rules as for

80137 characters in double-quotes described in this section except that it shall additionally retain  
 80138 its special meaning as an escape character when followed by '}' and this shall prevent the  
 80139 escaped '}' from being considered when determining the matching '}' (using the rule in  
 80140 [Section 2.6.2](#)).

80141 ‧ The backquote shall retain its special meaning introducing the other form of command  
 80142 substitution (see [Section 2.6.3](#)). The portion of the quoted string from the initial backquote  
 80143 and the characters up to the next backquote that is not preceded by a <backslash>, having  
 80144 escape characters removed, defines that command whose output replaces "`...`" when  
 80145 the word is expanded. Either of the following cases produces undefined results:

- 80146 • A quoted (single-quoted, double-quoted, or dollar-single-quoted) string that begins,  
 80147 but does not end, within the "`...`" sequence
- 80148 • A "`...`" sequence that begins, but does not end, within the same double-quoted  
 80149 string

80150 \ Outside of "\$(...)" and "\${...}" the <backslash> shall retain its special meaning as an  
 80151 escape character (see [Section 2.2.1](#)) only when immediately followed by one of the following  
 80152 characters:

80153 \$ ` \ <newline>

80154 or by a double-quote character that would otherwise be considered special (see [Section 2.6.4](#)  
 80155 (on page 2490) and [Section 2.7.4](#), on page 2495).

80156 When double-quotes are used to quote a parameter expansion, command substitution, or  
 80157 arithmetic expansion, the literal value of all characters within the result of the expansion shall be  
 80158 preserved.

80159 The application shall ensure that a double-quote that is not within "\$(...)" nor within  
 80160 "\${...}" is immediately preceded by a <backslash> in order to be included within double-  
 80161 quotes. The parameter '@' has special meaning inside double-quotes and is described in [Section](#)  
 80162 [2.5.2](#).

## 80163 2.2.4 Dollar-Single-Quotes

80164 A sequence of characters starting with a <dollar-sign> immediately followed by a single-quote  
 80165 (\$') shall preserve the literal value of all characters up to an unescaped terminating single-quote  
 80166 ('), with the exception of certain <backslash>-escape sequences, as follows:

- 80167 • \" yields a <quotation-mark> (double-quote) character, but note that <quotation-mark>  
 80168 can be included unescaped.
- 80169 • \' yields an <apostrophe> (single-quote) character.
- 80170 • \\ yields a <backslash> character.
- 80171 • \a yields an <alert> character.
- 80172 • \b yields a <backspace> character.
- 80173 • \e yields an <ESC> character.
- 80174 • \f yields a <form-feed> character.
- 80175 • \n yields a <newline> character.
- 80176 • \r yields a <carriage-return> character.

- 80177           • `\t` yields a <tab> character.
- 80178           • `\v` yields a <vertical-tab> character.
- 80179           • `\cX` yields the control character listed in the **Value** column of [Table 3-21](#) in the OPERANDS section of the `stty` utility when *X* is one of the characters listed in the **^c** column of the same table, except that `\c\\` yields the <FS> control character since the <backslash> character has to be escaped.
- 80180
- 80181
- 80182
- 80183           • `\xXX` yields the byte whose value is the hexadecimal value *XX* (one or more hexadecimal digits). If more than two hexadecimal digits follow `\x`, the results are unspecified.
- 80184
- 80185           • `\ddd` yields the byte whose value is the octal value *ddd* (one to three octal digits).
- 80186           • The behavior of an unescaped <backslash> immediately followed by any other character, including <newline>, is unspecified.
- 80187

80188           In cases where a variable number of characters can be used to specify an escape sequence (`\xXX` and `\ddd`), the escape sequence shall be terminated by the first character that is not of the expected type or, for `\ddd` sequences, when the maximum number of characters specified has been found, whichever occurs first.

80192           These <backslash>-escape sequences shall be processed (replaced with the bytes or characters they yield) immediately prior to word expansion (see [Section 2.6](#)) of the word in which the dollar-single-quotes sequence occurs.

80195           If a `\xXX` or `\ddd` escape sequence yields a byte whose value is 0, it is unspecified whether that null byte is included in the result or if that byte and any following regular characters and escape sequences up to the terminating unescaped single-quote are evaluated and discarded.

80198           If the octal value specified by `\ddd` will not fit in a byte, the results are unspecified.

80199           If a `\e` or `\cX` escape sequence specifies a character that does not have an encoding in the locale in effect when these <backslash>-escape sequences are processed, the result is implementation-defined. However, implementations shall not replace an unsupported character with bytes that do not form valid characters in that locale's character set.

80203           If a <backslash>-escape sequence represents a single-quote character (for example `\'`), that sequence shall not terminate the dollar-single-quote sequence.

## 80205   2.3   Token Recognition

80206           The shell shall read its input in terms of lines. (For details about how the shell reads its input, see the description of *sh*.) The input lines can be of unlimited length. These lines shall be parsed using two major modes: ordinary token recognition and processing of here-documents.

80209           When an **io\_here** token has been recognized by the grammar (see [Section 2.10](#)), one or more of the subsequent lines immediately following the next **NEWLINE** token form the body of a here-document and shall be parsed according to the rules of [Section 2.7.4](#). Any non-**NEWLINE** tokens (including more **io\_here** tokens) that are recognized while searching for the next **NEWLINE** token shall be saved for processing after the here-document has been parsed. If a saved token is an **io\_here** token, the corresponding here-document shall start on the line immediately following the line containing the trailing delimiter of the previous here-document. If any saved token includes a <newline> character, the behavior is unspecified.

80217           When it is not processing an **io\_here**, the shell shall break its input into tokens by applying the first applicable rule below to each character in turn in its input. At the start of input or after a previous token has just been delimited, the first or next token, respectively, shall start with the first character that has not already been included in a token and is not discarded according to

80221 the rules below. Once a token has started, zero or more characters from the input shall be  
80222 appended to the token until the end of the token is delimited according to one of the rules below.  
80223 When both the start and end of a token have been delimited, the characters forming the token  
80224 shall be exactly those in the input between the two delimiters, including any quoting characters.  
80225 If a rule below indicates that a token is delimited, and no characters have been included in the  
80226 token, that empty token shall be discarded.

- 80227 1. If the end of input is recognized, the current token (if any) shall be delimited.
- 80228 2. If the previous character was used as part of an operator and the current character is not  
80229 quoted and can be used with the previous characters to form an operator, it shall be used  
80230 as part of that (operator) token.
- 80231 3. If the previous character was used as part of an operator and the current character cannot  
80232 be used with the previous characters to form an operator, the operator containing the  
80233 previous character shall be delimited.
- 80234 4. If the current character is an unquoted <backslash>, single-quote, or double-quote or is  
80235 the first character of an unquoted <dollar-sign> single-quote sequence, it shall affect  
80236 quoting for subsequent characters up to the end of the quoted text. The rules for quoting  
80237 are as described in [Section 2.2](#). During token recognition no substitutions shall be  
80238 actually performed, and the result token shall contain exactly the characters that appear  
80239 in the input unmodified, including any embedded or enclosing quotes or substitution  
80240 operators, between the start and the end of the quoted text. The token shall not be  
80241 delimited by the end of the quoted field.
- 80242 5. If the current character is an unquoted '\$' or '`', the shell shall identify the start of any  
80243 candidates for parameter expansion ([Section 2.6.2](#)), command substitution ([Section 2.6.3](#)),  
80244 or arithmetic expansion ([Section 2.6.4](#)) from their introductory unquoted character  
80245 sequences: '\${' or '\$(', '\$(' or '`', and '\$(', respectively. The shell shall read  
80246 sufficient input to determine the end of the unit to be expanded (as explained in the cited  
80247 sections). While processing the characters, if instances of expansions or quoting are  
80248 found nested within the substitution, the shell shall recursively process them in the  
80249 manner specified for the construct that is found. For '\$(' and '`' only, if instances of  
80250 **io\_here** tokens are found nested within the substitution, they shall be parsed according to  
80251 the rules of [Section 2.7.4](#); if the terminating ')' or '`' of the substitution occurs before  
80252 the **NEWLINE** token marking the start of the here-document, the behavior is unspecified.  
80253 The characters found from the beginning of the substitution to its end, allowing for any  
80254 recursion necessary to recognize embedded constructs, shall be included unmodified in  
80255 the result token, including any embedded or enclosing substitution operators or quotes.  
80256 The token shall not be delimited by the end of the substitution.
- 80257 6. If the current character is not quoted and can be used as the first character of a new  
80258 operator, the current token (if any) shall be delimited. The current character shall be used  
80259 as the beginning of the next (operator) token.
- 80260 7. If the current character is an unquoted <blank>, any token containing the previous  
80261 character is delimited and the current character shall be discarded.
- 80262 8. If the previous character was part of a word, the current character shall be appended to  
80263 that word.
- 80264 9. If the current character is a '#', it and all subsequent characters up to, but excluding, the  
80265 next <newline> shall be discarded as a comment. The <newline> that ends the line is not  
80266 considered part of the comment.

80267 10. The current character is used as the start of a new word.

80268 Once a token is delimited, it is categorized as required by the grammar in [Section 2.10](#).

80269 In situations where the shell parses its input as a *program*, once a *complete\_command* has been  
80270 recognized by the grammar (see [Section 2.10](#)), the *complete\_command* shall be executed before the  
80271 next *complete\_command* is tokenized and parsed.

### 80272 2.3.1 Alias Substitution

80273 After a token has been categorized as type **TOKEN** (see [Section 2.10.1](#)), including (recursively)  
80274 any token resulting from an alias substitution, the **TOKEN** shall be subject to alias substitution if  
80275 all of the following conditions are true:

- 80276 • The **TOKEN** does not contain any quoting characters.
- 80277 • The **TOKEN** is a valid alias name (see [XBD Section 3.10](#)).
- 80278 • An alias with that name is in effect.
- 80279 • The **TOKEN** did not either fully or, optionally, partially result from an alias substitution of  
80280 the same alias name at any earlier recursion level.
- 80281 • Either the **TOKEN** is being considered for alias substitution because it follows an alias  
80282 substitution whose replacement value ended with a <blank> (see below) or the **TOKEN**  
80283 could be parsed as the command name word of a simple command (see [Section 2.10](#)),  
80284 based on this **TOKEN** and the tokens (if any) that preceded it, but ignoring whether any  
80285 subsequent characters would allow that.

80286 except that if the **TOKEN** meets the above conditions and would be recognized as a reserved  
80287 word (see [Section 2.4](#)) if it occurred in an appropriate place in the input, it is unspecified  
80288 whether the **TOKEN** is subject to alias substitution.

80289 When a **TOKEN** is subject to alias substitution, the value of the alias shall be processed as if it  
80290 had been read from the input instead of the **TOKEN**, with token recognition (see [Section 2.3](#))  
80291 resuming at the start of the alias value. When the end of the alias value is reached, the shell may  
80292 behave as if an additional <space> character had been read from the input after the **TOKEN** that  
80293 was replaced. If it does not add this <space>, it is unspecified whether the current token is  
80294 delimited before token recognition is applied to the character (if any) that followed the **TOKEN**  
80295 in the input.

80296 **Note:** A future version of this standard may disallow adding this <space>.

80297 If the value of the alias replacing the **TOKEN** ends in a <blank> that would be unquoted after  
80298 substitution, and optionally if it ends in a <blank> that would be quoted after substitution, the  
80299 shell shall check the next token in the input, if it is a **TOKEN**, for alias substitution; this process  
80300 shall continue until a **TOKEN** is found that is not a valid alias or an alias value does not end in  
80301 such a <blank>.

80302 An implementation may defer the effect of a change to an alias but the change shall take effect  
80303 no later than the completion of the currently executing *complete\_command* (see [Section 2.10](#)).  
80304 Changes to aliases shall not take effect out of order. Implementations may provide predefined  
80305 aliases that are in effect when the shell is invoked.

80306 When used as specified by this volume of POSIX.1-2024, alias definitions shall not be inherited  
80307 by separate invocations of the shell or by the utility execution environments invoked by the  
80308 shell; see [Section 2.13](#).

## 80309 2.4 Reserved Words

80310 Reserved words are words that have special meaning to the shell; see [Section 2.9](#). The following  
80311 words shall be recognized as reserved words:

80312	<b>!</b>	<b>do</b>	<b>esac</b>	<b>in</b>
80313	<b>{</b>	<b>done</b>	<b>fi</b>	<b>then</b>
80314	<b>}</b>	<b>elif</b>	<b>for</b>	<b>until</b>
80315	<b>case</b>	<b>else</b>	<b>if</b>	<b>while</b>

80316 This recognition shall only occur when none of the characters is quoted and when the word is  
80317 used as:

- 80318 • The first word of a command
- 80319 • The first word following one of the reserved words other than **case**, **for**, or **in**
- 80320 • The third word in a **case** command (only **in** is valid in this case)
- 80321 • The third word in a **for** command (only **in** and **do** are valid in this case)

80322 See the grammar in [Section 2.10](#).

80323 When used in circumstances where reserved words are recognized (described above), the  
80324 following words may be recognized as reserved words, in which case the results are unspecified  
80325 except as described below for **time**:

80326	<b>[</b>	<b>]</b>	<b>function</b>	<b>namespace</b>	<b>select</b>	<b>time</b>
-------	----------	----------	-----------------	------------------	---------------	-------------

80327 When the word **time** is recognized as a reserved word in circumstances where it would, if it  
80328 were not a reserved word, be the command name (see [Section 2.9.1.1](#)) of a simple command that  
80329 would execute the *time* utility in a manner other than one for which *time* states that the results  
80330 are unspecified, the behavior shall be as specified for the *time* utility.

80331 When used in circumstances where reserved words are recognized (described above), all words  
80332 whose final character is a <colon> (':') are reserved; their use in those circumstances produces  
80333 unspecified results.

## 80334 2.5 Parameters and Variables

80335 A parameter can be denoted by a name, a number, or one of the special characters listed in  
80336 [Section 2.5.2](#). A variable is a parameter denoted by a name.

80337 A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can  
80338 only be unset by using the *unset* special built-in command.

80339 Parameters can contain arbitrary byte sequences, except for the null byte. The shell shall process  
80340 their values as characters only when performing operations that are described in this standard in  
80341 terms of characters.

80342 **2.5.1 Positional Parameters**

80343 A positional parameter is a parameter denoted by a decimal representation of a positive integer.  
 80344 The digits denoting the positional parameters shall always be interpreted as a decimal value,  
 80345 even if there is a leading zero. When a positional parameter with more than one digit is  
 80346 specified, the application shall enclose the digits in braces (see [Section 2.6.2](#)).

80347 Examples:

- 80348 • "\$8", "\${8}", "\${08}", "\${008}", etc. all expand to the value of the eighth positional  
 80349 parameter.
- 80350 • "\${10}" expands to the value of the tenth positional parameter.
- 80351 • "\$10" expands to the value of the first positional parameter followed by the character '0'.

80352 **Note:** 0 is a special parameter, not a positional parameter, and therefore the results of expanding  
 80353 \${00} are unspecified.

80354 Positional parameters are initially assigned when the shell is invoked (see *sh*), temporarily  
 80355 replaced when a shell function is invoked (see [Section 2.9.5](#)), and can be reassigned with the *set*  
 80356 special built-in command.

80357 **2.5.2 Special Parameters**

80358 Listed below are the special parameters and the values to which they shall expand. Only the  
 80359 values of the special parameters are listed; see [Section 2.6](#) for a detailed summary of all the  
 80360 stages involved in expanding words.

80361 @ Expands to the positional parameters, starting from one, initially producing one field for  
 80362 each positional parameter that is set. When the expansion occurs in a context where field  
 80363 splitting will be performed, any empty fields may be discarded and each of the non-empty  
 80364 fields shall be further split as described in [Section 2.6.5](#). When the expansion occurs within  
 80365 double-quotes, the behavior is unspecified unless one of the following is true:

- 80366 • Field splitting as described in [Section 2.6.5](#) would be performed if the expansion were  
 80367 not within double-quotes (regardless of whether field splitting would have any effect;  
 80368 for example, if *IFS* is null).
- 80369 • The double-quotes are within the *word* of a  $\${parameter:-word}$  or a  $\${parameter:+word}$   
 80370 expansion (with or without the <colon>; see [Section 2.6.2](#)) which would have been  
 80371 subject to field splitting if *parameter* had been expanded instead of *word*.

80372 If one of these conditions is true, the initial fields shall be retained as separate fields, except  
 80373 that if the parameter being expanded was embedded within a word, the first field shall be  
 80374 joined with the beginning part of the original word and the last field shall be joined with the  
 80375 end part of the original word. In all other contexts the results of the expansion are  
 80376 unspecified. If there are no positional parameters, the expansion of '@' shall generate zero  
 80377 fields, even when '@' is within double-quotes; however, if the expansion is embedded  
 80378 within a word which contains one or more other parts that expand to a quoted null string,  
 80379 these null string(s) shall still produce an empty field, except that if the other parts are all  
 80380 within the same double-quotes as the '@', it is unspecified whether the result is zero fields  
 80381 or one empty field.

80382 \* Expands to the positional parameters, starting from one, initially producing one field for  
 80383 each positional parameter that is set. When the expansion occurs in a context where field  
 80384 splitting will be performed, any empty fields may be discarded and each of the non-empty  
 80385 fields shall be further split as described in [Section 2.6.5](#). When the expansion occurs in a

- 80386 context where field splitting will not be performed, the initial fields shall be joined to form a  
 80387 single field with the value of each parameter separated by the first character of the *IFS*  
 80388 variable if *IFS* contains at least one character, or separated by a <space> if *IFS* is unset, or  
 80389 with no separation if *IFS* is set to a null string.
- 80390 # Expands to the shortest representation of the decimal number of positional parameters. The  
 80391 command name (parameter 0) shall not be counted in the number given by '# ' because it is  
 80392 a special parameter, not a positional parameter.
- 80393 ? Expands to the shortest representation of the decimal exit status (see [Section 2.8.2](#)) of the  
 80394 pipeline (see [Section 2.9.2](#)) executed from the current shell execution environment (not a  
 80395 subshell environment) that most recently either terminated or, optionally but only if the  
 80396 shell is interactive and job control is enabled, was stopped by a signal. If this pipeline  
 80397 terminated, the status value shall be its exit status; otherwise, the status value shall be the  
 80398 same as the exit status that would have resulted if the pipeline had been terminated by a  
 80399 signal with the same number as the signal that stopped it. The value of the special  
 80400 parameter '?' shall be set to 0 during initialization of the shell. When a subshell  
 80401 environment is created, the value of the special parameter '?' from the invoking shell  
 80402 environment shall be preserved in the subshell.
- 80403 **Note:** In `var=$(some_command); echo $?` the output is the exit status of `some_command`,  
 80404 which is executed in a subshell environment, but this is because its exit status becomes the  
 80405 exit status of the assignment command `var=$(some_command)` (see [Section 2.9.1](#)) and  
 80406 this assignment command is the most recently completed pipeline. Likewise for any  
 80407 pipeline consisting entirely of a simple command that has no command word, but contains  
 80408 one or more command substitutions. (See [Section 2.9.1](#).)
- 80409 – (Hyphen.) Expands to the current option flags (the single-letter option names concatenated  
 80410 into a string) as specified on invocation, by the *set* special built-in command, or implicitly  
 80411 by the shell. It is unspecified whether the `-c` and `-s` options are included in the expansion  
 80412 of "\$-". The `-i` option shall be included in "\$-" if the shell is interactive, regardless of  
 80413 whether it was specified on invocation.
- 80414 \$ Expands to the shortest representation of the decimal process ID of the invoked shell. In a  
 80415 subshell (see [Section 2.13](#)), '\$ ' shall expand to the same value as that of the current shell.
- 80416 ! Expands to the shortest representation of the decimal process ID associated with the most  
 80417 recent asynchronous AND-OR list (see [Section 2.9.3.1](#)) executed from the current shell  
 80418 execution environment, or to the shortest representation of the decimal process ID of the  
 80419 last command specified in the currently executing pipeline in the job-control background  
 80420 job that most recently resumed execution through the use of *bg*, whichever is the most  
 80421 recent.
- 80422 0 (Zero.) Expands to the name of the shell or shell script. See *sh* for a detailed description of  
 80423 how this name is derived.
- 80424 See the description of the *IFS* variable in [Section 2.5.3](#).



80425 **2.5.3 Shell Variables**

80426 Variables shall be initialized from the environment (as defined by XBD Chapter 8 and the *exec*  
 80427 function in the System Interfaces volume of POSIX.1-2024) and can be given new values with  
 80428 variable assignment commands. Shell variables shall be initialized only from environment  
 80429 variables that have valid names. If a variable is initialized from the environment, it shall be  
 80430 marked for export immediately; see the *export* special built-in. New variables can be defined and  
 80431 initialized with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter  
 80432 in a **for** loop, with the  $\${name=word}$  expansion, or with other mechanisms provided as  
 80433 implementation extensions.

80434 The following variables shall affect the execution of the shell:

80435 UP **ENV** The processing of the *ENV* shell variable shall be supported if the system  
 80436 supports the User Portability Utilities option.

80437 This variable, when and only when an interactive shell is invoked, shall be  
 80438 subjected to parameter expansion (see Section 2.6.2) by the shell and the  
 80439 resulting value shall be used as a pathname of a file. Before any interactive  
 80440 commands are read, the shell shall tokenize (see Section 2.3) the contents of  
 80441 the file, parse the tokens as a *program* (see Section 2.10), and execute the  
 80442 resulting commands in the current environment. (In other words, the contents  
 80443 of the *ENV* file are not parsed as a single *compound\_list*. This distinction  
 80444 matters because it influences when aliases take effect.) The file need not be  
 80445 executable. If the expanded value of *ENV* is not an absolute pathname, the  
 80446 results are unspecified. *ENV* shall be ignored if the user's real and effective  
 80447 user IDs or real and effective group IDs are different.

80448 **HOME** The pathname of the user's home directory. The contents of *HOME* are used in  
 80449 tilde expansion (see Section 2.6.1).

80450 **IFS** A string treated as a list of characters that is used for field splitting, expansion  
 80451 of the '\*' special parameter, and to split lines into fields with the *read* utility.  
 80452 If the value of *IFS* includes any bytes that do not form part of a valid character,  
 80453 the results of field splitting, expansion of '\*', and use of the *read* utility are  
 80454 unspecified.

80455 If *IFS* is not set, it shall behave as normal for an unset variable, except that  
 80456 field splitting by the shell and line splitting by the *read* utility shall be  
 80457 performed as if the value of *IFS* is <space><tab><newline>; see Section 2.6.5.

80458 The shell shall set *IFS* to <space><tab><newline> when it is invoked.

80459 **LANG** Provide a default value for the internationalization variables that are unset or  
 80460 null. (See XBD Section 8.2 for the precedence of internationalization variables  
 80461 used to determine the values of locale categories.)

80462 **LC\_ALL** The value of this variable overrides the *LC\_\** variables and *LANG*, as  
 80463 described in XBD Chapter 8.

80464 **LC\_COLLATE** Determine the behavior of range expressions, equivalence classes, and multi-  
 80465 character collating elements within pattern matching.

80466 **LC\_CTYPE** Determine the interpretation of sequences of bytes of text data as characters  
 80467 (for example, single-byte as opposed to multi-byte characters), which  
 80468 characters are defined as letters (character class **alpha**) and <blank> characters  
 80469 (character class **blank**), and the behavior of character classes within pattern  
 80470 matching. Changing the value of *LC\_CTYPE* after the shell has started shall  
 80471 not affect the lexical processing of shell commands in the current shell

80472			execution environment or its subshells. Invoking a shell script or performing
80473			<code>exec sh</code> subjects the new shell to the changes in <code>LC_CTYPE</code> .
80474		<code>LC_MESSAGES</code>	Determine the language in which messages should be written.
80475	UP	<code>LINENO</code>	The processing of the <code>LINENO</code> shell variable shall be supported if the system
80476			supports the User Portability Utilities option.
80477			Set by the shell to a decimal number representing the current sequential line
80478			number (numbered starting with 1) within a script or function before it
80479			executes each command. If the user unsets or resets <code>LINENO</code> , the variable may
80480			lose its special meaning for the life of the shell. If the shell is not currently
80481			executing a script or function, the value of <code>LINENO</code> is unspecified.
80482	XSI	<code>NLSPATH</code>	Determine the location of message catalogs for the processing of
80483			<code>LC_MESSAGES</code> .
80484		<code>PATH</code>	A string formatted as described in XBD Chapter 8, used to effect command
80485			interpretation; see Section 2.9.1.4.
80486		<code>PPID</code>	Set by the shell to the decimal value of its parent process ID during
80487			initialization of the shell. In a subshell (see Section 2.13), <code>PPID</code> shall be set to
80488			the same value as that of the parent of the current shell. For example, <code>echo</code>
80489			<code>\$PPID</code> and ( <code>echo \$PPID</code> ) would produce the same value.
80490	UP	<code>PS1</code>	The processing of the <code>PS1</code> shell variable shall be supported if the system
80491			supports the User Portability Utilities option.
80492			Each time an interactive shell is ready to read a command, the value of this
80493			variable shall be subjected to parameter expansion (see Section 2.6.2) and
80494			exclamation-mark expansion (see below). Whether the value is also subjected
80495			to command substitution (see Section 2.6.3) or arithmetic expansion (see
80496			Section 2.6.4) or both is unspecified. After expansion, the value shall be
80497			written to standard error.
80498			The expansions shall be performed in two passes, where the result of the first
80499			pass is input to the second pass. One of the passes shall perform only the
80500			exclamation-mark expansion described below. The other pass shall perform
80501			the other expansion(s) according to the rules in Section 2.6. Which of the two
80502			passes is performed first is unspecified.
80503			The default value shall be <code>"\$ "</code> . For users who have specific additional
80504			implementation-defined privileges, the default may be another,
80505			implementation-defined value.
80506			Exclamation-mark expansion: The shell shall replace each instance of the
80507			<exclamation-mark> character ( <code>'!'</code> ) with the history file number (see
80508			Command History List) of the next command to be typed. An <exclamation-
80509			mark> character escaped by another <exclamation-mark> character (that is,
80510			<code>"!!"</code> ) shall expand to a single <exclamation-mark> character.
80511	UP	<code>PS2</code>	The processing of the <code>PS2</code> shell variable shall be supported if the system
80512			supports the User Portability Utilities option.
80513			Each time the user enters a <newline> prior to completing a command line in
80514			an interactive shell, the value of this variable shall be subjected to parameter
80515			expansion (see Section 2.6.2). Whether the value is also subjected to command
80516			substitution (see Section 2.6.3) or arithmetic expansion (see Section 2.6.4) or
80517			both is unspecified. After expansion, the value shall be written to standard

80518		error. The default value shall be "> ".
80519	UP	<i>PS4</i>
80520		The processing of the <i>PS4</i> shell variable shall be supported if the system supports the User Portability Utilities option.
80521		When an execution trace ( <i>set -x</i> ) is being performed, before each line in the execution trace, the value of this variable shall be subjected to parameter expansion (see <a href="#">Section 2.6.2</a> ). Whether the value is also subjected to command substitution (see <a href="#">Section 2.6.3</a> ) or arithmetic expansion (see <a href="#">Section 2.6.4</a> ) or both is unspecified. After expansion, the value shall be written to standard error. The default value shall be "+ ".
80522		
80523		
80524		
80525		
80526		
80527		<i>PWD</i>
80528		Set by the shell and by the <i>cd</i> utility. In the shell the value shall be initialized from the environment as follows. If a value for <i>PWD</i> is passed to the shell in the environment when it is executed, the value is an absolute pathname of the current working directory that is no longer than { <i>PATH_MAX</i> } bytes including the terminating null byte, and the value does not contain any components that are dot or dot-dot, then the shell shall set <i>PWD</i> to the value from the environment. Otherwise, if a value for <i>PWD</i> is passed to the shell in the environment when it is executed, the value is an absolute pathname of the current working directory, and the value does not contain any components that are dot or dot-dot, then it is unspecified whether the shell sets <i>PWD</i> to the value from the environment or sets <i>PWD</i> to the pathname that would be output by <i>pwd -P</i> . Otherwise, the <i>sh</i> utility sets <i>PWD</i> to the pathname that would be output by <i>pwd -P</i> . In cases where <i>PWD</i> is set to the value from the environment, the value can contain components that refer to files of type symbolic link. In cases where <i>PWD</i> is set to the pathname that would be output by <i>pwd -P</i> , if there is insufficient permission on the current working directory, or on any parent of that directory, to determine what that pathname would be, the value of <i>PWD</i> is unspecified. Assignments to this variable may be ignored. If an application sets or unsets the value of <i>PWD</i> , the behaviors of the <i>cd</i> and <i>pwd</i> utilities are unspecified.
80529		
80530		
80531		
80532		
80533		
80534		
80535		
80536		
80537		
80538		
80539		
80540		
80541		
80542		
80543		
80544		
80545		
80546		

## 80547 2.6 Word Expansions

80548 This section describes the various expansions that are performed on words. Not all expansions  
 80549 are performed on every word, as explained in the following sections and elsewhere in this  
 80550 chapter. The expansions that are performed for a given word shall be performed in the following  
 80551 order:

- 80552 1. Tilde expansion (see [Section 2.6.1](#)), parameter expansion (see [Section 2.6.2](#)), command  
 80553 substitution (see [Section 2.6.3](#)), and arithmetic expansion (see [Section 2.6.4](#)) shall be  
 80554 performed, beginning to end. See item 5 in [Section 2.3](#).
- 80555 2. Field splitting (see [Section 2.6.5](#)) shall be performed on the portions of the fields  
 80556 generated by step 1.
- 80557 3. Pathname expansion (see [Section 2.6.6](#)) shall be performed, unless *set -f* is in effect.
- 80558 4. Quote removal (see [Section 2.6.7](#)), if performed, shall always be performed last.

80559 Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and  
 80560 quote removals that occur within a single word shall expand to a single field, except as  
 80561 described below. The shell shall create multiple fields or no fields from a single word only as a  
 80562 result of field splitting, pathname expansion, or the following cases:

80563 1. Parameter expansion of the special parameters '@' and '\*', as described in [Section 2.5.2](#),  
80564 can create multiple fields or no fields from a single word.

80565 2. When the expansion occurs in a context where field splitting will be performed, a word  
80566 that contains all of the following somewhere within it, before any expansions are applied,  
80567 in the order specified:

- 80568 • an unquoted <left-curly-bracket> ('{') that is not immediately preceded by an  
80569 unquoted <dollar-sign> ('\$')

- 80570 • one or more unquoted <comma> (',') characters or a sequence that consists of two  
80571 adjacent <period> ('.') characters surrounded by other characters (which can also  
80572 be <period> characters)

- 80573 • an unquoted <right-curly-bracket> ('}') )

80574 may be subject to an additional implementation-defined form of expansion that can create  
80575 multiple fields from a single word. This expansion, if supported, shall be applied before  
80576 all the other word expansions are applied. The other expansions shall then be applied to  
80577 each field that results from this expansion.

80578 When the expansions in this section are performed other than in the context of preparing a  
80579 command for execution, they shall be carried out in the current shell execution environment.

80580 When expanding words for a command about to be executed, and the word will be the  
80581 command name or an argument to the command, the expansions shall be carried out in the  
80582 current shell execution environment. (The environment for the command to be executed is  
80583 unknown until the command word is known.)

80584 When expanding the words in a command about to be executed that are used with variable  
80585 assignments or redirections, it is unspecified whether the expansions are carried out in the  
80586 current execution environment or in the environment of the command about to be executed.

80587 The '\$' character is used to introduce parameter expansion, command substitution, or  
80588 arithmetic evaluation. If a '\$' that is neither within single-quotes nor escaped by a <backslash>  
80589 is immediately followed by a character that is not a <space>, not a <tab>, not a <newline>, and  
80590 is not one of the following:

- 80591 • A numeric character
- 80592 • The name of one of the special parameters (see [Section 2.5.2](#))
- 80593 • A valid first character of a variable name
- 80594 • A <left-curly-bracket> ('{')
- 80595 • A <left-parenthesis>
- 80596 • A single-quote

80597 the result is unspecified. If a '\$' that is neither within single-quotes nor escaped by a  
80598 <backslash> is immediately followed by a <space>, <tab>, or a <newline>, or is not followed by  
80599 any character, the '\$' shall be treated as a literal character.

80600 **2.6.1 Tilde Expansion**

80601 A ``tilde-prefix'' consists of an unquoted <tilde> character at the beginning of a word, followed  
 80602 by all of the characters preceding the first unquoted <slash> in the word, or all the characters in  
 80603 the word if there is no <slash>. In an assignment (see XBD Section 4.26), multiple tilde-prefixes  
 80604 can be used: one at the beginning of the word (that is, following the <equals-sign> of the  
 80605 assignment), or one following any unquoted <colon>, or both. A tilde-prefix in an assignment is  
 80606 terminated by the first unquoted <colon> or <slash>, or the end of the assignment word.

80607 If the tilde-prefix consists of only the <tilde> character, it shall be replaced by the value of the  
 80608 variable *HOME*. If *HOME* is unset, the results are unspecified.

80609 Otherwise, the characters in the tilde-prefix following the <tilde> shall be treated as a possible  
 80610 login name from the user database. If these characters do not form a portable login name (see  
 80611 the description of the *LOGNAME* environment variable in XBD Section 8.3), the results are  
 80612 unspecified.

80613 **Note:** Since the tilde-prefix is not subject to further word expansions after the <tilde> is removed to  
 80614 obtain the login name, none of the following has a portable login name following the <tilde>:

```
80615 ~"string"
80616 ~'string'
80617 ~$var
80618 ~\bin
```

80619 owing to the presence of ' ', '\ ', '\$ ', '\\ ', and '/' characters in the login name.

80620 If the characters in the tilde-prefix following the <tilde> form a portable login name, the tilde-  
 80621 prefix shall be replaced by a pathname of the initial working directory associated with the login  
 80622 name. The pathname shall be obtained as if by using the *getpwnam()* function as defined in the  
 80623 System Interfaces volume of POSIX.1-2024. If the system does not recognize the login name, the  
 80624 results are unspecified.

80625 The pathname that replaces the tilde-prefix shall be treated as if quoted to prevent it being  
 80626 altered by field splitting and pathname expansion; if a <slash> follows the tilde-prefix and the  
 80627 pathname ends with a <slash>, the trailing <slash> from the pathname should be omitted from  
 80628 the replacement. If the word being expanded consists of only the <tilde> character and *HOME*  
 80629 is set to the null string, this produces an empty field (as opposed to zero fields) as the expanded  
 80630 word.

80631 **Note:** A future version of this standard may require that if a <slash> follows the tilde-prefix and the  
 80632 pathname ends with a <slash>, the trailing <slash> from the pathname is omitted from the  
 80633 replacement.

80634 **2.6.2 Parameter Expansion**

80635 The format for parameter expansion is as follows:

```
80636 ${expression}
```

80637 where *expression* consists of all characters until the matching '}'. Any '}' escaped by a  
 80638 <backslash> or within a quoted string, and characters in embedded arithmetic expansions,  
 80639 command substitutions, and variable expansions, shall not be examined in determining the  
 80640 matching '}'.

80641 The simplest form for parameter expansion is:

```
80642 ${parameter}
```

- 80643 The value, if any, of *parameter* shall be substituted.
- 80644 The parameter name or symbol can be enclosed in braces, which are optional except for  
80645 positional parameters with more than one digit or when *parameter* is a name and is followed by a  
80646 character that could be interpreted as part of the name.
- 80647 For a parameter that is not enclosed in braces:
- 80648 • If the parameter is a name, the expansion shall use the longest valid name (see XBD  
80649 [Section 3.216](#)), whether or not the variable denoted by that name exists.
  - 80650 • Otherwise, the parameter is a single-character symbol, and behavior is unspecified if that  
80651 character is neither a digit nor one of the special parameters (see [Section 2.5.2](#)).
- 80652 In addition, a parameter expansion can be modified by using one of the following formats. In  
80653 each case that a value of *word* is needed (based on the state of *parameter*, as described below),  
80654 *word* shall be subjected to tilde expansion, parameter expansion, command substitution,  
80655 arithmetic expansion, and quote removal. If *word* is not needed, it shall not be expanded. The  
80656 '}' character that delimits the following parameter expansion modifications shall be  
80657 determined as described previously in this section and in [Section 2.2.3](#). If *parameter* is '\*' or  
80658 '@', the result of the expansion is unspecified.
- 80659  $\${parameter}:-[word]$  **Use Default Values.** If *parameter* is unset or null, the expansion of *word*  
80660 (or an empty string if *word* is omitted) shall be substituted; otherwise, the  
80661 value of *parameter* shall be substituted.
- 80662  $\${parameter}:= [word]$  **Assign Default Values.** If *parameter* is unset or null, quote removal shall  
80663 be performed on the expansion of *word* and the result (or an empty string  
80664 if *word* is omitted) shall be assigned to *parameter*. In all cases, the final  
80665 value of *parameter* shall be substituted. Only variables, not positional  
80666 parameters or special parameters, can be assigned in this way.
- 80667  $\${parameter}?:[word]$  **Indicate Error if Null or Unset.** If *parameter* is unset or null, the  
80668 expansion of *word* (or a message indicating it is unset if *word* is omitted)  
80669 shall be written to standard error and the shell exits with a non-zero exit  
80670 status. Otherwise, the value of *parameter* shall be substituted. An  
80671 interactive shell need not exit.
- 80672  $\${parameter}+ [word]$  **Use Alternative Value.** If *parameter* is unset or null, null shall be  
80673 substituted; otherwise, the expansion of *word* (or an empty string if *word*  
80674 is omitted) shall be substituted.
- 80675 In the parameter expansions shown previously, use of the <colon> in the format shall result in a  
80676 test for a parameter that is unset or null; omission of the <colon> shall result in a test for a  
80677 parameter that is only unset. If *parameter* is '#' and the colon is omitted, the application shall  
80678 ensure that *word* is specified (this is necessary to avoid ambiguity with the string length  
80679 expansion). The following table summarizes the effect of the <colon>:

	<i>parameter</i> <b>Set and Not Null</b>	<i>parameter</i> <b>Set But Null</b>	<i>parameter</i> <b>Unset</b>
80680	<code>\${parameter:-word}</code>	substitute <i>parameter</i>	substitute <i>word</i>
80681	<code>\${parameter-word}</code>	substitute <i>parameter</i>	substitute <i>word</i>
80682	<code>\${parameter:=word}</code>	substitute <i>parameter</i>	assign <i>word</i>
80683	<code>\${parameter=word}</code>	substitute <i>parameter</i>	assign <i>word</i>
80684	<code>\${parameter?word}</code>	substitute <i>parameter</i>	error, exit
80685	<code>\${parameter?word}</code>	substitute <i>parameter</i>	error, exit
80686	<code>\${parameter+word}</code>	substitute <i>word</i>	substitute null
80687	<code>\${parameter+word}</code>	substitute <i>word</i>	substitute null

In all cases shown with “substitute”, the expression is replaced with the value shown. In all cases shown with “assign”, *parameter* is assigned that value, which also replaces the expression.

`#{parameter}` **String Length.** The shortest decimal representation of the length in characters of the value of *parameter* shall be substituted. If *parameter* is '\*' or '@', the result of the expansion is unspecified. If *parameter* is unset and *set -u* is in effect, the expansion shall fail.

The following four varieties of parameter expansion provide for character substring processing. In each case, pattern matching notation (see Section 2.14), rather than regular expression notation, shall be used to evaluate the patterns. If *parameter* is '#', '\*', or '@', the result of the expansion is unspecified. If *parameter* is unset and *set -u* is in effect, the expansion shall fail. Enclosing the full parameter expansion string in double-quotes shall not cause the following four varieties of pattern characters to be quoted, whereas quoting characters within the braces shall have this effect. In each variety, if *word* is omitted, the empty pattern shall be used.

`{parameter%[word]}` **Remove Smallest Suffix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the suffix matched by the *pattern* deleted. If present, *word* shall not begin with an unquoted '%'.

`{parameter%%[word]}` **Remove Largest Suffix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the suffix matched by the *pattern* deleted.

`{parameter#[word]}` **Remove Smallest Prefix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the prefix matched by the *pattern* deleted. If present, *word* shall not begin with an unquoted '#'.

`{parameter##[word]}` **Remove Largest Prefix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the prefix matched by the *pattern* deleted.

## Examples

`{parameter}`

In this example, the effects of omitting braces are demonstrated.

a=1

set 2

echo \${a}b-\$ab-\${1}0-\${10}-\${10}

**1b--20--20**

```

80724     ${parameter-word}
80725         This example demonstrates the difference between unset and set to the empty string, as well
80726         as the rules for finding the delimiting close brace.

80727         foo=asdf
80728         echo ${foo-bar}xyz}
80729         asdfxyz}
80730         foo=
80731         echo ${foo-bar}xyz}
80732         xyz}
80733         unset foo
80734         echo ${foo-bar}xyz}
80735         barxyz}

80736     ${parameter:-word}
80737         In this example, ls is executed only if x is null or unset. (The  $(ls)$  command substitution
80738         notation is explained in Section 2.6.3.)
80739         ${x:-$(ls)}

80740     ${parameter:=word}
80741         unset X
80742         echo ${X:=abc}
80743         abc

80744     ${parameter:?word}
80745         unset posix
80746         echo ${posix:?}
80747         sh: posix: parameter null or not set

80748     ${parameter:+word}
80749         set a b c
80750         echo ${3:+posix}
80751         posix

80752     ${#parameter}
80753         HOME=/usr/posix
80754         echo ${#HOME}
80755         10

80756     ${parameter%word}
80757         x=file.c
80758         echo ${x%.c}.o
80759         file.o

80760     ${parameter%%word}
80761         x=posix/src/std
80762         echo ${x%%/*}
80763         posix

80764     ${parameter#word}
80765         x=$HOME/src/cmd
80766         echo ${x#$HOME}
80767         /src/cmd

```



```

80768     ${parameter##*word}
80769         x=/one/two/three
80770         echo ${x##*/}
80771         three

```

80772 The double-quoting of patterns is different depending on where the double-quotes are placed:

80773 "\$ {x#\*} " The <asterisk> is a pattern character.

80774 \${x#"\*" } The literal <asterisk> is quoted and not special.

### 80775 2.6.3 Command Substitution

80776 Command substitution allows the output of one or more commands to be substituted in place of  
80777 the commands themselves. Command substitution shall occur when command(s) are enclosed  
80778 as follows:

80779 \$ ( *commands* )

80780 or (backquoted version):

80781 ` *commands* `

80782 The shell shall expand the command substitution by executing *commands* in a subshell  
80783 environment (see [Section 2.13](#)) and replacing the command substitution (the text of the  
80784 *commands* string plus the enclosing "\$ () " or backquotes) with the standard output of the  
80785 command(s); if the output ends with one or more bytes that have the encoded value of a  
80786 <newline> character, they shall not be included in the replacement. Any such bytes that occur  
80787 elsewhere shall be included in the replacement; however, they might be treated as field  
80788 delimiters and eliminated during field splitting, depending on the value of *IFS* and quoting that  
80789 is in effect. If the output contains any null bytes, the behavior is unspecified.

80790 Within the backquoted style of command substitution, if the command substitution is not within  
80791 double-quotes, <backslash> shall retain its literal meaning, except when followed by: '\$', '`',  
80792 or <backslash>. See [Section 2.2.3](#) for the handling of <backslash> when the command  
80793 substitution is within double-quotes. The search for the matching backquote shall be satisfied  
80794 by the first unquoted non-escaped backquote; during this search, if a non-escaped backquote is  
80795 encountered within a shell comment, a here-document, an embedded command substitution of  
80796 the \$(*commands*) form, or a quoted string, undefined results occur. A quoted string that begins,  
80797 but does not end, within the "` . . . `" sequence produces undefined results.

80798 With the \$(*commands*) form, all characters following the open parenthesis to the matching closing  
80799 parenthesis constitute the *commands* string.

80800 With both the backquoted and \$(*commands*) forms, the *commands* string shall be tokenized (see  
80801 [Section 2.3](#)) and parsed (see [Section 2.10](#)). It is unspecified whether the *commands* string is  
80802 parsed and executed incrementally as a *program* (as for a shell script), or is parsed as a single  
80803 *compound\_list* that is executed after the string has been completely parsed. In addition, it is  
80804 unspecified whether the terminating ')' of the \$(*commands*) form can result from alias  
80805 substitution. With the \$(*commands*) form any syntactically correct *program* can be used for  
80806 *commands*, except that:

- 80807 • If the *commands* string consists solely of redirections, the results are unspecified.
- 80808 • If the *commands* string is parsed as a single *compound\_list*, before any commands are  
80809 executed, *alias* and *unalias* commands in *commands* have no effect during parsing (see  
80810 [Section 2.3.1](#)). Strictly conforming applications shall ensure that the *commands* string does

80811 not depend on alias changes taking effect incrementally as would be the case if parsed and  
80812 executed as a *program*.

- 80813 • The behavior is unspecified if the terminating ' ) ' is not present in the token containing  
80814 the command substitution; that is, if the ' ) ' is expected to result from alias substitution.

80815 The results of command substitution shall not be processed for further tilde expansion,  
80816 parameter expansion, command substitution, or arithmetic expansion.

80817 Command substitution can be nested. To specify nesting within the backquoted version, the  
80818 application shall precede the inner backquotes with <backslash> characters; for example:

80819 `\` commands \``

80820 The syntax of the shell command language has an ambiguity for expansions beginning with  
80821 "\$ ( ", which can introduce an arithmetic expansion or a command substitution that starts with  
80822 a subshell. Arithmetic expansion has precedence; that is, the shell shall first determine whether  
80823 it can parse the expansion as an arithmetic expansion and shall only parse the expansion as a  
80824 command substitution if it determines that it cannot parse the expansion as an arithmetic  
80825 expansion. The shell need not evaluate nested expansions when performing this determination.  
80826 If it encounters the end of input without already having determined that it cannot parse the  
80827 expansion as an arithmetic expansion, the shell shall treat the expansion as an incomplete  
80828 arithmetic expansion and report a syntax error. A conforming application shall ensure that it  
80829 separates the "\$ ( " and ' ( ' into two tokens (that is, separate them with white space) in a  
80830 command substitution that starts with a subshell. For example, a command substitution  
80831 containing a single subshell could be written as:

80832 `$ ( ( commands ) )`

#### 80833 2.6.4 Arithmetic Expansion

80834 Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and  
80835 substituting its value. The format for arithmetic expansion shall be as follows:

80836 `$ ( ( expression ) )`

80837 The expression shall be treated as if it were in double-quotes, except that a double-quote inside  
80838 the expression is not treated specially. The shell shall expand all tokens in the expression for  
80839 parameter expansion, command substitution, and quote removal.

80840 Next, the shell shall treat this as an arithmetic expression and substitute the value of the  
80841 expression. The arithmetic expression shall be processed according to the rules given in [Section](#)  
80842 [1.1.2.1](#), with the following exceptions:

- 80843 • Only signed long integer arithmetic is required.
- 80844 • Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in  
80845 the ISO C standard, Section 6.4.4.1 are required to be recognized as constants.
- 80846 • The *sizeof*( ) operator and the prefix and postfix "++" and "--" operators are not required.
- 80847 • Selection, iteration, and jump statements are not supported.

80848 All changes to variables in an arithmetic expression shall be in effect after the arithmetic  
80849 expansion, as in the parameter expansion "\${x=value}" .

80850 If the shell variable *x* contains a value that forms a valid integer constant, optionally including a  
80851 leading <plus-sign> or <hyphen-minus>, then the arithmetic expansions "\$ ( ( x ) )" and  
80852 "\$ ( ( \$ x ) )" shall return the same value.

80853 As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell  
 80854 may use a signed integer type with a rank larger than the rank of **signed long**. The shell may  
 80855 use a real-floating type instead of **signed long** as long as it does not affect the results in cases  
 80856 where there is no overflow. If the expression is invalid, or the contents of a shell variable used in  
 80857 the expression are not recognized by the shell, the expansion fails and the shell shall write a  
 80858 diagnostic message to standard error indicating the failure.

### 80859 **Examples**

80860 A simple example using arithmetic expansion:

```
80861 # repeat a command 100 times
80862 x=100
80863 while [ $x -gt 0 ]
80864 do
80865     command
80866     x=$((x-1))
80867 done
```

### 80868 **2.6.5 Field Splitting**

80869 After parameter expansion (Section 2.6.2), command substitution (Section 2.6.3), and arithmetic  
 80870 expansion (Section 2.6.4), if the shell variable *IFS* (see Section 2.5.3) is set and its value is not  
 80871 empty, or if *IFS* is unset, the shell shall scan each field containing results of expansions and  
 80872 substitutions that did not occur in double-quotes for field splitting; zero, one or multiple fields  
 80873 can result.

80874 For the remainder of this section, any reference to the results of an expansion, or results of  
 80875 expansions, shall be interpreted to mean the results from one or more unquoted variable or  
 80876 arithmetic expansions, or unquoted command substitutions.

80877 If the *IFS* variable is set and has an empty string as its value, no field splitting shall occur.  
 80878 However, if an input field which contained the results of an expansion is entirely empty, it shall  
 80879 be removed. Note that this occurs before quote removal; any input field that contains any  
 80880 quoting characters can never be empty at this point. After the removal of any such fields from  
 80881 the input, the possibly modified input field list shall become the output.

80882 Each input field shall be considered in sequence, first to last, with the results of the algorithm  
 80883 described in this section causing output fields to be generated, which shall remain in the same  
 80884 order as the input fields from which they originated.

80885 Fields which contain no results from expansions shall not be affected by field splitting, and shall  
 80886 remain unaltered, simply moving from the list of input fields to be next in the list of output  
 80887 fields.

80888 In the remainder of this description, it is assumed that there is present in the field at least one  
 80889 expansion result; this assumption will not be restated. Field splitting only ever alters those parts  
 80890 of the field.

80891 For the purposes of this section, the term “*IFS* white space” is used to mean any of the white-  
 80892 space bytes (see XBD Section 3.413 (on page 92), Section 3.414 (on page 92), and Section 3.415, on  
 80893 page 92) <space>, <tab>, or <newline> from the portable character set (see XBD Section 6.1, on  
 80894 page 117) which are present in the value of the *IFS* variable, and perhaps other white-space  
 80895 characters. It is implementation-defined whether other white-space characters which appear in

80896 the value of *IFS* are also considered as “*IFS* white space”. The three characters above specified as  
80897 *IFS* white-space bytes are always *IFS* white space, when they occur in the value of *IFS*,  
80898 regardless of whether they are white-space characters in any relevant locale. For other locale-  
80899 specific white-space characters allowed by the implementation it is unspecified whether the  
80900 character is considered as *IFS* white space if it is white space at the time it is assigned to the *IFS*  
80901 variable, or if it is white space at the time field splitting occurs. (The locale might have changed  
80902 between those events.)

80903 If the *IFS* variable is unset, then for the purposes of this section, but without altering the value of  
80904 the variable, its value shall be considered to contain the three single-byte characters <space>,  
80905 <tab>, and <newline> from the portable character set, all of which are *IFS* white-space  
80906 characters.

80907 The shell shall use the byte sequences that form the characters in the value of the *IFS* variable as  
80908 delimiters. Each of the characters <space>, <tab>, and <newline> which appears in the value of  
80909 *IFS* shall be a single-byte delimiter. The shell shall use these delimiters as field terminators to  
80910 split the results of expansions, along with other adjacent bytes, into separate fields, as described  
80911 below. Note that these delimiters terminate a field; they do not, of themselves, cause a new field  
80912 to start—subsequent bytes that are not from the results of an expansion, or that do not form *IFS*  
80913 white-space characters are required for a new field to begin.

80914 Note that the shell processes arbitrary bytes from the input fields; there is no requirement that  
80915 those bytes form valid characters.

80916 If the results of the algorithm are that no fields are delimited; that is, if the input field is wholly  
80917 empty or consists entirely of *IFS* white space, the result shall be zero fields (rather than an empty  
80918 field).

80919 For the purposes of this section, when a field is said to be delimited, then the candidate field, as  
80920 generated below shall become an output field. When the algorithm transforms a candidate into  
80921 an output field it shall be appended to the current list of output fields.

80922 Each field containing the results from an expansion shall be processed in order, intermixed with  
80923 fields not containing the results of expansions, processed as described above, as if by using the  
80924 following algorithm, examining bytes in the input field, from beginning to end:

- 80925 • Begin with an empty candidate field and the input as specified above.
- 80926 • When instructed to start the next iteration of the loop, this is the start of the loop. While the  
80927 input (as modified by earlier iterations of this loop) is not empty:
  - 80928 — Consider the leading remaining byte or byte sequence of the input. No such byte  
80929 sequence shall contain data such that some bytes in the sequence resulted from an  
80930 expansion, and others did not, nor which contains bytes resulting from the results of  
80931 more than one expansion. If the byte or sequence of bytes is:
    - 80932 1. A byte (or sequence of bytes) in the input which did not result from an  
80933 expansion:

80934 Append this byte (or sequence) to the candidate, and remove it from the  
80935 input. Start the next iteration of the loop.
    - 80936 2. A byte sequence in the input which resulted from an expansion and which  
80937 does not form a character in *IFS*:

80938 Append the first byte of the sequence to the candidate, and remove that byte  
80939 from the input. Start the next iteration of the loop.

80940 3. A byte sequence in the input which resulted from an expansion and which  
80941 forms an *IFS* white space character:

80942 Remove that byte sequence from the input, consider the new leading input  
80943 byte sequence, and repeat this step.

80944 4. A byte sequence in the input which resulted from an expansion and which  
80945 forms an *IFS* character that is not *IFS* white space:

80946 Remove that byte sequence from the input, but note it was observed.

80947 At this point, if the candidate is not empty, or if a sequence of bytes representing an  
80948 *IFS* character that is not *IFS* white space was seen at step 4, then a field is said to have  
80949 been delimited, and the candidate shall become an output field.

80950 — Empty (clear) the candidate, and start the next iteration of the loop.

80951 • Once the input is empty, the candidate shall become an output field if and only if it is not  
80952 empty.

80953 The ordered list of output fields so produced, which might be empty, shall replace the list of  
80954 input fields.

## 80955 2.6.6 Pathname Expansion

80956 After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be  
80957 expanded using the algorithm described in [Section 2.14](#), qualified by the rules in [Section 2.14.3](#).

## 80958 2.6.7 Quote Removal

80959 The quote character sequence `<dollar-sign> single-quote` and the single-character quote  
80960 characters (`<backslash>`, `single-quote`, and `double-quote`) that were present in the original word  
80961 shall be removed unless they have themselves been quoted. Note that the single-quote character  
80962 that terminates a `<dollar-sign> single-quote` sequence is itself a single-character quote character.

80963 **Note:** After quote removal the shell still remembers which characters were quoted. This is necessary  
80964 for purposes such as matching patterns in a `case` conditional construct (see [Section 2.9.4.3](#) and  
80965 [Section 2.14](#)).

## 80966 2.7 Redirection

80967 Redirection is used to open and close files for the current shell execution environment (see  
80968 [Section 2.13](#)) or for any command. Redirection operators can be used with numbers representing  
80969 file descriptors (see [XBD Section 3.141](#)) as described below.

80970 The overall format used for redirection is:

80971 `[n] redir-op word`

80972 The number *n* is an optional one or more digit decimal number designating the file descriptor  
80973 number; the application shall ensure it is delimited from any preceding text and immediately  
80974 precedes the redirection operator *redir-op* (with no intervening `<blank>` characters allowed). If *n*  
80975 is quoted, the number shall not be recognized as part of the redirection expression. For example:

80976 `echo \2>a`

80977 writes the character 2 into file `a`. If any part of *redir-op* is quoted, no redirection expression is

80978 recognized. For example:

80979 `echo 2>a`

80980 writes the characters `2>a` to standard output. The optional number, redirection operator, and  
80981 *word* shall not appear in the arguments provided to the command to be executed (if any).

80982 The shell may support an additional format used for redirection:

80983 `{location}redir-op word`

80984 where *location* is non-empty and indicates a location where an integer value can be stored, such  
80985 as the name of a shell variable. If this format is supported its behavior is implementation-  
80986 defined.

80987 The largest file descriptor number supported in shell redirections is implementation-defined;  
80988 however, all implementations shall support at least 0 to 9, inclusive, for use by the application.

80989 If the redirection operator is "`<<`" or "`<<-`", the word that follows the redirection operator shall  
80990 be subjected to quote removal; it is unspecified whether any of the other expansions occur. For  
80991 the other redirection operators, the word that follows the redirection operator shall be subjected  
80992 to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and  
80993 quote removal. Pathname expansion shall not be performed on the word by a non-interactive  
80994 shell; an interactive shell may perform it, but if the expansion would result in more than one  
80995 word it is unspecified whether the redirection proceeds without pathname expansion being  
80996 performed or the redirection fails.

80997 **Note:** A future version of this standard may require that the redirection fails in this case.

80998 If more than one redirection operator is specified with a command, the order of evaluation is  
80999 from beginning to end.

81000 A failure to open or create a file shall cause a redirection to fail.

## 81001 2.7.1 Redirecting Input

81002 Input redirection shall cause the file whose name results from the expansion of *word* to be  
81003 opened for reading on the designated file descriptor, or standard input if the file descriptor is  
81004 not specified.

81005 The general format for redirecting input is:

81006 `[n]<word`

81007 where the optional *n* represents the file descriptor number. If the number is omitted, the  
81008 redirection shall refer to standard input (file descriptor 0).

## 81009 2.7.2 Redirecting Output

81010 The two general formats for redirecting output are:

81011 `[n]>word`

81012 `[n]>|word`

81013 where the optional *n* represents the file descriptor number. If the number is omitted, the  
81014 redirection shall refer to standard output (file descriptor 1).

81015 Output redirection using the '`>`' format shall fail if the *noclobber* option is set (see the  
81016 description of `set -C`) and the file named by the expansion of *word* exists and is either a regular  
81017 file or a symbolic link that resolves to a regular file; it may also fail if the file is a symbolic link

81018 that does not resolve to an existing file. The check for existence, file creation, and open  
 81019 operations shall be performed atomically as is done by the *open()* function as defined in System  
 81020 Interfaces volume of POSIX.1-2024 when the *O\_CREAT* and *O\_EXCL* flags are set, except that if  
 81021 the file exists and is a symbolic link, the open operation need not fail with [EEXIST] unless the  
 81022 symbolic link resolves to an existing regular file. Performing these operations atomically  
 81023 ensures that the creation of lock files and unique (often temporary) files is reliable, with  
 81024 important caveats detailed in Section C.2.7.2 (on page 3891). The check for the type of the file  
 81025 need not be performed atomically with the check for existence, file creation, and open  
 81026 operations. If not, there is a potential race condition that may result in a misleading shell  
 81027 diagnostic message when redirection fails. See XRAT Section C.2.7.2 (on page 3891) for more  
 81028 details.

81029 In all other cases (*noclobber* not set, redirection using '>' does not fail for the reasons stated  
 81030 above, or redirection using the ">|" format), output redirection shall cause the file whose name  
 81031 results from the expansion of *word* to be opened for output on the designated file descriptor, or  
 81032 standard output if none is specified. If the file does not exist, it shall be created as an empty file;  
 81033 otherwise, it shall be opened as if the *open()* function was called with the *O\_TRUNC* flag set.

### 81034 2.7.3 Appending Redirected Output

81035 Appended output redirection shall cause the file whose name results from the expansion of  
 81036 *word* to be opened for output on the designated file descriptor. The file shall be opened as if the  
 81037 *open()* function as defined in the System Interfaces volume of POSIX.1-2024 was called with the  
 81038 *O\_APPEND* flag set. If the file does not exist, it shall be created.

81039 The general format for appending redirected output is as follows:

```
81040 [n]>>word
```

81041 where the optional *n* represents the file descriptor number. If the number is omitted, the  
 81042 redirection refers to standard output (file descriptor 1).

### 81043 2.7.4 Here-Document

81044 The redirection operators "<<" and "<<-" both allow redirection of subsequent lines read by  
 81045 the shell to the input of a command. The redirected lines are known as a "here-document".

81046 The here-document shall be treated as a single word that begins after the next **NEWLINE** token  
 81047 and continues until there is a line containing only the delimiter and a <newline>, with no  
 81048 <blank> characters in between. Then the next here-document starts, if there is one. For the  
 81049 purposes of locating this terminating line, the end of a *command\_string* operand (see *sh*) shall be  
 81050 treated as a <newline> character, and the end of the *commands* string in  $\$(commands)$  and  
 81051  $\`commands\`$  may be treated as a <newline>. If the end of input is reached without finding the  
 81052 terminating line, the shell should, but need not, treat this as a redirection error. The format is as  
 81053 follows:

```
81054 [n]<<word  

  81055     here-document  

  81056 delimiter
```

81057 where the optional *n* represents the file descriptor number. If the number is omitted, the here-  
 81058 document refers to standard input (file descriptor 0). It is unspecified whether the file descriptor  
 81059 is opened as a regular file or some other type of file. Portable applications cannot rely on the file  
 81060 descriptor being seekable (see XSH *lseek()*).

81061 If any part of *word* is quoted, not counting double-quotes outside a command substitution if the  
 81062 here-document is inside one, the delimiter shall be formed by performing quote removal on  
 81063 *word*, and the here-document lines shall not be expanded. Otherwise:

- 81064 • The delimiter shall be the *word* itself.
- 81065 • The removal of <backslash><newline> for line continuation (see [Section 2.2.1](#)) shall be  
 81066 performed during the search for the trailing delimiter. (As a consequence, the trailing  
 81067 delimiter is not recognized immediately after a <newline> that was removed by line  
 81068 continuation.) It is unspecified whether the line containing the trailing delimiter is itself  
 81069 subject to this line continuation.
- 81070 • All lines of the here-document shall be expanded, when the redirection operator is  
 81071 evaluated but after the trailing delimiter for the here-document has been located, for  
 81072 parameter expansion, command substitution, and arithmetic expansion. If the redirection  
 81073 operator is never evaluated (because the command it is part of is not executed), the here-  
 81074 document shall be read without performing any expansions.
- 81075 • Any <backslash> characters in the input shall behave as the <backslash> inside double-  
 81076 quotes (see [Section 2.2.3](#)). However, the double-quote character ( ' " ' ) shall not be treated  
 81077 specially within a here-document, except when the double-quote appears within "\$ ( ) ",  
 81078 "` ` ", or "\${ } ".

81079 If the redirection operator is "<<-", all leading <tab> characters shall be stripped from input  
 81080 lines after <backslash><newline> line continuation (when it applies) has been performed, and  
 81081 from the line containing the trailing delimiter. Stripping of leading <tab> characters shall occur  
 81082 as the here-document is read from the shell input (and consequently does not affect any <tab>  
 81083 characters that result from expansions).

81084 If more than one "<<" or "<<-" operator is specified on a line, the here-document associated  
 81085 with the first operator shall be supplied first by the application and shall be read first by the  
 81086 shell.

81087 When a here-document is read from a terminal device and the shell is interactive, it shall write  
 81088 the contents of the variable *PS2*, processed as described in [Section 2.5.3](#), to standard error before  
 81089 reading each line of input until the delimiter has been recognized.

## 81090 Examples

81091 An example of a here-document follows:

```
81092 cat <<eof1; cat <<eof2
81093 Hi,
81094 eof1
81095 Helene.
81096 eof2
```



### 81097 **2.7.5 Duplicating an Input File Descriptor**

81098 The redirection operator:

81099 `[n] <&word`

81100 shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one  
81101 or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be  
81102 made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent an  
81103 already open file descriptor, a redirection error shall result (see [Section 2.8.1](#)); if the file  
81104 descriptor denoted by *word* represents an open file descriptor that is not open for input, a  
81105 redirection error may result. If *word* evaluates to '-', file descriptor *n*, or standard input if *n* is  
81106 not specified, shall be closed. Attempts to close a file descriptor that is not open shall not  
81107 constitute an error. If *word* evaluates to something else, the behavior is unspecified.

### 81108 **2.7.6 Duplicating an Output File Descriptor**

81109 The redirection operator:

81110 `[n] >&word`

81111 shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to  
81112 one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall  
81113 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent  
81114 an already open file descriptor, a redirection error shall result (see [Section 2.8.1](#)); if the file  
81115 descriptor denoted by *word* represents an open file descriptor that is not open for output, a  
81116 redirection error may result. If *word* evaluates to '-', file descriptor *n*, or standard output if *n* is  
81117 not specified, is closed. Attempts to close a file descriptor that is not open shall not constitute an  
81118 error. If *word* evaluates to something else, the behavior is unspecified.

### 81119 **2.7.7 Open File Descriptors for Reading and Writing**

81120 The redirection operator:

81121 `[n] <>word`

81122 shall cause the file whose name is the expansion of *word* to be opened for both reading and  
81123 writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does  
81124 not exist, it shall be created.

## 81125 **2.8 Exit Status and Errors**

### 81126 **2.8.1 Consequences of Shell Errors**

81127 Certain errors shall cause the shell to write a diagnostic message to standard error and exit as  
81128 shown in the following table:

	<b>Error</b>	<b>Non-Interactive Shell</b>	<b>Interactive Shell</b>	<b>Shell Diagnostic Message Required</b>
81129	Shell language syntax error	shall exit	shall not exit	yes
81130				
81131	Special built-in utility error	shall exit <sup>1</sup>	shall not exit	no <sup>2</sup>
81132	Other utility (not a special built-in) error	shall not exit	shall not exit	no <sup>3</sup>
81133				
81134	Redirection error with special built-in utilities	shall exit	shall not exit	yes
81135				
81136	Redirection error with compound commands	shall not exit	shall not exit	yes
81137				
81138	Redirection error with function execution	shall not exit	shall not exit	yes
81139				
81140	Redirection error with other utilities (not special built-ins)	shall not exit	shall not exit	yes
81141				
81142	Variable assignment error	shall exit	shall not exit	yes
81143	Expansion error	shall exit	shall not exit	yes
81144	Command not found	may exit	shall not exit	yes
81145	Unrecoverable read error when reading commands	shall exit <sup>4</sup>	shall exit <sup>4</sup>	yes
81146				
81147				

81148 Notes:

- 81149 1. The shell shall exit only if the special built-in utility is executed directly. If it is executed  
81150 via the *command* utility, the shell shall not exit.
- 81151 2. Although special built-ins are part of the shell, a diagnostic message written by a special  
81152 built-in is not considered to be a shell diagnostic message, and can be redirected like any  
81153 other utility.
- 81154 3. The shell is not required to write a diagnostic message, but the utility itself shall write a  
81155 diagnostic message if required to do so.
- 81156 4. If an unrecoverable read error occurs when reading commands, other than from the *file*  
81157 operand of the *dot* special built-in, the shell shall execute no further commands (including  
81158 any already successfully read but not yet executed) other than any specified in a  
81159 previously defined EXIT *trap* action. An unrecoverable read error while reading from the  
81160 *file* operand of the *dot* special built-in shall be treated as a special built-in utility error.

81161 An expansion error is one that occurs when the shell expansions defined in [Section 2.6](#) are  
81162 carried out (for example, "\$ {x!y}", because '!' is not a valid operator); an implementation  
81163 may treat these as syntax errors if it is able to detect them during tokenization, rather than  
81164 during expansion.

81165 If any of the errors shown as "shall exit" or "may exit" occur in a subshell environment, the shell  
81166 shall (respectively, may) exit from the subshell environment with a non-zero status and continue  
81167 in the environment from which that subshell environment was invoked.

81168 In all of the cases shown in the table where an interactive shell is required not to exit and a non-  
81169 interactive shell is required to exit, an interactive shell shall not perform any further processing  
81170 of the command in which the error occurred.

## 81171 2.8.2 Exit Status for Commands

81172 Each command has an exit status that can influence the behavior of other shell commands. The  
81173 exit status of commands that are not utilities is documented in this section. The exit status of the  
81174 standard utilities is documented in their respective sections.

81175 The exit status of a command shall be determined as follows:

- 81176 • If the command is not found, the exit status shall be 127.
- 81177 • Otherwise, if the command name is found, but it is not an executable utility, the exit status  
81178 shall be 126.
- 81179 • Otherwise, if the command terminated due to the receipt of a signal, the shell shall assign  
81180 it an exit status greater than 128. The exit status shall identify, in an implementation-  
81181 defined manner, which signal terminated the command. Note that shell implementations  
81182 are permitted to assign an exit status greater than 255 if a command terminates due to a  
81183 signal.
- 81184 • Otherwise, the exit status shall be the value obtained by the equivalent of the  
81185 WEXITSTATUS macro applied to the status obtained by the *wait()* function (as defined in  
81186 the System Interfaces volume of POSIX.1-2024). Note that for C programs, this value is  
81187 equal to the result of performing a modulo 256 operation on the value passed to *\_Exit()*,  
81188 *\_exit()*, or *exit()* or returned from *main()*.

## 81189 2.9 Shell Commands

81190 This section describes the basic structure of shell commands. The following command  
81191 descriptions each describe a format of the command that is only used to aid the reader in  
81192 recognizing the command type, and does not formally represent the syntax. In particular, the  
81193 representations include spacing between tokens in some places where <blank>s would not be  
81194 necessary (when one of the tokens is an operator). Each description discusses the semantics of  
81195 the command; for a formal definition of the command language, consult [Section 2.10](#).

81196 A *command* is one of the following:

- 81197 • Simple command (see [Section 2.9.1](#))
- 81198 • Pipeline (see [Section 2.9.2](#))
- 81199 • List compound-list (see [Section 2.9.3](#))
- 81200 • Compound command (see [Section 2.9.4](#))
- 81201 • Function definition (see [Section 2.9.5](#))

81202 Unless otherwise stated, the exit status of a command shall be that of the last simple command  
81203 executed by the command. There shall be no limit on the size of any shell command other than  
81204 that imposed by the underlying system (memory constraints, {ARG\_MAX}, and so on).

81205 **2.9.1 Simple Commands**

81206 A “simple command” is a sequence of optional variable assignments and redirections, in any  
81207 sequence, optionally followed by words and redirections.

81208 *2.9.1.1 Order of Processing*

81209 When a given simple command is required to be executed (that is, when any conditional  
81210 construct such as an AND-OR list or a **case** statement has not bypassed the simple command),  
81211 the following expansions, assignments, and redirections shall all be performed from the  
81212 beginning of the command text to the end:

- 81213 1. The words that are recognized as variable assignments or redirections according to  
81214 [Section 2.10.2](#) are saved for processing in steps 3 and 4.
- 81215 2. The first word (if any) that is not a variable assignment or redirection shall be expanded.  
81216 If any fields remain following its expansion, the first field shall be considered the  
81217 command name. If no fields remain, the next word (if any) shall be expanded, and so on,  
81218 until a command name is found or no words remain. If there is a command name and it is  
81219 recognized as a declaration utility, then any remaining words after the word that  
81220 expanded to produce the command name, that would be recognized as a variable  
81221 assignment in isolation, shall be expanded as a variable assignment (tilde expansion after  
81222 the first <equals-sign> and after any unquoted <colon>, parameter expansion, command  
81223 substitution, arithmetic expansion, and quote removal, but no field splitting or pathname  
81224 expansion); while remaining words that would not be a variable assignment in isolation  
81225 shall be subject to regular expansion (tilde expansion for only a leading <tilde>,  
81226 parameter expansion, command substitution, arithmetic expansion, field splitting,  
81227 pathname expansion, and quote removal). For all other command names, words after the  
81228 word that produced the command name shall be subject only to regular expansion. All  
81229 fields resulting from the expansion of the word that produced the command name and  
81230 the subsequent words, except for the field containing the command name, shall be the  
81231 arguments for the command.
- 81232 3. Redirections shall be performed as described in [Section 2.7](#).
- 81233 4. Each variable assignment shall be expanded for tilde expansion, parameter expansion,  
81234 command substitution, arithmetic expansion, and quote removal prior to assigning the  
81235 value.

81236 In the preceding list, the order of steps 3 and 4 may be reversed if no command name results  
81237 from step 2 or if the command name matches the name of a special built-in utility; see [Section](#)  
81238 [2.15](#).

81239 When determining whether a command name is a declaration utility, an implementation may  
81240 use only lexical analysis. It is unspecified whether assignment context will be used if the  
81241 command name would only become recognized as a declaration utility after word expansions.

81242 *2.9.1.2 Variable Assignments*

81243 Variable assignments shall be performed as follows:

- 81244 • If no command name results, variable assignments shall affect the current execution  
81245 environment.
- 81246 • If the command name is not a special built-in utility or function, the variable assignments  
81247 shall be exported for the execution environment of the command and shall not affect the  
81248 current execution environment except as a side-effect of the expansions performed in step

- 81249 4. In this case it is unspecified:
- 81250 — Whether or not the assignments are visible for subsequent expansions in step 4
- 81251 — Whether variable assignments made as side-effects of these expansions are visible for
- 81252 subsequent expansions in step 4, or in the current shell execution environment, or
- 81253 both
- 81254 • If the command name is a standard utility implemented as a function (see XBD [Section](#)
  - 81255 [4.25](#)), the effect of variable assignments shall be as if the utility was not implemented as a
  - 81256 function.
  - 81257 • If the command name is a special built-in utility, variable assignments shall affect the
  - 81258 current execution environment before the utility is executed and remain in effect when the
  - 81259 command completes; if an assigned variable is further modified by the utility, the
  - 81260 modifications made by the utility shall persist. Unless the *set -a* option is on (see *set*), it is
  - 81261 unspecified:
    - 81262 — Whether or not the variables gain the *export* attribute during the execution of the
    - 81263 special built-in utility
    - 81264 — Whether or not *export* attributes gained as a result of the variable assignments persist
    - 81265 after the completion of the special built-in utility
    - 81266 • If the command name is a function that is not a standard utility implemented as a function,
    - 81267 variable assignments shall affect the current execution environment during the execution
    - 81268 of the function. It is unspecified:
      - 81269 — Whether or not the variable assignments persist after the completion of the function
      - 81270 — Whether or not the variables gain the *export* attribute during the execution of the
      - 81271 function
      - 81272 — Whether or not *export* attributes gained as a result of the variable assignments persist
      - 81273 after the completion of the function (if variable assignments persist after the
      - 81274 completion of the function)
- 81275 If any of the variable assignments attempt to assign a value to a variable for which the *readonly*
- 81276 attribute is set in the current shell environment (regardless of whether the assignment is made in
- 81277 that environment), a variable assignment error shall occur. See [Section 2.8.1](#) for the consequences
- 81278 of these errors.

#### 81279 2.9.1.3 *Commands with no Command Name*

81280 If a simple command has no command name after word expansion (see [Section 2.9.1.1](#)), any

81281 redirections shall be performed in a subshell environment; it is unspecified whether this subshell

81282 environment is the same one as that used for a command substitution within the command. (To

81283 affect the current execution environment, see the *exec* special built-in.) If any of the redirections

81284 performed in the current shell execution environment fail, the command shall immediately fail

81285 with an exit status greater than zero, and the shell shall write an error message indicating the

81286 failure. See [Section 2.8.1](#) for the consequences of these failures on interactive and non-interactive

81287 shells.

81288 Additionally, if there is no command name but the command contains a command substitution,

81289 the command shall complete with the exit status of the command substitution whose exit status

81290 was the last to be obtained. Otherwise, the command shall complete with a zero exit status.

## 81291 2.9.1.4 Command Search and Execution

81292 If a simple command has a command name and an optional list of arguments after word  
81293 expansion (see [Section 2.9.1.1](#)), the following actions shall be performed:

81294 1. If the command name does not contain any <slash> characters, the first successful step in  
81295 the following sequence shall occur:

81296 a. If the command name matches the name of a special built-in utility, that special  
81297 built-in utility shall be invoked.

81298 b. If the command name matches the name of a utility listed in the following table,  
81299 the results are unspecified.

81300	<i>alloc</i>	<i>compcall</i>	<i>compvalues</i>	<i>history</i>	<i>print</i>
81301	<i>autoload</i>	<i>compctl</i>	<i>declare</i>	<i>hist</i>	<i>pushd</i>
81302	<i>bind</i>	<i>compdescribe</i>	<i>dirs</i>	<i>integer</i>	<i>readarray</i>
81303	<i>bindkey</i>	<i>compfiles</i>	<i>disable</i>	<i>let</i>	<i>repeat</i>
81304	<i>builtin</i>	<i>compgen</i>	<i>disown</i>	<i>local</i>	<i>savehistory</i>
81305	<i>bye</i>	<i>compgroups</i>	<i>dosh</i>	<i>login</i>	<i>source</i>
81306	<i>caller</i>	<i>complete</i>	<i>echoct</i>	<i>logout</i>	<i>shopt</i>
81307	<i>cap</i>	<i>compound</i>	<i>echoit</i>	<i>map</i>	<i>stop</i>
81308	<i>chdir</i>	<i>compquote</i>	<i>enum</i>	<i>mapfile</i>	<i>suspend</i>
81309	<i>clone</i>	<i>comptags</i>	<i>float</i>	<i>nameref</i>	<i>typeset</i>
81310	<i>comparuments</i>	<i>comptry</i>	<i>help</i>	<i>popd</i>	<i>whence</i>

81311 c. If the command name matches the name of a function known to this shell, the  
81312 function shall be invoked as described in [Section 2.9.5](#). If the implementation has  
81313 provided a standard utility in the form of a function, and that function definition  
81314 still exists (i.e. has not been removed using *unset -f* or replaced via another  
81315 function definition with the same name), it shall not be recognized at this point. It  
81316 shall be invoked in conjunction with the path search in step 1e.

81317 d. If the command name matches the name of an intrinsic utility (see [Section 1.7](#), on  
81318 page 2470), that utility shall be invoked.

81319 e. Otherwise, the command shall be searched for using the *PATH* environment  
81320 variable as described in [XBD Chapter 8](#):

81321 i. If the search is successful:

81322 a. If the system has implemented the utility as a built-in or as a shell  
81323 function, and the built-in or function is associated with the directory  
81324 that was most recently tested during the successful *PATH* search, that  
81325 built-in or function shall be invoked.

81326 b. Otherwise, the shell shall execute a non-built-in utility as described  
81327 in [Section 2.9.1.6](#).

81328 Once a utility has been searched for and found (either as a result of this  
81329 specific search or as part of an unspecified shell start-up activity), an  
81330 implementation may remember its location and need not search for the  
81331 utility again unless the *PATH* variable has been the subject of an assignment.  
81332 If the remembered location fails for a subsequent invocation, the shell shall  
81333 repeat the search to find the new location for the utility, if any.

81334 ii. If the search is unsuccessful, the command shall fail with an exit status of  
81335 127 and the shell shall write an error message.

- 81336 2. If the command name contains at least one <slash>, the shell shall execute a non-built-in  
81337 utility as described in [Section 2.9.1.6](#).

81338 2.9.1.5 *Standard File Descriptors*

81339 If the utility would be executed with file descriptor 0, 1, or 2 closed, implementations may  
81340 execute the utility with the file descriptor open to an unspecified file. If a standard utility or a  
81341 conforming application is executed with file descriptor 0 not open for reading or with file  
81342 descriptor 1 or 2 not open for writing, the environment in which the utility or application is  
81343 executed shall be deemed non-conforming, and consequently the utility or application might not  
81344 behave as described in this standard.

81345 2.9.1.6 *Non-built-in Utility Execution*

81346 When the shell executes a non-built-in utility, if the execution is not being made via the *exec*  
81347 special built-in utility, the shell shall execute the utility in a separate utility environment (see  
81348 [Section 2.13](#)).

81349 If the execution is being made via the *exec* special built-in utility, the shell shall not create a  
81350 separate utility environment for this execution; the new process image shall replace the current  
81351 shell execution environment. If the current shell environment is a subshell environment, the new  
81352 process image shall replace the subshell environment and the shell shall continue in the  
81353 environment from which that subshell environment was invoked.

81354 In either case, execution of the utility in the specified environment shall be performed as follows:

- 81355 1. If the command name does not contain any <slash> characters, the command name shall  
81356 be searched for using the *PATH* environment variable as described in [XBD Chapter 8](#) (on  
81357 page 167):

- 81358 a. If the search is successful, the shell shall execute the utility with actions equivalent  
81359 to calling the *execl()* function as defined in the System Interfaces volume of  
81360 POSIX.1-2024 with the *path* argument set to the pathname resulting from the  
81361 search, *arg0* set to the command name, and the remaining *execl()* arguments set to  
81362 the command arguments (if any) and the null terminator.

81363 If the *execl()* function fails due to an error equivalent to the [ENOEXEC] error  
81364 defined in the System Interfaces volume of POSIX.1-2024, the shell shall execute a  
81365 command equivalent to having a shell invoked with the pathname resulting from  
81366 the search as its first operand, with any remaining arguments passed to the new  
81367 shell, except that the value of "\$0" in the new shell may be set to the command  
81368 name. The shell may apply a heuristic check to determine if the file to be executed  
81369 could be a script and may bypass this command execution if it determines that the  
81370 file cannot be a script. In this case, it shall write an error message, and the  
81371 command shall fail with an exit status of 126.

81372 **Note:** A common heuristic for rejecting files that cannot be a script is locating a NUL  
81373 byte prior to a <newline> byte within a fixed-length prefix of the file. Since *sh* is  
81374 required to accept input files with unlimited line lengths, the heuristic check  
81375 cannot be based on line length.

81376 It is unspecified whether environment variables that were passed to the shell when  
81377 it was invoked, but were not used to initialize shell variables (see [Section 2.5.3](#))  
81378 because they had invalid names, are included in the environment passed to *execl()*  
81379 and (if *execl()* fails as described above) to the new shell.

81380 b. If the search is unsuccessful, the command shall fail with an exit status of 127 and  
81381 the shell shall write an error message.

81382 2. If the command name contains at least one <slash>:

81383 a. If the named utility exists, the shell shall execute the utility with actions equivalent  
81384 to calling the *execl()* function defined in the System Interfaces volume of  
81385 POSIX.1-2024 with the *path* and *arg0* arguments set to the command name, and the  
81386 remaining *execl()* arguments set to the command arguments (if any) and the null  
81387 terminator.

81388 If the *execl()* function fails due to an error equivalent to the [ENOEXEC] error, the  
81389 shell shall execute a command equivalent to having a shell invoked with the  
81390 command name as its first operand, with any remaining arguments passed to the  
81391 new shell. The shell may apply a heuristic check to determine if the file to be  
81392 executed could be a script and may bypass this command execution if it  
81393 determines that the file cannot be a script. In this case, it shall write an error  
81394 message, and the command shall fail with an exit status of 126.

81395 **Note:** A common heuristic for rejecting files that cannot be a script is locating a NUL  
81396 byte prior to a <newline> byte within a fixed-length prefix of the file. Since *sh* is  
81397 required to accept input files with unlimited line lengths, the heuristic check  
81398 cannot be based on line length.

81399 It is unspecified whether environment variables that were passed to the shell when  
81400 it was invoked, but were not used to initialize shell variables (see [Section 2.5.3](#))  
81401 because they had invalid names, are included in the environment passed to *execl()*  
81402 and (if *execl()* fails as described above) to the new shell.

81403 b. If the named utility does not exist, the command shall fail with an exit status of 127  
81404 and the shell shall write an error message.

## 81405 2.9.2 Pipelines

81406 A *pipeline* is a sequence of one or more commands separated by the control operator '|'. For  
81407 each command but the last, the shell shall connect the standard output of the command to the  
81408 standard input of the next command as if by creating a pipe and passing the write end of the  
81409 pipe as the standard output of the command and the read end of the pipe as the standard input  
81410 of the next command.

81411 The format for a pipeline is:

```
81412 [!] command1 [ | command2 ... ]
```

81413 If the pipeline begins with the reserved word ! and *command1* is a subshell command, the  
81414 application shall ensure that the ( operator at the beginning of *command1* is separated from the !  
81415 by one or more <blank> characters. The behavior of the reserved word ! immediately followed  
81416 by the ( operator is unspecified.

81417 The standard output of *command1* shall be connected to the standard input of *command2*. The  
81418 standard input, standard output, or both of a command shall be considered to be assigned by  
81419 the pipeline before any redirection specified by redirection operators that are part of the  
81420 command (see [Section 2.7](#)).

81421 If the pipeline is not in the background (see [Section 2.9.3.1](#) and [Section 2.11](#)), the shell shall wait  
81422 for the last command specified in the pipeline to complete, and may also wait for all commands  
81423 to complete.



81424 **Exit Status**

81425 The exit status of a pipeline shall depend on whether or not the *pipefail* option (see *set*) is enabled  
 81426 and whether or not the pipeline begins with the `!` reserved word, as described in the following  
 81427 table. The *pipefail* option determines which command in the pipeline the exit status is derived  
 81428 from; the `!` reserved word causes the exit status to be the logical NOT of the exit status of that  
 81429 command. The shell shall use the *pipefail* setting at the time it begins execution of the pipeline,  
 81430 not the setting at the time it sets the exit status of the pipeline. (For example, in `command1 |`  
 81431 `set -o pipefail` the exit status of `command1` has no effect on the exit status of the pipeline,  
 81432 even if the shell executes `set -o pipefail` in the current shell environment.)

<i>pipefail</i> Enabled	Begins with <code>!</code>	Exit Status
no	no	The exit status of the last (rightmost) command specified in the pipeline.
no	yes	Zero, if the last (rightmost) command in the pipeline returned a non-zero exit status; otherwise, 1.
yes	no	Zero, if all commands in the pipeline returned an exit status of 0; otherwise, the exit status of the last (rightmost) command specified in the pipeline that returned a non-zero exit status.
yes	yes	Zero, if any command in the pipeline returned a non-zero exit status; otherwise, 1.

81444 **2.9.3 Lists**

81445 An *AND-OR list* is a sequence of one or more pipelines separated by the operators `&&` and  
 81446 `||`.

81447 A *list* is a sequence of one or more AND-OR lists separated by the operators `;` and `&'`.

81448 The operators `&&` and `||` shall have equal precedence and shall be evaluated with left  
 81449 associativity. For example, both of the following commands write solely **bar** to standard output:

```
81450 false && echo foo || echo bar
81451 true || echo foo && echo bar
```

81452 A `;` separator or a `&'` or `<newline>` terminator shall cause the preceding AND-OR list to be  
 81453 executed sequentially; an `&'` separator or terminator shall cause asynchronous execution of the  
 81454 preceding AND-OR list.

81455 The term “compound-list” is derived from the grammar in [Section 2.10](#); it is equivalent to a  
 81456 sequence of *lists*, separated by `<newline>` characters, that can be preceded or followed by an  
 81457 arbitrary number of `<newline>` characters.

81458 **Examples**

81459 The following is an example that illustrates `<newline>` characters in compound-lists:

```
81460 while
81461     # a couple of <newline>s
81462     # a list
81463     date && who || ls; cat file
81464     # a couple of <newline>s
81465     # another list
```

```

81466         wc file > output & true
81467     do
81468         # 2 lists
81469         ls
81470         cat file
81471     done

```

### 81472 2.9.3.1 Asynchronous AND-OR Lists

81473 If an AND-OR list is terminated by the control operator `&` ('&'), the shell shall  
81474 execute the AND-OR list asynchronously in a subshell environment. This subshell shall execute  
81475 in the background; that is, the shell shall not wait for the subshell to terminate before executing  
81476 the next command (if any); if there are no further commands to execute, the shell shall not wait  
81477 for the subshell to terminate before exiting.

81478 If job control is enabled (see *set*, `-m`), the AND-OR list shall become a job-control background  
81479 job and a job number shall be assigned to it. If job control is disabled, the AND-OR list may  
81480 become a non-job-control background job, in which case a job number shall be assigned to it; if  
81481 no job number is assigned it shall become a background command but not a background job.

81482 A job-control background job can be controlled as described in [Section 2.11](#).

81483 The process ID associated with the asynchronous AND-OR list shall become known in the  
81484 current shell execution environment; see [Section 2.13](#). This process ID shall remain known until  
81485 any one of the following occurs (and, unless otherwise specified, may continue to remain known  
81486 after it occurs).

- 81487 • The process terminates and the application waits for the process ID or the corresponding  
81488 job ID (see *wait*).
- 81489 • If the asynchronous AND-OR list did not become a background job: another asynchronous  
81490 AND-OR list is invoked before "\$!" (corresponding to the previous asynchronous AND-  
81491 OR list) is expanded in the current shell execution environment.
- 81492 • If the asynchronous AND-OR list became a background job: the *jobs* utility reports the  
81493 termination status of that job.
- 81494 • If the shell is interactive and the asynchronous AND-OR list became a background job: a  
81495 message indicating completion of the corresponding job is written to standard error. If *set*  
81496 `-b` is enabled, it is unspecified whether the process ID is removed from the list of known  
81497 process IDs when the message is written or immediately prior to when the shell writes the  
81498 next prompt for input.

81499 The implementation need not retain more than the {CHILD\_MAX} most recent entries in its list  
81500 of known process IDs in the current shell execution environment.

81501 If, and only if, job control is disabled, the standard input for the subshell in which an  
81502 asynchronous AND-OR list is executed shall initially be assigned to an open file description that  
81503 behaves as if `/dev/null` had been opened for reading only. This initial assignment shall be  
81504 overridden by any explicit redirection of standard input within the AND-OR list.

81505 If the shell is interactive and the asynchronous AND-OR list became a background job, the job  
81506 number and the process ID associated with the job shall be written to standard error using the  
81507 format:

```
81508 "[%d] %d\n", <job-number>, <process-id>
```

81509 If the shell is interactive and the asynchronous AND-OR list did not become a background job,  
81510 the process ID associated with the asynchronous AND-OR list shall be written to standard error  
81511 in an unspecified format.

81512 **Exit Status**

81513 The exit status of an asynchronous AND-OR list shall be zero.

81514 The exit status of the subshell in which the AND-OR list is asynchronously executed can be  
81515 obtained using the *wait* utility.

81516 2.9.3.2 *Sequential AND-OR Lists*

81517 AND-OR lists that are separated by a <semicolon> ( ';' ) shall be executed sequentially. The  
81518 format for executing AND-OR lists sequentially shall be:

81519 *aolist1* [ ; *aolist2* ] . . .

81520 Each AND-OR list shall be expanded and executed in the order specified.

81521 If job control is enabled, the AND-OR lists shall form all or part of a foreground job that can be  
81522 controlled as described in [Section 2.11](#).

81523 **Exit Status**

81524 The exit status of a sequential AND-OR list shall be the exit status of the last pipeline in the  
81525 AND-OR list that is executed.

81526 2.9.3.3 *AND Lists*

81527 The control operator "&&" denotes an AND list. The format shall be:

81528 *command1* [ && *command2* ] . . .

81529 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,  
81530 until a command has a non-zero exit status or there are no more commands left to execute. The  
81531 commands are expanded only if they are executed.

81532 **Exit Status**

81533 The exit status of an AND list shall be the exit status of the last command that is executed in the  
81534 list.

81535 2.9.3.4 *OR Lists*

81536 The control operator " | | " denotes an OR List. The format shall be:

81537 *command1* [ | | *command2* ] . . .

81538 First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and  
81539 so on, until a command has a zero exit status or there are no more commands left to execute.

81540 **Exit Status**

81541 The exit status of an OR list shall be the exit status of the last command that is executed in the  
81542 list.

81543 **2.9.4 Compound Commands**

81544 The shell has several programming constructs that are “compound commands”, which provide  
81545 control flow for commands. Each of these compound commands has a reserved word or control  
81546 operator at the beginning, and a corresponding terminator reserved word or operator at the end.  
81547 In addition, each can be followed by redirections on the same line as the terminator. Each  
81548 redirection shall apply to all the commands within the compound command that do not  
81549 explicitly override that redirection.

81550 In the descriptions below, the exit status of some compound commands is stated in terms of the  
81551 exit status of a *compound-list*. The exit status of a *compound-list* shall be the value that the special  
81552 parameter ' ? ' (see [Section 2.5.2](#)) would have immediately after execution of the *compound-list*.

81553 **2.9.4.1 Grouping Commands**

81554 The format for grouping commands is as follows:

81555 ( *compound-list* ) Execute *compound-list* in a subshell environment; see [Section 2.13](#).  
81556 Variable assignments and built-in commands that affect the environment  
81557 shall not remain in effect after the list finishes.

81558 If a character sequence beginning with " ( " would be parsed by the shell  
81559 as an arithmetic expansion if preceded by a '\$', shells which implement  
81560 an extension whereby " ( *expression* ) " is evaluated as an arithmetic  
81561 expression may treat the " ( " as introducing as an arithmetic evaluation  
81562 instead of a grouping command. A conforming application shall ensure  
81563 that it separates the two leading ' ( ' characters with white space to  
81564 prevent the shell from performing an arithmetic evaluation.

81565 { *compound-list* ; } Execute *compound-list* in the current process environment. The semicolon  
81566 shown here is an example of a control operator delimiting the } reserved  
81567 word. Other delimiters are possible, as shown in [Section 2.10](#); a  
81568 <newline> is frequently used.

81569 **Exit Status**

81570 The exit status of a grouping command shall be the exit status of *compound-list*.

81571 **2.9.4.2 The for Loop**

81572 The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for**  
81573 loop requires that the reserved words **do** and **done** be used to delimit the sequence of  
81574 commands.

81575 The format for the **for** loop is as follows:

```
81576 for name [ in [word ... ] ]
81577 do
81578     compound-list
81579 done
```

81580 First, the list of words following **in** shall be expanded to generate a list of items. Then, the  
 81581 variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no  
 81582 items result from the expansion, the *compound-list* shall not be executed. Omitting:

81583 `in word...`

81584 shall be equivalent to:

81585 `in "$@"`

## 81586 **Exit Status**

81587 If there is at least one item in the list of items, the exit status of a **for** command shall be the exit  
 81588 status of the last *compound-list* executed. If there are no items, the exit status shall be zero.

### 81589 2.9.4.3 *Case Conditional Construct*

81590 The conditional construct **case** shall execute the *compound-list* corresponding to the first *pattern*  
 81591 (see [Section 2.14](#)), if any are present, that is matched by the string resulting from the tilde  
 81592 expansion, parameter expansion, command substitution, arithmetic expansion, and quote  
 81593 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to  
 81594 be matched. Multiple patterns with the same *compound-list* shall be delimited by the '|'   
 81595 symbol. The control operator ')' terminates a list of patterns corresponding to a given action.  
 81596 The terminated pattern list and the following *compound-list* is called a **case** statement *clause*.  
 81597 Each **case** statement clause, with the possible exception of the last, shall be terminated with  
 81598 either ";" or "&". The **case** construct terminates with the reserved word **esac** (**case** reversed).

81599 The format for the **case** construct is as follows:

```
81600 case word in
81601     [(pattern | pattern) ... ] compound-list terminator ...
81602     [(pattern | pattern) ... ] compound-list
81603 esac
```

81604 Where *terminator* is either ";" or "&" and is optional for the last *compound-list*.

81605 In order from the beginning to the end of the **case** statement, each *pattern* that labels a *compound-*  
 81606 *list* shall be subjected to tilde expansion, parameter expansion, command substitution, and  
 81607 arithmetic expansion, and the result of these expansions shall be compared against the  
 81608 expansion of *word*, according to the rules described in [Section 2.14](#) (which also describes the  
 81609 effect of quoting parts of the pattern). After the first match, no more patterns in the **case**  
 81610 statement shall be expanded, and the *compound-list* of the matching clause shall be executed. If  
 81611 the **case** statement clause is terminated by ";;", no further clauses shall be examined. If the  
 81612 **case** statement clause is terminated by "&", then the *compound-list* (if any) of each subsequent  
 81613 clause shall be executed, in order, until either a clause terminated by ";;" is reached and its  
 81614 *compound-list* (if any) executed or there are no further clauses in the **case** statement. The order of  
 81615 expansion and comparison of multiple *patterns* that label a *compound-list* statement is  
 81616 unspecified.

81617 **Exit Status**

81618 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be  
81619 the exit status of the *compound-list* of the last clause to be executed.

81620 2.9.4.4 *The if Conditional Construct*

81621 The **if** command shall execute a *compound-list* and use its exit status to determine whether to  
81622 execute another *compound-list*.

81623 The format for the **if** construct is as follows:

```
81624 if compound-list
81625 then
81626     compound-list
81627 [elif compound-list
81628 then
81629     compound-list] ...
81630 [else
81631     compound-list]
81632 fi
```

81633 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be  
81634 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,  
81635 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command  
81636 shall complete. Otherwise, the **else** *compound-list* shall be executed.

81637 **Exit Status**

81638 The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that  
81639 was executed, or zero, if none was executed.

81640 **Note:** Although the exit status of the **if** or **elif** *compound-list* is ignored when determining the exit  
81641 status of the **if** command, it is available through the special parameter '?' (see [Section 2.5.2](#))  
81642 during execution of the next **then**, **elif**, or **else** *compound-list* (if any is executed) in the normal  
81643 way.

81644 2.9.4.5 *The while Loop*

81645 The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
81646 a zero exit status.

81647 The format of the **while** loop is as follows:

```
81648 while compound-list-1
81649 do
81650     compound-list-2
81651 done
```

81652 The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command  
81653 shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

81654 **Exit Status**

81655 The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or  
81656 zero if none was executed.

81657 **Note:** Since the exit status of *compound-list-1* is ignored when determining the exit status of the **while**  
81658 command, it is not possible to obtain the status of the command that caused the loop to exit,  
81659 other than via the special parameter '?' (see [Section 2.5.2](#)) during execution of *compound-list-1*,  
81660 for example:

```
81661 while some_command; st=$?; false; do ...
```

81662 The exit status of *compound-list-1* is available through the special parameter '?' during  
81663 execution of *compound-list-2*, but is known to be zero at that point anyway.

81664 2.9.4.6 *The until Loop*

81665 The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has  
81666 a non-zero exit status.

81667 The format of the **until** loop is as follows:

```
81668 until compound-list-1
81669 do
81670     compound-list-2
81671 done
```

81672 The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command  
81673 completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

81674 **Exit Status**

81675 The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or  
81676 zero if none was executed.

81677 **Note:** Although the exit status of *compound-list-1* is ignored when determining the exit status of the  
81678 **until** command, it is available through the special parameter '?' (see [Section 2.5.2](#)) during  
81679 execution of *compound-list-2* in the normal way.

81680 **2.9.5 Function Definition Command**

81681 A function is a user-defined name that is used as a simple command to call a compound  
81682 command with new positional parameters. A function is defined with a "function definition  
81683 command".

81684 The format of a function definition command is as follows:

```
81685 fname ( ) compound-command [io-redirect ...]
```

81686 The function is named *fname*; the application shall ensure that it is a name (see [XBD Section 3.216](#))  
81687 and that it is not the name of a special built-in utility. An implementation may allow other  
81688 characters in a function name as an extension. The implementation shall maintain separate name  
81689 spaces for functions and variables.

81690 The argument *compound-command* represents a compound command, as described in [Section 2.9.4](#).  
81691

81692 When the function is declared, none of the expansions in [Section 2.6](#) shall be performed on the  
81693 text in *compound-command* or *io-redirect*; all expansions shall be performed as normal each time

81694 the function is called. Similarly, the optional *io-redirect* redirections and any variable assignments  
 81695 within *compound-command* shall be performed during the execution of the function itself, not the  
 81696 function definition. See [Section 2.8.1](#) for the consequences of failures of these operations on  
 81697 interactive and non-interactive shells.

81698 When a function is executed, it shall have the syntax-error properties described for special built-  
 81699 in utilities in the first item in the enumerated list at the beginning of [Section 2.15](#).

81700 The *compound-command* shall be executed whenever the function name is specified as the name  
 81701 of a simple command (see [Section 2.9.1.4](#)). The operands to the command temporarily shall  
 81702 become the positional parameters during the execution of the *compound-command*; the special  
 81703 parameter '#' also shall be changed to reflect the number of operands. The special parameter 0  
 81704 shall be unchanged. When the function completes, the values of the positional parameters and  
 81705 the special parameter '#' shall be restored to the values they had before the function was  
 81706 executed. If the special built-in *return* (see *return*) is executed in the *compound-command*, the  
 81707 function completes and execution shall resume with the next command after the function call.

## 81708 **Exit Status**

81709 The exit status of a function definition shall be zero if the function was declared successfully;  
 81710 otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit  
 81711 status of the last command executed by the function.

## 81712 **2.10 Shell Grammar**

81713 The following grammar defines the Shell Command Language. This formal syntax shall take  
 81714 precedence over the preceding text syntax description.

### 81715 **2.10.1 Shell Grammar Lexical Conventions**

81716 The input language to the shell shall be first recognized at the character level. The resulting  
 81717 tokens shall be classified by their immediate context according to the following rules (applied in  
 81718 order). These rules shall be used to determine what a "token" is that is subject to parsing at the  
 81719 token level. The rules for token recognition in [Section 2.3](#) shall apply.

- 81720 1. If the token is an operator, the token identifier for that operator shall result.
- 81721 2. If the string consists solely of digits and the delimiter character is one of '<' or '>', the  
 81722 token identifier **IO\_NUMBER** shall result.
- 81723 3. If the string contains at least three characters, begins with a <left-curly-bracket> ('{')  
 81724 and ends with a <right-curly-bracket> ('}'), and the delimiter character is one of '<' or  
 81725 '>', the token identifier **IO\_LOCATION** may result; if the result is not **IO\_LOCATION**,  
 81726 the token identifier **TOKEN** shall result.
- 81727 4. Otherwise, the token identifier **TOKEN** shall result.

81728 Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields  
 81729 **WORD**, a **NAME**, an **ASSIGNMENT\_WORD**, or one of the reserved words below, dependent  
 81730 upon the context. Some of the productions in the grammar below are annotated with a rule  
 81731 number from the following list. When a **TOKEN** is seen where one of those annotated  
 81732 productions could be used to reduce the symbol, the applicable rule shall be applied to convert  
 81733 the token identifier type of the **TOKEN** to:



- 81734 • The token identifier of the recognized reserved word, for rule 1
  - 81735 • A token identifier acceptable at that point in the grammar, for all other rules
- 81736 The reduction shall then proceed based upon the token identifier type yielded by the rule  
 81737 applied. When more than one rule applies, the highest numbered rule shall apply (which in turn  
 81738 may refer to another rule). (Note that except in rule 7, the presence of an '=' in the token has no  
 81739 effect.)
- 81740 The **WORD** tokens shall have the word expansion rules applied to them immediately before the  
 81741 associated command is executed, not at the time the command is parsed.

## 81742 2.10.2 Shell Grammar Rules

- 81743 1. [Command Name]
- 81744 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
 81745 shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any  
 81746 state where only a reserved word could be the next correct token, proceed as above.
- 81747 **Note:** Because at this point quoting characters (<backslash>, single-quote, <quotation-mark>,  
 81748 and the <dollar-sign> single-quote sequence) are retained in the token, quoted strings  
 81749 cannot be recognized as reserved words. This rule also implies that reserved words are  
 81750 not recognized except in certain positions in the input, such as after a <newline> or  
 81751 <semicolon>; the grammar presumes that if the reserved word is intended, it is properly  
 81752 delimited by the user, and does not attempt to reflect that requirement directly. Also  
 81753 note that line joining is done before tokenization, as described in [Section 2.2.1](#), so  
 81754 escaped <newline> characters are already removed at this point.
- 81755 Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or  
 81756 applies globally.
- 81757 2. [Redirection to or from filename]
- 81758 The expansions specified in [Section 2.7](#) shall occur. As specified there, exactly one field  
 81759 can result (or the result is unspecified), and there are additional requirements on  
 81760 pathname expansion.
- 81761 3. [Redirection from here-document]
- 81762 Quote removal shall be applied to the word to determine the delimiter that is used to find  
 81763 the end of the here-document that begins after the next <newline>.
- 81764 4. [Case statement termination]
- 81765 When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall  
 81766 result. Otherwise, the token **WORD** shall be returned.
- 81767 5. [NAME in for]
- 81768 When the **TOKEN** meets the requirements for a name (see XBD [Section 3.216](#)), the token  
 81769 identifier **NAME** shall result. Otherwise, the token **WORD** shall be returned.
- 81770 6. [Third word of **for** and **case**]
- 81771 a. [**case** only]
- 81772 When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall  
 81773 result. Otherwise, the token **WORD** shall be returned.

- 81774                   b. [for only]
- 81775                    When the **TOKEN** is exactly the reserved word **in** or **do**, the token identifier for **in**  
81776                    or **do** shall result, respectively. Otherwise, the token **WORD** shall be returned.
- 81777                   (For a. and b.: As indicated in the grammar, a *linebreak* precedes the tokens **in** and **do**. If  
81778                    <newline> characters are present at the indicated location, it is the token after them that is  
81779                    treated in this fashion.)
- 81780                   7. [Assignment preceding command name]
- 81781                    a. [When the first word]
- 81782                    If the **TOKEN** is exactly a reserved word, the token identifier for that reserved  
81783                    word shall result. Otherwise, 7b shall be applied.
- 81784                    b. [Not the first word]
- 81785                    If the **TOKEN** contains an unquoted (as determined while applying rule 4 from  
81786                    Section 2.3) <equals-sign> character that is not part of an embedded parameter  
81787                    expansion, command substitution, or arithmetic expansion construct (as  
81788                    determined while applying rule 5 from Section 2.3):
- 81789                      — If the **TOKEN** begins with '=', then the token **WORD** shall be returned.
- 81790                      — If all the characters in the **TOKEN** preceding the first such <equals-sign>  
81791                      form a valid name (see XBD Section 3.216, on page 63), the token  
81792                      **ASSIGNMENT\_WORD** shall be returned.
- 81793                      — Otherwise, it is implementation-defined whether the token **WORD** or  
81794                      **ASSIGNMENT\_WORD** is returned, or the **TOKEN** is processed in some  
81795                      other way.
- 81796                    Otherwise, the token **WORD** shall be returned.
- 81797                    If a returned **ASSIGNMENT\_WORD** token begins with a valid name, assignment of the  
81798                    value after the first <equals-sign> to the name shall occur as specified in Section 2.9.1. If a  
81799                    returned **ASSIGNMENT\_WORD** token does not begin with a valid name, the way in  
81800                    which the token is processed is unspecified.
- 81801                   8. [**NAME** in function]
- 81802                    When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word  
81803                    shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token  
81804                    identifier **NAME** shall result. Otherwise, rule 7 applies.
- 81805                   9. [Body of function]
- 81806                    Word expansion and assignment shall never occur, even when required by the rules  
81807                    above, when this rule is being parsed. Each **TOKEN** that might either be expanded or  
81808                    have assignment applied to it shall instead be returned as a single **WORD** consisting only  
81809                    of characters that are exactly the token described in Section 2.3.

```

81810      /* -----
81811         The grammar symbols
81812         ----- */
81813      %token  WORD
81814      %token  ASSIGNMENT_WORD
81815      %token  NAME
81816      %token  NEWLINE
81817      %token  IO_NUMBER
81818      %token  IO_LOCATION

81819      /* The following are the operators (see XBD Section 3.243)
81820         containing more than one character. */

81821      %token  AND_IF      OR_IF      DSEMI      SEMI_AND
81822      /*      '&&'      '| |'      ';;'      ';&'      */

81823      %token  DLESS      DGREAT      LESSAND      GREATAND      LESSGREAT      DLESSDASH
81824      /*      '<<'      '>>'      '<&'      '>&'      '<>'      '<<- '      */

81825      %token  CLOBBER
81826      /*      '>|'      */

81827      /* The following are the reserved words. */

81828      %token  If      Then      Else      Elif      Fi      Do      Done
81829      /*      'if'      'then'      'else'      'elif'      'fi'      'do'      'done'      */

81830      %token  Case      Esac      While      Until      For
81831      /*      'case'      'esac'      'while'      'until'      'for'      */

81832      /* These are reserved words, not operator tokens, and are
81833         recognized when reserved words are recognized. */

81834      %token  Lbrace      Rbrace      Bang
81835      /*      '{'      '}'      '!'      */

81836      %token  In
81837      /*      'in'      */

81838      /* -----
81839         The Grammar
81840         ----- */
81841      %start program
81842      %%
81843      program          : linebreak complete_commands linebreak
81844                       | linebreak
81845                       ;
81846      complete_commands: complete_commands newline_list complete_command
81847                       |                               complete_command
81848                       ;
81849      complete_command : list separator_op
81850                       | list
81851                       ;
81852      list              : list separator_op and_or
81853                       |                               and_or
81854                       ;
81855      and_or            :                               pipeline
81856                       | and_or AND_IF linebreak pipeline

```

```

81857         | and_or OR_IF linebreak pipeline
81858         ;
81859     pipeline      : pipe_sequence
81860         | Bang pipe_sequence
81861         ;
81862     pipe_sequence  : command
81863         | pipe_sequence '|' linebreak command
81864         ;
81865     command        : simple_command
81866         | compound_command
81867         | compound_command redirect_list
81868         | function_definition
81869         ;
81870     compound_command : brace_group
81871         | subshell
81872         | for_clause
81873         | case_clause
81874         | if_clause
81875         | while_clause
81876         | until_clause
81877         ;
81878     subshell       : '(' compound_list ')'
81879         ;
81880     compound_list  : linebreak term
81881         | linebreak term separator
81882         ;
81883     term           : term separator and_or
81884         | and_or
81885         ;
81886     for_clause     : For name do_group
81887         | For name sequential_sep do_group
81888         | For name linebreak in sequential_sep do_group
81889         | For name linebreak in wordlist sequential_sep do_group
81890         ;
81891     name           : NAME /* Apply rule 5 */
81892         ;
81893     in             : In /* Apply rule 6 */
81894         ;
81895     wordlist       : wordlist WORD
81896         | WORD
81897         ;
81898     case_clause    : Case WORD linebreak in linebreak case_list Esac
81899         | Case WORD linebreak in linebreak case_list_ns Esac
81900         | Case WORD linebreak in linebreak Esac
81901         ;
81902     case_list_ns   : case_list case_item_ns
81903         | case_item_ns
81904         ;
81905     case_list      : case_list case_item
81906         | case_item
81907         ;
81908     case_item_ns   : pattern_list ')' linebreak
81909         | pattern_list ')' compound_list

```

```

81910                                     ;
81911     case_item                       : pattern_list ')' linebreak      DSEMI linebreak
81912                                     | pattern_list ')' compound_list DSEMI linebreak
81913                                     | pattern_list ')' linebreak      SEMI_AND linebreak
81914                                     | pattern_list ')' compound_list SEMI_AND linebreak
81915                                     ;
81916     pattern_list                      : WORD /* Apply rule 4 */
81917                                     | '(' WORD /* Do not apply rule 4 */
81918                                     | pattern_list '|' WORD /* Do not apply rule 4 */
81919                                     ;
81920     if_clause                          : If compound_list Then compound_list else_part Fi
81921                                     | If compound_list Then compound_list Fi
81922                                     ;
81923     else_part                          : Elif compound_list Then compound_list
81924                                     | Elif compound_list Then compound_list else_part
81925                                     | Else compound_list
81926                                     ;
81927     while_clause                       : While compound_list do_group
81928                                     ;
81929     until_clause                      : Until compound_list do_group
81930                                     ;
81931     function_definition : fname '(' ')' linebreak function_body
81932                                     ;
81933     function_body                    : compound_command /* Apply rule 9 */
81934                                     | compound_command redirect_list /* Apply rule 9 */
81935                                     ;
81936     fname                            : NAME /* Apply rule 8 */
81937                                     ;
81938     brace_group                      : Lbrace compound_list Rbrace
81939                                     ;
81940     do_group                         : Do compound_list Done /* Apply rule 6 */
81941                                     ;
81942     simple_command                   : cmd_prefix cmd_word cmd_suffix
81943                                     | cmd_prefix cmd_word
81944                                     | cmd_prefix
81945                                     | cmd_name cmd_suffix
81946                                     | cmd_name
81947                                     ;
81948     cmd_name                         : WORD /* Apply rule 7a */
81949                                     ;
81950     cmd_word                         : WORD /* Apply rule 7b */
81951                                     ;
81952     cmd_prefix                       : io_redirect
81953                                     | cmd_prefix io_redirect
81954                                     | ASSIGNMENT_WORD
81955                                     | cmd_prefix ASSIGNMENT_WORD
81956                                     ;
81957     cmd_suffix                       : io_redirect
81958                                     | cmd_suffix io_redirect
81959                                     | WORD
81960                                     | cmd_suffix WORD
81961                                     ;
81962     redirect_list                   : io_redirect

```

```

81963         | redirect_list io_redirect
81964         ;
81965     io_redirect :          io_file
81966         | IO_NUMBER io_file
81967         | IO_LOCATION io_file /* Optionally supported */
81968         |          io_here
81969         | IO_NUMBER io_here
81970         | IO_LOCATION io_here /* Optionally supported */
81971         ;
81972     io_file    : '<'      filename
81973         | LESSAND  filename
81974         | '>'      filename
81975         | GREATAND filename
81976         | DGREAT  filename
81977         | LESSGREAT filename
81978         | CLOBBER  filename
81979         ;
81980     filename   : WORD          /* Apply rule 2 */
81981         ;
81982     io_here    : DLESS      here_end
81983         | DLESSDASH here_end
81984         ;
81985     here_end   : WORD          /* Apply rule 3 */
81986         ;
81987     newline_list :          NEWLINE
81988         | newline_list NEWLINE
81989         ;
81990     linebreak  : newline_list
81991         | /* empty */
81992         ;
81993     separator_op : '&'
81994         | ';'
81995         ;
81996     separator  : separator_op linebreak
81997         | newline_list
81998         ;
81999     sequential_sep : ';' linebreak
82000         | newline_list
82001         ;

```

## 82002 2.11 Job Control

82003 Job control is defined (see XBD [Section 3.181](#), on page 57) as a facility that allows users  
82004 selectively to stop (suspend) the execution of processes and continue (resume) their execution at  
82005 a later point. It is jointly supplied by the terminal I/O driver and a command interpreter. The  
82006 shell is one such command interpreter and job control in the shell is enabled by *set -m* (which is  
82007 enabled by default in interactive shells). The remainder of this section describes the job control  
82008 facility provided by the shell. Requirements relating to background jobs stated in this section  
82009 only apply to job-control background jobs.

82010 If the shell has a controlling terminal and it is the controlling process for the terminal session, it  
82011 shall initially set the foreground process group ID associated with the terminal to its own

82012 process group ID. Otherwise, if it has a controlling terminal, it shall initially perform the  
82013 following steps if interactive and may perform them if non-interactive:

- 82014 1. If its process group is the foreground process group associated with the terminal, the shell  
82015 shall set its process group ID to its process ID (if they are not already equal) and set the  
82016 foreground process group ID associated with the terminal to its process group ID.
- 82017 2. If its process group is not the foreground process group associated with the terminal  
82018 (which would result from it being started by a job-control shell as a background job), the  
82019 shell shall either stop itself by sending itself a SIGTTIN signal or, if interactive, attempt to  
82020 read from standard input (which generates a SIGTTIN signal if standard input is the  
82021 controlling terminal). If it is stopped, then when it continues execution (after receiving a  
82022 SIGCONT signal) it shall repeat these steps.

82023 Subsequently, the shell shall change the foreground process group associated with its controlling  
82024 terminal when a foreground job is running as noted in the description below.

82025 When job control is enabled, the shell shall create one or more jobs when it executes a list (see  
82026 [Section 2.9.3](#)) that has one of the following forms:

- 82027 • A single asynchronous AND-OR list
- 82028 • One or more sequentially executed AND-OR lists followed by at most one asynchronous  
82029 AND-OR list

82030 For the purposes of job control, a list that includes more than one asynchronous AND-OR list  
82031 shall be treated as if it were split into multiple separate lists, each ending with an asynchronous  
82032 AND-OR list.

82033 When a job consisting of a single asynchronous AND-OR list is created, it shall form a  
82034 *background job* and the associated process ID shall be that of a child process that is made a  
82035 process group leader, with all other processes (if any) that the shell creates to execute the AND-  
82036 OR list initially having this process ID as their process group ID.

82037 For a list consisting of one or more sequentially executed AND-OR lists followed by at most one  
82038 asynchronous AND-OR list, the whole list shall form a single *foreground job* up until the  
82039 sequentially executed AND-OR lists have all completed execution, at which point the  
82040 asynchronous AND-OR list (if any) shall form a background job as described above.

82041 For each pipeline in a foreground job, if the pipeline is executed while the list is still a  
82042 foreground job, the set of processes comprising the pipeline, and any processes descended from  
82043 it, shall all be in the same process group, unless the shell executes some of the commands in the  
82044 pipeline in the current shell execution environment and others in a subshell environment; in this  
82045 case the process group ID of the current shell need not change (or cannot change if it is the  
82046 session leader), and consequently the process group ID that the other processes all share may  
82047 differ from the process group ID of the current shell (which means that a SIGSTOP, SIGTSTP,  
82048 SIGTTIN, or SIGTTOU signal sent to one of those process groups does not cause the whole  
82049 pipeline to stop).

82050 A background job that was created on execution of an asynchronous AND-OR list can be  
82051 brought into the foreground by means of the *fg* utility (if supported); in this case the entire job  
82052 shall become a single foreground job. If a process that the shell subsequently waits for is part of  
82053 this foreground job and is stopped by a signal, the entire job shall become a suspended job and  
82054 the behavior shall be as if the process had been stopped while the job was running in the  
82055 background.

82056 When a foreground job is created, or a background job is brought into the foreground by the *fg*  
82057 utility, if the shell has a controlling terminal it shall set the foreground process group ID  
82058 associated with the terminal as follows:

- 82059 • If the job was originally created as a background job, the foreground process group ID  
82060 shall be set to the process ID of the process that the shell made a process group leader  
82061 when it executed the asynchronous AND-OR list.
- 82062 • If the job was originally created as a foreground job, the foreground process group ID shall  
82063 be set as follows when each pipeline in the job is executed:
  - 82064 — If the shell is not itself executing, in the current shell execution environment, all of  
82065 the commands in the pipeline, the foreground process group ID shall be set to the  
82066 process group ID that is shared by the other processes executing the pipeline (see  
82067 above).
  - 82068 — If all of the commands in the pipeline are being executed by the shell itself in the  
82069 current shell execution environment, the foreground process group ID shall be set to  
82070 the process group ID of the shell.

82071 When a foreground job terminates, or becomes a suspended job (see below), if the shell has a  
82072 controlling terminal it shall set the foreground process group ID associated with the terminal to  
82073 the process group ID of the shell.

82074 Each background job (whether suspended or not) shall have associated with it a job number and  
82075 a process ID that is known in the current shell execution environment. When a background job is  
82076 brought into the foreground by means of the *fg* utility, the associated job number shall be  
82077 removed from the shell's background jobs list and the associated process ID shall be removed  
82078 from the list of process IDs known in the current shell execution environment.

82079 If a process that the shell is waiting for is part of a foreground job that was started as a  
82080 foreground job and is stopped by a catchable signal (SIGTSTP, SIGTTIN, or SIGTTOU):

- 82081 • If the currently executing AND-OR list within the list comprising the foreground job  
82082 consists of a single pipeline in which all of the commands are simple commands, the shell  
82083 shall either create a suspended job consisting of at least that AND-OR list and the  
82084 remaining (if any) AND-OR lists in the same list, or create a suspended job consisting of  
82085 just that AND-OR list and discard the remaining (if any) AND-OR lists in the same list.
- 82086 • Otherwise, the shell shall create a suspended job consisting of a set of commands, from  
82087 within the list comprising the foreground job, that is unspecified except that the set shall  
82088 include at least the pipeline to which the stopped process belongs. Commands in the  
82089 foreground job that have not already completed and are not included in the suspended job  
82090 shall be discarded.

82091 **Note:** Although only a pipeline of simple commands is guaranteed to remain intact if started in the  
82092 foreground and subsequently suspended, it is possible to ensure that a complex AND-OR list  
82093 will remain intact when suspended by starting it in the background and immediately bringing  
82094 it into the foreground. For example:

```
82095 command1 && command2 | { command3 || command4; } & fg
```

82096 If a process that the shell is waiting for is part of a foreground job that was started as a  
82097 foreground job and is stopped by a SIGSTOP signal, the behavior shall be as described above for  
82098 a catchable signal unless the shell was executing a built-in utility in the current shell execution  
82099 environment when the SIGSTOP was delivered, resulting in the shell itself being stopped by the  
82100 signal, in which case if the shell subsequently receives a SIGCONT signal and has one or more  
82101 child processes that remain stopped, the shell shall create a suspended job as if only those child  
82102 processes had been stopped.

82103 When a suspended job is created as a result of a foreground job being stopped, it shall be  
82104 assigned a job number, and an interactive shell shall write, and a non-interactive shell may  
82105 write, a message to standard error, formatted as described by the *jobs* utility (without the *-l*



option) for a suspended job. The message may indicate that the commands comprising the job include commands that have already completed; in this case the completed commands shall not be repeated if execution of the job is subsequently continued. If the shell is interactive, it shall save the terminal settings before changing them to the settings it needs to read further commands.

When a process associated with a background job is stopped by a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal, the shell shall convert the (non-suspended) background job into a suspended job and an interactive shell shall write a message to standard error, formatted as described by the *jobs* utility (without the *-l* option) for a suspended job, at the following time:

- If *set -b* is enabled, the message shall be written either immediately after the job became suspended or immediately prior to writing the next prompt for input.
- If *set -b* is disabled, the message shall be written immediately prior to writing the next prompt for input.

Execution of a suspended job can be continued as a foreground job by means of the *fg* utility (if supported), or as a (non-suspended) background job either by means of the *bg* utility (if supported) or by sending the stopped processes a SIGCONT signal. The *fg* and *bg* utilities shall send a SIGCONT signal to the process group of the process(es) whose stopped wait status caused the shell to suspend the job. If the shell has a controlling terminal, the *fg* utility shall send the SIGCONT signal after it has set the foreground process group ID associated with the terminal (see above). If the *fg* utility is used from an interactive shell to bring into the foreground a suspended job that was created from a foreground job, before it sends the SIGCONT signal the *fg* utility shall restore the terminal settings to the ones that the shell saved when the job was suspended.

When a background job completes or is terminated by a signal, an interactive shell shall write a message to standard error, formatted as described by the *jobs* utility (without the *-l* option) for a job that completed or was terminated by a signal, respectively, at the following time:

- If *set -b* is enabled, the message shall be written immediately after the job completes or is terminated.
- If *set -b* is disabled, the message shall be written immediately prior to writing the next prompt for input.

In each case above where an interactive shell writes a message immediately prior to writing the next prompt for input, the same message may also be written by a non-interactive shell, at any of the following times:

- After the next time a foreground job terminates or is suspended
- Before the shell parses further input
- Before the shell exits

## 2.12 Signals and Error Handling

If job control is disabled (see the description of *set -m*) when the shell executes an asynchronous AND-OR list, the commands in the list shall inherit from the shell a signal action of ignored (SIG\_IGN) for the SIGINT and SIGQUIT signals. In all other cases, commands executed by the shell shall inherit the same signal actions as those inherited by the shell from its parent unless a signal action is modified by the *trap* special built-in (see *trap*)

When a signal for which a trap has been set is received while the shell is waiting for the completion of a utility executing a foreground command, the trap associated with that signal

82150 shall not be executed until after the foreground command has completed. When the shell is  
 82151 waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a  
 82152 signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit  
 82153 status >128, immediately after which the trap associated with that signal shall be taken.

82154 If multiple signals are pending for the shell for which there are associated trap actions, the order  
 82155 of execution of trap actions is unspecified.

## 82156 2.13 Shell Execution Environment

82157 A shell execution environment consists of the following:

- 82158 • Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- 82159 • Working directory as set by *cd*
- 82160 • File creation mask set by *umask*
- 82161 • File size limit as set by *ulimit*
- 82162 • Current traps set by *trap*
- 82163 • Shell parameters that are set by variable assignment (see the *set* special built-in) or from  
 82164 the System Interfaces volume of POSIX.1-2024 environment inherited by the shell when it  
 82165 begins (see the *export* special built-in)
- 82166 • Shell functions; see [Section 2.9.5](#)
- 82167 • Options turned on at invocation or by *set*
- 82168 • Background jobs and their associated process IDs, and process IDs of child processes  
 82169 created to execute asynchronous AND-OR lists while job control is disabled; together these  
 82170 process IDs constitute the process IDs “known to this shell environment”. If the  
 82171 implementation supports non-job-control background jobs, the list of known process IDs  
 82172 and the list of background jobs may form a single list even though this standard describes  
 82173 them as being updated separately. See [Section 2.9.3.1](#)
- 82174 • Shell aliases; see [Section 2.3.1](#)

82175 Utilities other than the special built-ins (see [Section 2.15](#)) shall be invoked in a separate  
 82176 environment that consists of the following. The initial value of these objects shall be the same as  
 82177 that for the parent shell, except as noted below.

- 82178 • Open files inherited on invocation of the shell, open files controlled by the *exec* special  
 82179 built-in plus any modifications, and additions specified by any redirections to the utility
- 82180 • Current working directory
- 82181 • File creation mask
- 82182 • If the utility is a shell script, traps caught by the shell shall be set to the default values and  
 82183 traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell  
 82184 script, the trap actions (default or ignore) shall be mapped into the appropriate signal  
 82185 handling actions for the utility
- 82186 • Variables with the *export* attribute, along with those explicitly exported for the duration of  
 82187 the command, shall be passed to the utility environment variables
- 82188 • It is unspecified whether environment variables that were passed to the invoking shell  
 82189 when it was invoked itself, but were not used to initialize shell variables (see [Section 2.5.3](#))  
 82190 because they had invalid names, are included in the invoked utility’s environment.

82191 The environment of the shell process shall not be changed by the utility unless explicitly  
82192 specified by the utility description (for example, *cd* and *umask*).

82193 A subshell environment shall be created as a duplicate of the shell environment, except that:

- 82194 • Unless specified otherwise (see *trap*), traps that are not being ignored shall be set to the  
82195 default action.
- 82196 • If the shell is interactive, the subshell shall behave as a non-interactive shell in all respects  
82197 except:
  - 82198 — The expansion of the special parameter '-' may continue to indicate that it is  
82199 interactive.
  - 82200 — The *set -n* option may be ignored.

82201 Changes made to the subshell environment shall not affect the shell environment. Command  
82202 substitution, commands that are grouped with parentheses, and asynchronous AND-OR lists  
82203 shall be executed in a subshell environment. Additionally, each command of a multi-command  
82204 pipeline is in a subshell environment; as an extension, however, any or all commands in a  
82205 pipeline may be executed in the current environment. Except where otherwise stated, all other  
82206 commands shall be executed in the current shell environment.

## 82207 2.14 Pattern Matching Notation

82208 The pattern matching notation described in this section is used to specify patterns for matching  
82209 character strings in the shell. This notation is also used by some other utilities (*find*, *pax*, and  
82210 optionally *make*) and by some system interfaces (*fnmatch()*, *glob()*, and *wordexp()*). Historically,  
82211 pattern matching notation is related to, but slightly different from, the regular expression  
82212 notation described in XBD Chapter 9. For this reason, the description of the rules for this pattern  
82213 matching notation are based on the description of regular expression notation, modified to  
82214 account for the differences.

82215 If an attempt is made to use pattern matching notation to match a string that contains one or  
82216 more bytes that do not form part of a valid character, the behavior is unspecified. Since  
82217 pathnames can contain such bytes, portable applications need to ensure that the current locale is  
82218 the C or POSIX locale when performing pattern matching (or expansion) on arbitrary  
82219 pathnames.

### 82220 2.14.1 Patterns Matching a Single Character

82221 The following patterns shall match a single character: ordinary characters, special pattern  
82222 characters, and pattern bracket expressions. The pattern bracket expression also shall match a  
82223 single collating element.

82224 In a pattern, or part of one, where a shell-quoting `<backslash>` can be used, a `<backslash>`  
82225 character shall escape the following character as described in Section 2.2.1, regardless of whether  
82226 or not the `<backslash>` is inside a bracket expression. (The sequence `"\"` represents one literal  
82227 `<backslash>`.)

82228 In a pattern, or part of one, where a shell-quoting `<backslash>` cannot be used to preserve the  
82229 literal value of a character that would otherwise be treated as special:

- 82230 • A `<backslash>` character that is not inside a bracket expression shall preserve the literal  
82231 value of the following character, unless the following character is in a part of the pattern  
82232 where shell quoting can be used and is a shell quoting character, in which case the

82233 behavior is unspecified.

- 82234 • For the shell only, it is unspecified whether or not a <backslash> character inside a bracket
- 82235 expression preserves the literal value of the following character.

82236 All of the requirements and effects of quoting on ordinary, shell special, and special pattern  
82237 characters shall apply to escaping in this context, except where specified otherwise. (Situations  
82238 where this applies include word expansions when a pattern used in pathname expansion is not  
82239 present in the original word but results from an earlier expansion, or the argument to the *find*  
82240 *-name* or *-path* primary as passed to *find*, or the *pattern* argument to the *fnmatch()* and *glob()*  
82241 functions when FNM\_NOESCAPE or GLOB\_NOESCAPE is not set in *flags*, respectively.)

82242 If a pattern ends with an unescaped <backslash>, the behavior is unspecified.

82243 An ordinary character is a pattern that shall match itself. In a pattern, or part of one, where a  
82244 shell-quoting <backslash> can be used, an ordinary character can be any character in the  
82245 supported character set except for NUL, those special shell characters in [Section 2.2](#) that require  
82246 quoting, and the three special pattern characters described below. In a pattern, or part of one,  
82247 where a shell-quoting <backslash> cannot be used to preserve the literal value of a character that  
82248 would otherwise be treated as special, an ordinary character can be any character in the  
82249 supported character set except for NUL and the three special pattern characters described below.  
82250 Matching shall be based on the bit pattern used for encoding the character, not on the graphic  
82251 representation of the character. If any character (ordinary, shell special, or pattern special) is  
82252 quoted, or escaped with a <backslash>, that pattern shall match the character itself. The  
82253 application shall ensure that it quotes or escapes any character that would otherwise be treated  
82254 as special, in order for it to be matched as an ordinary character.

82255 When unquoted, unescaped, and not inside a bracket expression, the following three characters  
82256 shall have special meaning in the specification of patterns:

- 82257 ? A <question-mark> is a pattern that shall match any character.
- 82258 \* An <asterisk> is a pattern that shall match multiple characters, as described in [Section](#)  
82259 [2.14.2](#).
- 82260 [ A <left-square-bracket> shall introduce a bracket expression if the characters following it  
82261 meet the requirements for bracket expressions stated in XBD [Section 9.3.5](#), except that the  
82262 <exclamation-mark> character ('!') shall replace the <circumflex> character ('^') in its  
82263 role in a non-matching list in the regular expression notation. A bracket expression starting  
82264 with an unquoted <circumflex> character produces unspecified results. A <left-square-  
82265 bracket> that does not introduce a valid bracket expression shall match the character itself.

## 82266 2.14.2 Patterns Matching Multiple Characters

82267 The following rules are used to construct patterns matching multiple characters from patterns  
82268 matching a single character:

- 82269 1. The <asterisk> ('\*') is a pattern that shall match any string, including the null string.
- 82270 2. The concatenation of patterns matching a single character is a valid pattern that shall  
82271 match the concatenation of the single characters or collating elements matched by each of  
82272 the concatenated patterns.
- 82273 3. The concatenation of one or more patterns matching a single character with one or more  
82274 <asterisk> characters is a valid pattern. In such patterns, each <asterisk> shall match a  
82275 string of zero or more characters, matching the greatest possible number of characters  
82276 that still allows the remainder of the pattern to match the string.

82277 **2.14.3 Patterns Used for Filename Expansion**

82278 The rules described so far in [Section 2.14.1](#) and [Section 2.14.2](#) are qualified by the following rules  
82279 that apply when pattern matching notation is used for filename expansion:

- 82280 1. The <slash> character in a pathname shall be explicitly matched by using one or more  
82281 <slash> characters in the pattern; it shall neither be matched by the <asterisk> or  
82282 <question-mark> special characters nor by a bracket expression. <slash> characters in the  
82283 pattern shall be identified before bracket expressions; thus, a <slash> cannot be included  
82284 in a pattern bracket expression used for filename expansion. If a <slash> character is  
82285 found following an unescaped <left-square-bracket> character before a corresponding  
82286 <right-square-bracket> is found, the open bracket shall be treated as an ordinary  
82287 character. For example, the pattern "a[b/c]d" does not match such pathnames as **abd**  
82288 or **a/d**. It only matches a pathname of literally **a[b/c]d**.
- 82289 2. If a filename begins with a <period> ( '.' ), the <period> shall be explicitly matched by  
82290 using a <period> as the first character of the pattern or immediately following a <slash>  
82291 character. The leading <period> shall not be matched by:
  - 82292 • The <asterisk> or <question-mark> special characters
  - 82293 • A bracket expression containing a non-matching list, such as "[!a]", a range  
82294 expression, such as "[%-0]", or a character class expression, such as  
82295 "[[:punct:]]"

82296 It is unspecified whether an explicit <period> in a bracket expression matching list, such  
82297 as "[.abc]", can match a leading <period> in a filename.

- 82298 3. If a specified pattern contains any '\*', '?', or '[' characters that will be treated as  
82299 special (see [Section 2.14.1](#)), it shall be matched against existing filenames and pathnames,  
82300 as appropriate; if directory entries for dot and dot-dot exist, they may be ignored. Each  
82301 component that contains any such characters shall require read permission in the  
82302 directory containing that component. Each component that contains a <backslash> that  
82303 will be treated as special may require read permission in the directory containing that  
82304 component. Any component, except the last, that does not contain any '\*', '?', or '['  
82305 characters that will be treated as special shall require search permission. If these  
82306 permissions are denied, or if an attempt to open or search a pathname as a directory, or an  
82307 attempt to read an opened directory, fails because of an error condition that is related to  
82308 file system contents, this shall not be considered an error and pathname expansion shall  
82309 continue as if the pathname had named an existing directory which had been successfully  
82310 opened and read, or searched, and no matching directory entries had been found in it. For  
82311 other error conditions it is unspecified whether pathname expansion fails or they are  
82312 treated the same as when permission is denied.

82313 For example, given the pattern:

```
82314 /foo/bar/x*/bam
```

82315 search permission is needed for directories **/** and **foo**, search and read permissions are  
82316 needed for directory **bar**, and search permission is needed for each **x\*** directory.

82317 If the pattern matches any existing filenames or pathnames, the pattern shall be replaced  
82318 with those filenames and pathnames, sorted according to the collating sequence in effect  
82319 in the current locale. If this collating sequence does not have a total ordering of all  
82320 characters (see [XBD Section 7.3.2](#), on page 139), any filenames or pathnames that collate  
82321 equally shall be further compared byte-by-byte using the collating sequence for the  
82322 POSIX locale.

82323 If the pattern contains an open bracket ( '[' ) that does not introduce a bracket expression

82324 as in XBD [Section 9.3.5](#), it is unspecified whether other unquoted '\*', '?', '[' or  
 82325 <backslash> characters within the same slash-delimited component of the pattern retain  
 82326 their special meanings or are treated as ordinary characters. For example, the pattern  
 82327 "a\*[/b\*" may match all filenames beginning with 'b' in the directory "a\*[" or it may  
 82328 match all filenames beginning with 'b' in all directories with names beginning with 'a'  
 82329 and ending with '['.

82330 If the pattern does not match any existing filenames or pathnames, the pattern string shall  
 82331 be left unchanged.

82332 **Note:** A future version of this standard may require that directory entries for dot and dot-dot  
 82333 are ignored (if they exist) when matching patterns against existing filenames. For  
 82334 example, when expanding the pattern ".\*" the result would not include dot and dot-  
 82335 dot.

82336 4. If a specified pattern does not contain any '\*', '?' or '[' characters that will be treated  
 82337 as special, the pattern string shall be left unchanged.

## 82338 2.15 Special Built-In Utilities

82339 The following "special built-in" utilities shall be supported in the shell command language. The  
 82340 output of each command, if any, shall be written to standard output, subject to the normal  
 82341 redirection and piping possible with all commands.

82342 The term "built-in" implies that there is no need to execute a separate executable file because the  
 82343 utility is implemented in the shell itself. An implementation may choose to make any utility a  
 82344 built-in; however, the special built-in utilities described here differ from regular built-in utilities  
 82345 in two respects:

- 82346 1. An error in a special built-in utility may cause a shell executing that utility to abort, while  
 82347 an error in a regular built-in utility shall not cause a shell executing that utility to abort.  
 82348 (See [Section 2.8.1](#) for the consequences of errors on interactive and non-interactive shells.)  
 82349 If a special built-in utility encountering an error does not abort the shell, its exit value  
 82350 shall be non-zero.
- 82351 2. As described in [Section 2.9.1](#), variable assignments preceding the invocation of a special  
 82352 built-in utility affect the current execution environment; this shall not be the case with a  
 82353 regular built-in or other utility.

82354 The special built-in utilities in this section need not be provided in a manner accessible via the  
 82355 *exec* family of functions defined in the System Interfaces volume of POSIX.1-2024.

82356 Some of the special built-ins are described as conforming to XBD [Section 12.2](#). For those that are  
 82357 not, the requirement in [Section 1.4](#) that "--" be recognized as a first argument to be discarded  
 82358 does not apply and a conforming application shall not use that argument.

82359 **NAME**

82360 break — exit from for, while, or until loop

82361 **SYNOPSIS**82362 break [*n*]82363 **DESCRIPTION**

82364 If *n* is specified, the *break* utility shall exit from the *n*th enclosing **for**, **while**, or **until** loop. If *n* is  
82365 not specified, *break* shall behave as if *n* was specified as 1. Execution shall continue with the  
82366 command immediately following the exited loop. The application shall ensure that the value of  
82367 *n* is a positive decimal integer. If *n* is greater than the number of enclosing loops, the outermost  
82368 enclosing loop shall be exited. If there is no enclosing loop, the behavior is unspecified.

82369 A loop shall enclose a *break* or *continue* command if the loop lexically encloses the command. A  
82370 loop lexically encloses a *break* or *continue* command if the command is:

- 82371 • Executing in the same execution environment (see [Section 2.13](#)) as the compound-list of the  
82372 loop's do-group (see [Section 2.10.2](#)), and
- 82373 • Contained in a compound-list associated with the loop (either in the compound-list of the  
82374 loop's do-group or, if the loop is a **while** or **until** loop, in the compound-list following the  
82375 **while** or **until** reserved word), and
- 82376 • Not in the body of a function whose function definition command (see [Section 2.9.5](#)) is  
82377 contained in a compound-list associated with the loop.

82378 If *n* is greater than the number of lexically enclosing loops and there is a non-lexically enclosing  
82379 loop in progress in the same execution environment as the *break* or *continue* command, it is  
82380 unspecified whether that loop encloses the command.

82381 **OPTIONS**

82382 None.

82383 **OPERANDS**

82384 See the DESCRIPTION.

82385 **STDIN**

82386 Not used.

82387 **INPUT FILES**

82388 None.

82389 **ENVIRONMENT VARIABLES**

82390 None.

82391 **ASYNCHRONOUS EVENTS**

82392 Default.

82393 **STDOUT**

82394 Not used.

82395 **STDERR**

82396 The standard error shall be used only for diagnostic messages.

82397 **OUTPUT FILES**

82398 None.

82399 **EXTENDED DESCRIPTION**

82400 None.

82401 **EXIT STATUS**

82402 0 Successful completion.

82403 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.82404 **CONSEQUENCES OF ERRORS**

82405 Default.

82406 **APPLICATION USAGE**

82407 None.

82408 **EXAMPLES**

```
82409     for i in *
82410     do
82411         if test -d "$i"
82412         then break
82413         fi
82414     done
```

82415 The results of running the following example are unspecified: there are two loops in progress  
 82416 when the *break* command is executed, and they are in the same execution environment, but  
 82417 neither loop is lexically enclosing the *break* command. (There are no loops lexically enclosing the  
 82418 *continue* commands, either.)

```
82419     foo() {
82420         for j in 1 2; do
82421             echo 'break 2' >/tmp/do_break
82422             echo "  sourcing /tmp/do_break ($j)..."
82423             # the behavior of the break from running the following command
82424             # results in unspecified behavior:
82425             . /tmp/do_break

82426             do_continue() { continue 2; }
82427             echo "  running do_continue ($j)..."
82428             # the behavior of the continue in the following function call
82429             # results in unspecified behavior (if execution reaches this
82430             # point):
82431             do_continue

82432             trap 'continue 2' USR1
82433             echo "  sending SIGUSR1 to self ($j)..."
82434             # the behavior of the continue in the trap invoked from the
82435             # following signal results in unspecified behavior (if
82436             # execution reaches this point):
82437             kill -s USR1 $$
82438             sleep 1
82439         done
82440     }
82441     for i in 1 2; do
82442         echo "running foo ($i)..."
82443         foo
82444     done
```



82445 **RATIONALE**

82446 In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer  
82447 to a label associated with the appropriate loop as a preferable alternative to the *n* method.  
82448 However, this volume of POSIX.1-2024 does reserve the name space of command names ending  
82449 with a <colon>. It is anticipated that a future implementation could take advantage of this and  
82450 provide something like:

```
82451 outofloop: for i in a b c d e  
82452 do  
82453     for j in 0 1 2 3 4 5 6 7 8 9  
82454     do  
82455         if test -r "${i}${j}"  
82456         then break outofloop  
82457         fi  
82458     done  
82459 done
```

82460 and that this might be standardized after implementation experience is achieved.

82461 **FUTURE DIRECTIONS**

82462 None.

82463 **SEE ALSO**

82464 [Section 2.15](#)

82465 **CHANGE HISTORY**82466 **Issue 6**

82467 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
82468 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
82469 behavior is intended.

82470 **Issue 7**

82471 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0046 [842] is applied.

82472 **Issue 8**

82473 Austin Group Defect 1058 is applied, clarifying that the requirement for *n* to be a positive  
82474 decimal integer is a requirement on the application.

82475 **NAME**

82476 colon — null utility

82477 **SYNOPSIS**82478 : [*argument...*]82479 **DESCRIPTION**82480 This utility shall do nothing except return a 0 exit status. It is used when a command is needed,  
82481 as in the **then** condition of an **if** command, but nothing is to be done by the command.82482 **OPTIONS**82483 This utility shall not recognize the "--" argument in the manner specified by Guideline 10 of  
82484 XBD [Section 12.2](#) (on page 215).

82485 Implementations shall not support any options.

82486 **OPERANDS**

82487 See the DESCRIPTION.

82488 **STDIN**

82489 Not used.

82490 **INPUT FILES**

82491 None.

82492 **ENVIRONMENT VARIABLES**

82493 None.

82494 **ASYNCHRONOUS EVENTS**

82495 Default.

82496 **STDOUT**

82497 Not used.

82498 **STDERR**

82499 Not used.

82500 **OUTPUT FILES**

82501 None.

82502 **EXTENDED DESCRIPTION**

82503 None.

82504 **EXIT STATUS**

82505 Zero.

82506 **CONSEQUENCES OF ERRORS**

82507 None.

82508 **APPLICATION USAGE**82509 See the APPLICATION USAGE for *true*.82510 **EXAMPLES**82511 : "\${X=abc}"  
82512 if false  
82513 then :  
82514 else printf '%s\n' "\$X"  
82515 fi  
82516 **abc**

82517 As with any of the special built-ins, the null utility can also have variable assignments and

82518 redirections associated with it, such as:

82519 `x=y : > z`

82520 which sets variable *x* to the value *y* (so that it persists after the null utility completes) and creates  
82521 or truncates file *z*; if the file cannot be created or truncated, a non-interactive shell exits (see  
82522 [Section 2.8.1](#)).

82523 **RATIONALE**

82524 None.

82525 **FUTURE DIRECTIONS**

82526 None.

82527 **SEE ALSO**

82528 [Section 2.15](#), *true*

82529 **CHANGE HISTORY**

82530 **Issue 6**

82531 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
82532 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
82533 behavior is intended.

82534 **Issue 7**

82535 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82536 **Issue 8**

82537 Austin Group Defect 1272 is applied, clarifying that the null utility does not process its  
82538 arguments, does not recognize the "--" end-of-options delimiter, does not support any options,  
82539 and does not write to standard error.

82540 Austin Group Defect 1640 is applied, changing the APPLICATION USAGE section.

82541 **NAME**

82542 continue — continue for, while, or until loop

82543 **SYNOPSIS**82544 continue [*n*]82545 **DESCRIPTION**

82546 If *n* is specified, the *continue* utility shall return to the top of the *n*th enclosing **for**, **while**, or **until**  
82547 loop. If *n* is not specified, *continue* shall behave as if *n* was specified as 1. Returning to the top of  
82548 the loop involves repeating the condition list of a **while** or **until** loop or performing the next  
82549 assignment of a **for** loop, and re-executing the loop if appropriate.

82550 The application shall ensure that the value of *n* is a positive decimal integer. If *n* is greater than  
82551 the number of enclosing loops, the outermost enclosing loop shall be used. If there is no  
82552 enclosing loop, the behavior is unspecified.

82553 The meaning of “enclosing” shall be as specified in the description of the *break* utility.

82554 **OPTIONS**

82555 None.

82556 **OPERANDS**

82557 See the DESCRIPTION.

82558 **STDIN**

82559 Not used.

82560 **INPUT FILES**

82561 None.

82562 **ENVIRONMENT VARIABLES**

82563 None.

82564 **ASYNCHRONOUS EVENTS**

82565 Default.

82566 **STDOUT**

82567 Not used.

82568 **STDERR**

82569 The standard error shall be used only for diagnostic messages.

82570 **OUTPUT FILES**

82571 None.

82572 **EXTENDED DESCRIPTION**

82573 None.

82574 **EXIT STATUS**

82575 0 Successful completion.

82576 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.82577 **CONSEQUENCES OF ERRORS**

82578 Default.

**82579 APPLICATION USAGE**

82580 None.

**82581 EXAMPLES**

```
82582     for i in *
82583     do
82584         if test -d "$i"
82585         then continue
82586         fi
82587         printf "%s" is not a directory.\n' "$i"
82588     done
```

**82589 RATIONALE**

82590 None.

**82591 FUTURE DIRECTIONS**

82592 None.

**82593 SEE ALSO**

82594 [Section 2.15](#)

**82595 CHANGE HISTORY****82596 Issue 6**

82597 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
82598 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
82599 behavior is intended.

**82600 Issue 7**

82601 The example is changed to use the *printf* utility rather than *echo*.

82602 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0046 [842] is applied.

**82603 Issue 8**

82604 Austin Group Defect 1058 is applied, clarifying that the requirement for *n* to be a positive  
82605 decimal integer is a requirement on the application.

82606 **NAME**

82607 dot — execute commands in the current environment

82608 **SYNOPSIS**

82609 . *file*

82610 **DESCRIPTION**

82611 The shell shall tokenize (see [Section 2.3](#)) the contents of the *file*, parse the tokens (see [Section 2.10](#)), and execute the resulting commands in the current environment. It is unspecified whether  
82612 the commands are parsed and executed as a *program* (as for a shell script) or are parsed as a  
82613 single *compound\_list* that is executed after the entire file has been parsed.  
82614

82615 If *file* does not contain a <slash>, the shell shall use the search path specified by *PATH* to find the  
82616 directory containing *file*. Unlike normal command search, however, the file searched for by the  
82617 *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort;  
82618 an interactive shell shall write a diagnostic message to standard error.

82619 The *dot* special built-in shall support XBD [Section 12.2](#) (on page 215), except for Guidelines 1 and  
82620 2.

82621 **OPTIONS**

82622 None.

82623 **OPERANDS**

82624 See the DESCRIPTION.

82625 **STDIN**

82626 Not used.

82627 **INPUT FILES**

82628 See the DESCRIPTION.

82629 **ENVIRONMENT VARIABLES**

82630 See the DESCRIPTION.

82631 **ASYNCHRONOUS EVENTS**

82632 Default.

82633 **STDOUT**

82634 Not used.

82635 **STDERR**

82636 The standard error shall be used only for diagnostic messages.

82637 **OUTPUT FILES**

82638 None.

82639 **EXTENDED DESCRIPTION**

82640 None.

82641 **EXIT STATUS**

82642 If no readable file was found or if the commands in the file could not be parsed, and the shell is  
82643 interactive (and therefore does not abort; see [Section 2.8.1](#)), the exit status shall be non-zero.  
82644 Otherwise, return the value of the last command executed, or a zero exit status if no command is  
82645 executed.

82646 **CONSEQUENCES OF ERRORS**

82647 Default.

82648 **APPLICATION USAGE**

82649 None.

82650 **EXAMPLES**

```
82651     cat foobar
82652     foo=hello bar=world
82653     . ./foobar
82654     echo $foo $bar
82655     hello world
```

82656 **RATIONALE**

82657 Some older implementations searched the current directory for the *file*, even if the value of *PATH*  
82658 disallowed it. This behavior was omitted from this volume of POSIX.1-2024 due to concerns  
82659 about introducing the susceptibility to trojan horses that the user might be trying to avoid by  
82660 leaving **dot** out of *PATH*.

82661 The KornShell version of *dot* takes optional arguments that are set to the positional parameters.  
82662 This is a valid extension that allows a *dot* script to behave identically to a function.

82663 **FUTURE DIRECTIONS**

82664 None.

82665 **SEE ALSO**82666 [Section 2.15, \*return\*](#)82667 **CHANGE HISTORY**82668 **Issue 6**

82669 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
82670 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
82671 behavior is intended.

82672 **Issue 7**

82673 SD5-XCU-ERN-164 is applied.

82674 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0038 [114] and XCU/TC1-2008/0039  
82675 [214] are applied.

82676 **Issue 8**

82677 Austin Group Defect 252 is applied, adding a requirement for *dot* to support XBD [Section 12.2](#)  
82678 (except for Guidelines 1 and 2, since the utility's name is ' . ').

82679 Austin Group Defect 953 is applied, clarifying how the commands in the *file* are parsed.

82680 Austin Group Defect 1265 is applied, updating the DESCRIPTION to align with the changes  
82681 made to [Section 2.8.1](#) between Issue 6 and Issue 7.

82682 **NAME**

82683 eval — construct command by concatenating arguments

82684 **SYNOPSIS**82685 eval [*argument...*]82686 **DESCRIPTION**

82687 The *eval* utility shall construct a command string by concatenating *arguments* together,  
82688 separating each with a <space> character. The constructed command string shall be tokenized  
82689 (see [Section 2.3](#)), parsed (see [Section 2.10](#)), and executed by the shell in the current environment.  
82690 It is unspecified whether the commands are parsed and executed as a *program* (as for a shell  
82691 script) or are parsed as a single *compound\_list* that is executed after the entire constructed  
82692 command string has been parsed.

82693 **OPTIONS**

82694 None.

82695 **OPERANDS**

82696 See the DESCRIPTION.

82697 **STDIN**

82698 Not used.

82699 **INPUT FILES**

82700 None.

82701 **ENVIRONMENT VARIABLES**

82702 None.

82703 **ASYNCHRONOUS EVENTS**

82704 Default.

82705 **STDOUT**

82706 Not used.

82707 **STDERR**

82708 The standard error shall be used only for diagnostic messages.

82709 **OUTPUT FILES**

82710 None.

82711 **EXTENDED DESCRIPTION**

82712 None.

82713 **EXIT STATUS**

82714 If there are no *arguments*, or only null *arguments*, *eval* shall return a zero exit status; otherwise, it  
82715 shall return the exit status of the command defined by the string of concatenated *arguments*  
82716 separated by <space> characters, or a non-zero exit status if the concatenation could not be  
82717 parsed as a command and the shell is interactive (and therefore did not abort).

82718 **CONSEQUENCES OF ERRORS**

82719 Default.



**82720 APPLICATION USAGE**

82721 Since *eval* is not required to recognize the "--" end of options delimiter, in cases where the  
82722 argument(s) to *eval* might begin with '-' it is recommended that the first argument is prefixed  
82723 by a string that will not alter the commands to be executed, such as a <space> character:

```
82724 eval " $commands"
```

82725 or:

```
82726 eval " $(some_command)"
```

**82727 EXAMPLES**

```
82728 foo=10 x=foo
```

```
82729 y='$'$x
```

```
82730 echo $y
```

```
82731 $foo
```

```
82732 eval y='$'$x
```

```
82733 echo $y
```

```
82734 10
```

**82735 RATIONALE**

82736 This standard allows, but does not require, *eval* to recognize "--". Although this means  
82737 applications cannot use "--" to protect against options supported as an extension (or errors  
82738 reported for unsupported options), the nature of the *eval* utility is such that other means can be  
82739 used to provide this protection (see APPLICATION USAGE above).

**82740 FUTURE DIRECTIONS**

82741 None.

**82742 SEE ALSO**

82743 [Section 2.15](#)

**82744 CHANGE HISTORY****82745 Issue 6**

82746 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
82747 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
82748 behavior is intended.

**82749 Issue 7**

82750 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82751 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0040 [114], XCU/TC1-2008/0041 [163],  
82752 and XCU/TC1-2008/0042 [163] are applied.

**82753 Issue 8**

82754 Austin Group Defect 953 is applied, clarifying how the commands in the constructed command  
82755 string are parsed.

82756 **NAME**82757 `exec` — perform redirections in the current shell or execute a utility82758 **SYNOPSIS**82759 `exec` [*utility* [*argument...*]]82760 **DESCRIPTION**

82761 If `exec` is specified with no operands, any redirections associated with the `exec` command shall be  
82762 made in the current shell execution environment. If any file descriptors with numbers greater  
82763 than 2 are opened by those redirections, it is unspecified whether those file descriptors remain  
82764 open when the shell invokes another utility. Scripts concerned that child shells could misuse  
82765 open file descriptors can always close them explicitly, as shown in one of the following  
82766 examples. If the result of the redirections would be that file descriptor 0, 1, or 2 is closed,  
82767 implementations may open the file descriptor to an unspecified file.

82768 If `exec` is specified with a *utility* operand, the shell shall execute a non-built-in utility as described  
82769 in [Section 2.9.1.6](#) with *utility* as the command name and the *argument* operands (if any) as the  
82770 command arguments.

82771 If the `exec` command fails, a non-interactive shell shall exit from the current shell execution  
82772 UP environment; an interactive shell may exit from a subshell environment but shall not exit if the  
82773 current shell environment is not a subshell environment.

82774 If the `exec` command fails and the shell does not exit, any redirections associated with the `exec`  
82775 command that were successfully made shall take effect in the current shell execution  
82776 environment.

82777 The `exec` special built-in shall support XBD [Section 12.2](#) (on page 215).

82778 **OPTIONS**

82779 None.

82780 **OPERANDS**

82781 See the DESCRIPTION.

82782 **STDIN**

82783 Not used.

82784 **INPUT FILES**

82785 None.

82786 **ENVIRONMENT VARIABLES**82787 The following environment variable shall affect the execution of `exec`:

82788 *PATH* Determine the search path when looking for the utility given as the *utility* operand;  
82789 see XBD [Section 8.3](#) (on page 174).

82790 **ASYNCHRONOUS EVENTS**

82791 Default.

82792 **STDOUT**

82793 Not used.

82794 **STDERR**

82795 The standard error shall be used only for diagnostic messages.

82796 **OUTPUT FILES**

82797 None.

82798 **EXTENDED DESCRIPTION**

82799 None.

82800 **EXIT STATUS**

82801 If *utility* is specified and is executed, *exec* shall not return to the shell; rather, the exit status of the  
82802 current shell execution environment shall be the exit status of *utility*. If *utility* is specified and an  
82803 attempt to execute it as a non-built-in utility fails, the exit status shall be as described in [Section](#)  
82804 [2.9.1.6](#). If a redirection error occurs (see [Section 2.8.1](#)), the exit status shall be a value in the range  
82805 1–125. Otherwise, *exec* shall return a zero exit status.

82806 **CONSEQUENCES OF ERRORS**

82807 Default.

82808 **APPLICATION USAGE**

82809 None.

82810 **EXAMPLES**82811 Open *readfile* as file descriptor 3 for reading:82812 `exec 3< readfile`82813 Open *writefile* as file descriptor 4 for writing:82814 `exec 4> writefile`

82815 Make file descriptor 5 a copy of file descriptor 0:

82816 `exec 5<&0`

82817 Close file descriptor 3:

82818 `exec 3<&-`82819 Cat the file **maggie** by replacing the current shell with the *cat* utility:82820 `exec cat maggie`

82821 An application that is not concerned with strict conformance can make use of optional `%g`  
82822 support known to be present in the implementation's *printf* utility by ensuring that any shell  
82823 built-in version is not executed instead, and using a subshell so that the shell continues  
82824 afterwards:

82825 `(exec printf '%g\n' "$float_value")`82826 **RATIONALE**

82827 Most historical implementations were not conformant in that:

82828 `foo=bar exec cmd`82829 did not pass **foo** to **cmd**.82830 **FUTURE DIRECTIONS**

82831 None.

82832 **SEE ALSO**82833 [Section 2.15](#)82834 **CHANGE HISTORY**

- 82835 **Issue 6**  
82836 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
82837 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
82838 behavior is intended.
- 82839 **Issue 7**  
82840 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 82841 **Issue 8**  
82842 Austin Group Defect 252 is applied, adding a requirement for *exec* to support XBD [Section 12.2](#).  
82843 Austin Group Defect 1157 is applied, clarifying the execution of non-built-in utilities.  
82844 Austin Group Defect 1587 is applied, changing the ENVIRONMENT VARIABLES section.

82845 **NAME**

82846 exit — cause the shell to exit

82847 **SYNOPSIS**82848 exit [*n*]82849 **DESCRIPTION**

82850 The *exit* utility shall cause the shell to exit from its current execution environment. If the current  
82851 execution environment is a subshell environment, the shell shall exit from the subshell  
82852 environment and continue in the environment from which that subshell environment was  
82853 invoked; otherwise, the shell utility shall terminate. The wait status of the shell or subshell shall  
82854 be determined by the unsigned decimal integer *n*, if specified.

82855 If *n* is specified and has a value between 0 and 255 inclusive, the wait status of the shell or  
82856 subshell shall indicate that it exited with exit status *n*. If *n* is specified and has a value greater  
82857 than 256 that corresponds to an exit status the shell assigns to commands terminated by a valid  
82858 signal (see [Section 2.8.2](#)), the wait status of the shell or subshell shall indicate that it was  
82859 terminated by that signal. No other actions associated with the signal, such as execution of *trap*  
82860 actions or creation of a core image, shall be performed by the shell.

82861 If *n* is specified and is not an unsigned decimal integer, or has a value of 256, or has a value  
82862 greater than 256 but not corresponding to an exit status the shell assigns to commands  
82863 terminated by a valid signal, the wait status of the shell or subshell is unspecified.

82864 If *n* is not specified, the result shall be as if *n* were specified with the current value of the special  
82865 parameter '?' (see [Section 2.5.2](#)), except that if the *exit* command would cause the end of  
82866 execution of a *trap* action, the value for the special parameter '?' that is considered "current"  
82867 shall be the value it had immediately preceding the *trap* action.

82868 A *trap* action on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is  
82869 invoked in that *trap* action itself, in which case the shell shall exit immediately. It is unspecified  
82870 whether setting a new *trap* action on **EXIT** during execution of a *trap* action on **EXIT** will cause  
82871 the new *trap* action to be executed before the shell terminates.

82872 **OPTIONS**

82873 None.

82874 **OPERANDS**

82875 See the DESCRIPTION.

82876 **STDIN**

82877 Not used.

82878 **INPUT FILES**

82879 None.

82880 **ENVIRONMENT VARIABLES**

82881 None.

82882 **ASYNCHRONOUS EVENTS**

82883 Default.

82884 **STDOUT**

82885 Not used.

82886 **STDERR**

82887 The standard error shall be used only for diagnostic messages.

82888 **OUTPUT FILES**

82889 None.

82890 **EXTENDED DESCRIPTION**

82891 None.

82892 **EXIT STATUS**82893 The *exit* utility causes the shell to exit from its current execution environment, and therefore does  
82894 not itself return an exit status.82895 **CONSEQUENCES OF ERRORS**

82896 Default.

82897 **APPLICATION USAGE**82898 As explained in other sections, certain exit status values have been reserved for special uses and  
82899 should be used by applications only for those purposes:

82900 126 A file to be executed was found, but it was not an executable utility.

82901 127 A utility to be executed was not found.

82902 128 An unrecoverable read error was detected by the shell while reading commands, except  
82903 from the *file* operand of the *dot* special built-in.

82904 &gt;128 A command was interrupted by a signal.

82905 **EXAMPLES**82906 Exit with a *true* value:82907 `exit 0`82908 Exit with a *false* value:82909 `exit 1`

82910 Propagate error handling from within a subshell:

82911 

```
(  
82912     command1 || exit 1  
82913     command2 || exit 1  
82914     exec command3  
82915 ) > outputfile || exit 1  
82916 echo "outputfile created successfully"
```

82917 **RATIONALE**82918 The behavior of *exit* when given an invalid argument or unknown option is unspecified, because  
82919 of differing practices in the various historical implementations. A value larger than 255 might be  
82920 truncated by the shell, and be unavailable even to a parent process that uses *waitid()* to get the  
82921 full exit value. It is recommended that implementations that detect any usage error should cause  
82922 a non-zero exit status (or, if the shell is interactive and the error does not cause the shell to abort,  
82923 store a non-zero value in "\$?"), but even this was not done historically in all shells.82924 See also [Section C.2.8.2](#) (on page 3894).82925 **FUTURE DIRECTIONS**

82926 None.

82927 **SEE ALSO**82928 [Section 2.15](#)82929 **CHANGE HISTORY**82930 **Issue 6**

82931 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
82932 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
82933 behavior is intended.

82934 **Issue 7**

82935 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0047 [717], XCU/TC2-2008/0048  
82936 [960], XCU/TC2-2008/0049 [717], and XCU/TC2-2008/0050 [960] are applied.

82937 **Issue 8**

82938 Austin Group Defect 51 is applied, specifying the behavior when *n* has a value greater than 256  
82939 that corresponds to an exit status the shell assigns to commands terminated by a valid signal.

82940 Austin Group Defect 1029 is applied, changing ``trap'' to ``trap action'' in the DESCRIPTION  
82941 section.

82942 Austin Group Defect 1309 is applied, changing the EXIT STATUS section.

82943 Austin Group Defect 1425 is applied, clarifying the requirements for a *trap* action on **EXIT**.

82944 Austin Group Defect 1602 is applied, clarifying the behavior of *exit* in a *trap* action.

82945 Austin Group Defect 1629 is applied, adding exit status 128 to the APPLICATION USAGE  
82946 section.

82947 **NAME**

82948 export — set the export attribute for variables

82949 **SYNOPSIS**

82950 export name [=word] . . .

82951 export -p

82952 **DESCRIPTION**

82953 The shell shall give the *export* attribute to the variables corresponding to the specified *names*,  
82954 which shall cause them to be in the environment of subsequently executed commands. If the  
82955 name of a variable is followed by =*word*, then the value of that variable shall be set to *word*.

82956 The *export* special built-in shall be a declaration utility. Therefore, if *export* is recognized as the  
82957 command name of a simple command, then subsequent words of the form *name=word* shall be  
82958 expanded in an assignment context. See [Section 2.9.1.1](#).

82959 The *export* special built-in shall support XBD [Section 12.2](#).

82960 When **-p** is specified, *export* shall write to the standard output the names and values of all  
82961 exported variables, in the following format:

82962 "export %s=%s\n", <name>, <value>

82963 if *name* is set, and:

82964 "export %s\n", <name>

82965 if *name* is unset.

82966 The shell shall format the output, including the proper use of quoting, so that it is suitable for  
82967 reinput to the shell as commands that achieve the same exporting results, except:

- 82968 1. Read-only variables with values cannot be reset.
- 82969 2. Variables that were unset at the time they were output need not be reset to the unset state  
82970 if a value is assigned to the variable between the time the state was saved and the time at  
82971 which the saved output is reinput to the shell.

82972 When no arguments are given, the results are unspecified.

82973 **OPTIONS**

82974 See the DESCRIPTION.

82975 **OPERANDS**

82976 See the DESCRIPTION.

82977 **STDIN**

82978 Not used.

82979 **INPUT FILES**

82980 None.

82981 **ENVIRONMENT VARIABLES**

82982 None.

82983 **ASYNCHRONOUS EVENTS**

82984 Default.



82985 **STDOUT**

82986 See the DESCRIPTION.

82987 **STDERR**

82988 The standard error shall be used only for diagnostic messages.

82989 **OUTPUT FILES**

82990 None.

82991 **EXTENDED DESCRIPTION**

82992 None.

82993 **EXIT STATUS**

82994 0 Successful completion.

82995 >0 At least one operand could not be processed as requested, such as a *name* operand that  
 82996 could not be exported or an attempt to modify a *readonly* variable using a *name=word*  
 82997 operand, or the **-p** option was specified and a write error occurred.

82998 **CONSEQUENCES OF ERRORS**

82999 Default.

83000 **APPLICATION USAGE**83001 Note that, unless *X* was previously marked *readonly*, the value of "\$?" after:83002 `export X=$(false)`

83003 will be 0 (because *export* successfully set *X* to the empty string) and that execution continues,  
 83004 even if *set -e* is in effect. In order to detect command substitution failures, a user must separate  
 83005 the assignment from the export, as in:

83006 `X=$(false)`83007 `export X`

83008 In shells that support extended assignment syntax, for example to allow an array to be  
 83009 populated with a single assignment, such extensions can typically only be used in assignments  
 83010 specified as arguments to *export* if the command word is literally *export*, and not if it is some  
 83011 other word that expands to *export*. For example:

83012 `# Shells that support array assignment as an extension generally`83013 `# support this:`83014 `export x=(1 2 3); echo ${x[0]} # outputs 1`83015 `# But generally do not support this:`83016 `e=export; $e x=(1 2 3); echo ${x[0]} # syntax error`83017 **EXAMPLES**83018 Export *PWD* and *HOME* variables:83019 `export PWD HOME`83020 Set and export the *PATH* variable:83021 `export PATH="/local/bin:$PATH"`

83022 Save and restore all exported variables:

83023 `export -p > temp-file`83024 `unset a lot of variables`83025 `... processing`83026 `./temp-file`

83027 **Note:** If LANG, LC\_CTYPE or LC\_ALL are left altered or unset in the above example prior to sourcing  
83028 `temp-file`, the results may be undefined.

### 83029 RATIONALE

83030 Some historical shells use the no-argument case as the functional equivalent of what is required  
83031 here with `-p`. This feature was left unspecified because it is not historical practice in all shells,  
83032 and some scripts may rely on the now-unspecified results on their implementations. Attempts to  
83033 specify the `-p` output as the default case were unsuccessful in achieving consensus. The `-p`  
83034 option was added to allow portable access to the values that can be saved and then later restored  
83035 using; for example, a *dot* script.

83036 Some implementations extend the shell's assignment syntax, for example to allow an array to be  
83037 populated with a single assignment, and in order for such an extension to be usable in  
83038 assignments specified as arguments to *export* these shells have *export* as a separate token in their  
83039 grammar. This standard only permits an extension of this nature when the input to the shell  
83040 would contain a syntax error according to the standard grammar. Note that although *export* can  
83041 be a separate token in the shell's grammar, it cannot be a reserved word since *export* is a  
83042 candidate for alias substitution whereas reserved words are not (see [Section 2.3.1](#)).

### 83043 FUTURE DIRECTIONS

83044 None.

### 83045 SEE ALSO

83046 [Section 2.9.1.1](#), [Section 2.15](#)

83047 [XBD Section 12.2](#)

### 83048 CHANGE HISTORY

#### 83049 Issue 6

83050 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

83051 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
83052 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
83053 behavior is intended.

83054 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/6 is applied, adding the following text to  
83055 the end of the first paragraph of the DESCRIPTION: ``If the name of a variable is followed by  
83056 `=word`, then the value of that variable shall be set to `word`.``. The reason for this change is that the  
83057 SYNOPSIS for *export* includes:

```
83058 export name [=word] . . .
```

83059 but the meaning of the optional `=word` is never explained in the text.

#### 83060 Issue 7

83061 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0043 [352] is applied.

83062 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0051 [654] and XCU/TC2-2008/0052  
83063 [960] are applied.

#### 83064 Issue 8

83065 Austin Group Defect 351 is applied, requiring *export* to be a declaration utility.

83066 Austin Group Defect 367 is applied, changing the EXIT STATUS section.

83067 Austin Group Defect 1258 is applied, changing the EXAMPLES section.

83068  
83069

Austin Group Defect 1393 is applied, changing the APPLICATION USAGE and RATIONALE sections.

83070 **NAME**

83071           readonly — set the readonly attribute for variables

83072 **SYNOPSIS**

83073           readonly name [=word] . . .

83074           readonly -p

83075 **DESCRIPTION**

83076           The variables whose *names* are specified shall be given the *readonly* attribute. The values of  
83077           variables with the *readonly* attribute cannot be changed by subsequent assignment or use of the  
83078           *export*, *getopts*, *readonly*, or *read* utilities, nor can those variables be unset by the *unset* utility. As  
83079           described in XBD [Section 8.1](#), conforming applications shall not request to mark a variable as  
83080           *readonly* if it is documented as being manipulated by a shell built-in utility, as it may render  
83081           those utilities unable to complete successfully. If the name of a variable is followed by *=word*,  
83082           then the value of that variable shall be set to *word*.

83083           The *readonly* special built-in shall be a declaration utility. Therefore, if *readonly* is recognized as  
83084           the command name of a simple command, then subsequent words of the form *name=word* shall  
83085           be expanded in an assignment context. See [Section 2.9.1.1](#).

83086           The *readonly* special built-in shall support XBD [Section 12.2](#).

83087           When *-p* is specified, *readonly* writes to the standard output the names and values of all read-  
83088           only variables, in the following format:

83089           "readonly %s=%s\n", <name>, <value>

83090           if *name* is set, and

83091           "readonly %s\n", <name>

83092           if *name* is unset.

83093           The shell shall format the output, including the proper use of quoting, so that it is suitable for  
83094           reinput to the shell as commands that achieve the same value and *readonly* attribute-setting  
83095           results in a shell execution environment in which:

- 83096           1. Variables with values at the time they were output do not have the *readonly* attribute set.
- 83097           2. Variables that were unset at the time they were output do not have a value at the time at  
83098           which the saved output is reinput to the shell.

83099           When no arguments are given, the results are unspecified.

83100 **OPTIONS**

83101           See the DESCRIPTION.

83102 **OPERANDS**

83103           See the DESCRIPTION.

83104 **STDIN**

83105           Not used.

83106 **INPUT FILES**

83107           None.

83108 **ENVIRONMENT VARIABLES**

83109           None.

83110 **ASYNCHRONOUS EVENTS**

83111 Default.

83112 **STDOUT**

83113 See the DESCRIPTION.

83114 **STDERR**

83115 The standard error shall be used only for diagnostic messages.

83116 **OUTPUT FILES**

83117 None.

83118 **EXTENDED DESCRIPTION**

83119 None.

83120 **EXIT STATUS**

83121 0 Successful completion.

83122 >0 At least one operand could not be processed as requested, such as a *name* operand that  
 83123 could not be marked *readonly* or an attempt to modify an already *readonly* variable using a  
 83124 *name=word* operand, or the **-p** option was specified and a write error occurred.

83125 **CONSEQUENCES OF ERRORS**

83126 Default.

83127 **APPLICATION USAGE**

83128 In shells that support extended assignment syntax, for example to allow an array to be  
 83129 populated with a single assignment, such extensions can typically only be used in assignments  
 83130 specified as arguments to *readonly* if the command word is literally *readonly*, and not if it is some  
 83131 other word that expands to *readonly*. For example:

```
83132 # Shells that support array assignment as an extension generally
83133 # support this:
83134 readonly x=(1 2 3); echo ${x[0]} # outputs 1
83135 # But generally do not support this:
83136 r=readonly; $r x=(1 2 3); echo ${x[0]} # syntax error
```

83137 **EXAMPLES**83138 `readonly HOME`83139 **RATIONALE**

83140 Some historical shells preserve the *readonly* attribute across separate invocations. This volume of  
 83141 POSIX.1-2024 allows this behavior, but does not require it.

83142 The **-p** option allows portable access to the values that can be saved and then later restored  
 83143 using, for example, a *dot* script. Also see the RATIONALE for *export* for a description of the no-  
 83144 argument and **-p** output cases and a related example.

83145 Read-only functions were considered, but they were omitted as not being historical practice or  
 83146 particularly useful. Furthermore, functions must not be read-only across invocations to preclude  
 83147 “spoofing” (spoofing is the term for the practice of creating a program that acts like a well-  
 83148 known utility with the intent of subverting the real intent of the user) of administrative or  
 83149 security-relevant (or security-conscious) shell scripts.

83150 Attempts to set the *readonly* attribute on certain variables, such as *PWD*, may have surprising  
 83151 results. Either *readonly* will reject the attempt, or the attempt will succeed but the shell will  
 83152 continue to alter the contents of *PWD* during the *cd* utility, or the attempt will succeed and  
 83153 render the *cd* utility inoperative (since it must not change directories if it cannot also update

83154 *PWD*).

83155 Some implementations extend the shell's assignment syntax, for example to allow an array to be  
83156 populated with a single assignment, and in order for such an extension to be usable in  
83157 assignments specified as arguments to *readonly* these shells have *readonly* as a separate token in  
83158 their grammar. This standard only permits an extension of this nature when the input to the  
83159 shell would contain a syntax error according to the standard grammar. Note that although  
83160 *readonly* can be a separate token in the shell's grammar, it cannot be a reserved word since  
83161 *readonly* is a candidate for alias substitution whereas reserved words are not (see [Section 2.3.1](#)).

#### 83162 **FUTURE DIRECTIONS**

83163 None.

#### 83164 **SEE ALSO**

83165 [Section 2.9.1.1](#), [Section 2.15](#)

83166 [XBD Section 12.2](#)

#### 83167 **CHANGE HISTORY**

##### 83168 **Issue 6**

83169 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

83170 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
83171 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
83172 behavior is intended.

83173 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/7 is applied, adding the following text to  
83174 the end of the first paragraph of the DESCRIPTION: ``If the name of a variable is followed by  
83175 *=word*, then the value of that variable shall be set to *word*.''. The reason for this change is that the  
83176 SYNOPSIS for *readonly* includes:

```
83177 readonly name [=word] . . .
```

83178 but the meaning of the optional ``*=word*'' is never explained in the text.

##### 83179 **Issue 7**

83180 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0052 [960] is applied.

##### 83181 **Issue 8**

83182 Austin Group Defect 351 is applied, requiring *readonly* to be a declaration utility.

83183 Austin Group Defect 367 is applied, clarifying that the values of *readonly* variables cannot be  
83184 changed by subsequent use of the *export*, *getopts*, *readonly*, or *read* utilities, and changing the EXIT  
83185 STATUS, EXAMPLES and RATIONALE sections.

83186 Austin Group Defect 1393 is applied, changing the APPLICATION USAGE and RATIONALE  
83187 sections.

83188 **NAME**

83189 return — return from a function or dot script

83190 **SYNOPSIS**83191 return [*n*]83192 **DESCRIPTION**83193 The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the  
83194 shell is not currently executing a function or *dot* script, the results are unspecified.83195 **OPTIONS**

83196 None.

83197 **OPERANDS**

83198 See the DESCRIPTION.

83199 **STDIN**

83200 Not used.

83201 **INPUT FILES**

83202 None.

83203 **ENVIRONMENT VARIABLES**

83204 None.

83205 **ASYNCHRONOUS EVENTS**

83206 Default.

83207 **STDOUT**

83208 Not used.

83209 **STDERR**

83210 The standard error shall be used only for diagnostic messages.

83211 **OUTPUT FILES**

83212 None.

83213 **EXTENDED DESCRIPTION**

83214 None.

83215 **EXIT STATUS**83216 The exit status shall be *n*, if specified, except that the behavior is unspecified if *n* is not an  
83217 unsigned decimal integer or is greater than 255. If *n* is not specified, the result shall be as if *n*  
83218 were specified with the current value of the special parameter '?' (see [Section 2.5.2](#)), except that  
83219 if the *return* command would cause the end of execution of a *trap* action, the value for the special  
83220 parameter '?' that is considered ``current'' shall be the value it had immediately preceding the  
83221 *trap* action.83222 **CONSEQUENCES OF ERRORS**

83223 Default.

83224 **APPLICATION USAGE**

83225 None.

83226 **EXAMPLES**

83227 None.

83228 **RATIONALE**

83229 The behavior of *return* when not in a function or *dot* script differs between the System V shell  
83230 and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is  
83231 the same as *exit*.

83232 The results of returning a number greater than 255 are undefined because of differing practices  
83233 in the various historical implementations. Some shells AND out all but the low-order 8 bits;  
83234 others allow larger values, but not of unlimited size.

83235 See the discussion of appropriate exit status values under *exit*.

83236 **FUTURE DIRECTIONS**

83237 None.

83238 **SEE ALSO**83239 [Section 2.9.5](#), [Section 2.15](#), *dot*83240 **CHANGE HISTORY**83241 **Issue 6**

83242 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
83243 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
83244 behavior is intended.

83245 **Issue 7**

83246 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0044 [214] and XCU/TC1-2008/0045  
83247 [214] are applied.

83248 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0052 [960] is applied.

83249 **Issue 8**

83250 Austin Group Defect 1309 is applied, changing the EXIT STATUS section.

83251 Austin Group Defect 1602 is applied, clarifying the behavior of *return* in a *trap* action.



83252 **NAME**

83253 set — set or unset options and positional parameters

83254 **SYNOPSIS**83255 set [-abCefhmnvux] [-o *option*] [*argument...*]83256 set [+abCefhmnvux] [+o *option*] [*argument...*]83257 set -- [*argument...*]

83258 set -o

83259 set +o

83260 **DESCRIPTION**

83261 If no *options* or *arguments* are specified, *set* shall write the names and values of all shell variables  
 83262 in the collation sequence of the current locale. Each *name* shall start on a separate line, using the  
 83263 format:

83264 "%s=%s\n", <*name*>, <*value*>

83265 The *value* string shall be written with appropriate quoting; see the description of shell quoting in  
 83266 [Section 2.2](#). The output shall be suitable for reinput to the shell, setting or resetting, as far as  
 83267 possible, the variables that are currently set; read-only variables cannot be reset.

83268 When options are specified, they shall set or unset attributes of the shell, as described below.  
 83269 When *arguments* are specified, they cause positional parameters to be set or unset, as described  
 83270 below. Setting or unsetting attributes and positional parameters are not necessarily related  
 83271 actions, but they can be combined in a single invocation of *set*.

83272 The *set* special built-in shall support XBD [Section 12.2](#) except that options can be specified with  
 83273 either a leading <hyphen-minus> (meaning enable the option) or <plus-sign> (meaning disable  
 83274 it) unless otherwise specified.

83275 Implementations shall support the options in the following list in both their <hyphen-minus>  
 83276 and <plus-sign> forms. These options can also be specified as options to *sh*.

83277 **-a** Set the *export* attribute for all variable assignments. When this option is on, whenever a  
 83278 value is assigned to a variable in the current shell execution environment, the *export*  
 83279 attribute shall be set for the variable. This applies to all forms of assignment, including  
 83280 those made as a side-effect of variable expansions or arithmetic expansions, and those made  
 83281 as a result of the operation of the *cd*, *getopts*, or *read* utilities.

83282 **Note:** As discussed in [Section 2.9.1](#), not all variable assignments happen in the current execution  
 83283 environment. When an assignment happens in a separate execution environment the  
 83284 *export* attribute is still set for the variable, but that does not affect the current execution  
 83285 environment.

83286 **-b** This option shall be supported if the implementation supports the User Portability Utilities  
 83287 option. When job control and **-b** are both enabled, the shell shall write asynchronous  
 83288 notifications of background job completions (including termination by a signal), and may  
 83289 write asynchronous notifications of background job suspensions. See [Section 2.11](#) for  
 83290 details. When job control is disabled, the **-b** option shall have no effect. Asynchronous  
 83291 notification shall not be enabled by default.

83292 **-C** (Uppercase C.) Prevent existing regular files from being overwritten by the shell's '>'   
 83293 redirection operator (see [Section 2.7.2](#)); the '>|' redirection operator shall override this  
 83294 *noclobber* option for an individual file.

83295 **-e** When this option is on, when any command fails (for any of the reasons listed in [Section](#)  
 83296 [2.8.1](#) or by returning an exit status greater than zero), the shell immediately shall exit, as if  
 83297 by executing the *exit* special built-in utility with no arguments, with the following  
 83298 exceptions:

- 83299 1. The failure of any individual command in a multi-command pipeline, or of any  
 83300 subshell environments in which command substitution was performed during word  
 83301 expansion, shall not cause the shell to exit. Only the failure of the pipeline itself shall  
 83302 be considered.
- 83303 2. The **-e** setting shall be ignored when executing the compound list following the  
 83304 **while**, **until**, **if**, or **elif** reserved word, a pipeline beginning with the **!** reserved word,  
 83305 or any command of an AND-OR list other than the last.
- 83306 3. If the exit status of a compound command other than a subshell command was the  
 83307 result of a failure while **-e** was being ignored, then **-e** shall not apply to this  
 83308 command.

83309 This requirement applies to the shell environment and each subshell environment  
 83310 separately. For example, in:

```
83311 set -e; (false; echo one) | cat; echo two
```

83312 the *false* command causes the subshell to exit without executing *echo one*; however, *echo*  
 83313 *two* is executed because the exit status of the pipeline `(false; echo one) | cat` is  
 83314 zero.

83315 In

```
83316 set -e; echo $(false; echo one) two
```

83317 the *false* command causes the subshell in which the command substitution is performed to  
 83318 exit without executing *echo one*; the exit status of the subshell is ignored and the shell  
 83319 then executes the word-expanded command *echo two*.

83320 **-f** The shell shall disable pathname expansion.

83321 **-h** Setting this option may speed up *PATH* searches (see [XBD Chapter 8](#)). This option may be  
 83322 enabled by default.

83323 **-m** This option shall be supported if the implementation supports the User Portability Utilities  
 83324 option. When this option is enabled, the shell shall perform job control actions as described  
 83325 in [Section 2.11](#). This option shall be enabled by default for interactive shells.

83326 **-n** The shell shall read commands but does not execute them; this can be used to check for  
 83327 shell script syntax errors. Interactive shells and subshells of interactive shells, recursively,  
 83328 may ignore this option.

83329 **-o** Write the current settings of the options to standard output in an unspecified format.

83330 **+o** Write the current option settings to standard output in a format that is suitable for reinput  
 83331 to the shell as commands that achieve the same options settings.

83332 **-o option**

83333 Set various options, many of which shall be equivalent to the single option letters. The  
 83334 following values of *option* shall be supported:

83335 *allexport* Equivalent to **-a**.

83336	<i>errexist</i>	Equivalent to <b>-e</b> .
83337	<i>ignoreeof</i>	Prevent an interactive shell from exiting on end-of-file. This setting prevents accidental logouts when <control>-D is entered. A user shall explicitly <i>exit</i> to leave the interactive shell. This option shall be supported if the system supports the User Portability Utilities option.
83338		
83339		
83340		
83341	<i>monitor</i>	Equivalent to <b>-m</b> . This option shall be supported if the system supports the User Portability Utilities option.
83342		
83343	<i>noclobber</i>	Equivalent to <b>-C</b> (uppercase C).
83344	<i>noglob</i>	Equivalent to <b>-f</b> .
83345	<i>noexec</i>	Equivalent to <b>-n</b> .
83346	OB <i>nolog</i>	Prevent the entry of function definitions into the command history; see <a href="#">Command History List</a> . This option may have no effect; it is kept for compatibility with previous versions of the standard. This option shall be supported if the system supports the User Portability Utilities option.
83347		
83348		
83349		
83350	<i>notify</i>	Equivalent to <b>-b</b> .
83351	<i>nounset</i>	Equivalent to <b>-u</b> .
83352	<i>pipefail</i>	Derive the exit status of a pipeline from the exit statuses of all of the commands in the pipeline, not just the last (rightmost) command, as described in <a href="#">Section 2.9.2</a> .
83353		
83354		
83355	<i>verbose</i>	Equivalent to <b>-v</b> .
83356	<i>vi</i>	Allow shell command line editing using the built-in <i>vi</i> editor. Enabling <i>vi</i> mode shall disable any other command line editing mode provided as an implementation extension. This option shall be supported if the system supports the User Portability Utilities option.
83357		
83358		
83359		
83360		It need not be possible to set <i>vi</i> mode on for certain block-mode terminals.
83361	<i>xtrace</i>	Equivalent to <b>-x</b> .
83362	<b>-u</b>	When the shell tries to expand, in a parameter expansion or an arithmetic expansion, an unset parameter other than the '@' and '*' special parameters, it shall write a message to standard error and the expansion shall fail with the consequences specified in <a href="#">Section 2.8.1</a> .
83363		
83364		
83365	<b>-v</b>	The shell shall write its input to standard error as it is read.
83366	<b>-x</b>	The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.
83367		
83368		
83369		The default for all these options shall be off (unset) unless stated otherwise in the description of the option or unless the shell was invoked with them on; see <i>sh</i> .
83370		
83371		The remaining arguments shall be assigned in order to the positional parameters. The special parameter '#' shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.
83372		
83373		
83374		If the first argument is '-', the results are unspecified.
83375		The special argument "--" immediately following the <i>set</i> command name can be used to delimit the arguments if the first argument begins with '+' or '-', or to prevent inadvertent listing of all shell variables when there are no arguments. The command <i>set --</i> without <i>argument</i>
83376		
83377		

83378 shall unset all positional parameters and set the special parameter '# ' to zero.

83379 **OPTIONS**

83380 See the DESCRIPTION.

83381 **OPERANDS**

83382 See the DESCRIPTION.

83383 **STDIN**

83384 Not used.

83385 **INPUT FILES**

83386 None.

83387 **ENVIRONMENT VARIABLES**

83388 None.

83389 **ASYNCHRONOUS EVENTS**

83390 Default.

83391 **STDOUT**

83392 See the DESCRIPTION.

83393 **STDERR**

83394 The standard error shall be used only for diagnostic messages.

83395 **OUTPUT FILES**

83396 None.

83397 **EXTENDED DESCRIPTION**

83398 None.

83399 **EXIT STATUS**

83400 0 Successful completion.

83401 >0 An invalid option was specified, or an error occurred.

83402 **CONSEQUENCES OF ERRORS**

83403 Default.

83404 **APPLICATION USAGE**

83405 Application writers should avoid relying on *set -e* within functions. For example, in the  
83406 following script:

```
83407 set -e
83408 start () {
83409     some_server
83410     echo some_server started successfully
83411 }
83412 start || echo >&2 some_server failed
```

83413 the *-e* setting is ignored within the function body (because the function is a command in an  
83414 AND-OR list other than the last). Therefore, if *some\_server* fails, the function carries on to  
83415 echo "some\_server started successfully", and the exit status of the function is zero  
83416 (which means "some\_server failed" is not output).

83417 Use of *set -n* causes the shell to parse the rest of the script without executing any commands,  
83418 meaning that *set +n* cannot be used to undo the effect. Syntax checking is more commonly done  
83419 via *sh -n script\_name*.

83420 **EXAMPLES**

83421 Write out all variables and their values:

83422 `set`

83423 Set \$1, \$2, and \$3 and set "\$#" to 3:

83424 `set c a b`83425 Turn on the `-x` and `-v` options:83426 `set -xv`

83427 Unset all positional parameters:

83428 `set --`83429 Set \$1 to the value of *x*, even if it begins with '-' or '+':83430 `set -- "$x"`83431 Set the positional parameters to the expansion of *x*, even if *x* expands with a leading '-' or '+':83432 `set -- $x`83433 **RATIONALE**

83434 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the  
 83435 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether  
 83436 the `set --` form might be misinterpreted as being equivalent to `set` without any options or  
 83437 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`  
 83438 `--` only unsets parameters if there is at least one argument; the only way to unset all parameters  
 83439 is to use *shift*. Using the KornShell version should not affect System V scripts because there  
 83440 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

83441 `set -- "$@"`

83442 and there were in fact no arguments resulting from "\$@", unsetting the parameters would have  
 83443 no result.

83444 The `set +` form in early proposals was omitted as being an unnecessary duplication of `set` alone  
 83445 and not widespread historical practice.

83446 The *noclobber* option was changed to allow `set -C` as well as the `set -o noclobber` option. The  
 83447 single-letter version was added so that the historical "\$-" paradigm would not be broken; see  
 83448 [Section 2.5.2](#).

83449 The description of the `-e` option is intended to match the behavior of the 1988 version of the  
 83450 KornShell.

83451 The `-h` option is related to command name hashing. See *hash*. The normative description is  
 83452 deliberately vague because the way this option works varies between shell implementations.

83453 Earlier versions of this standard specified `-h` as a way to locate and remember utilities to be  
 83454 invoked by functions as those functions are defined (the utilities are normally located when the  
 83455 function is executed). However, this did not match existing practice in most shells.

83456 The following `set` options were omitted intentionally with the following rationale:

83457 **-k** The `-k` option was originally added by the author of the Bourne shell to make it easier for  
 83458 users of pre-release versions of the shell. In early versions of the Bourne shell the construct  
 83459 `set name=value` had to be used to assign values to shell variables. The problem with `-k` is  
 83460 that the behavior affects parsing, virtually precluding writing any compilers. To explain the

83461 behavior of `-k`, it is necessary to describe the parsing algorithm, which is implementation-  
83462 defined. For example:

```
83463 set -k; echo name=value
```

83464 and:

```
83465 set -k  
83466 echo name=value
```

83467 behave differently. The interaction with functions is even more complex. What is more, the  
83468 `-k` option is never needed, since the command line could have been reordered.

83469 `-t` The `-t` option is hard to specify and almost never used. The only known use could be done  
83470 with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page  
83471 says that it exits after reading and executing one command. What is one command? If the  
83472 input is *date;date*, *sh* executes both *date* commands while *ksh* does only the first.

83473 Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion  
83474 was that the *unset* utility should be used to unset options instead of using the non-*getopt*(-)-able  
83475 `+option` syntax. However, the conclusion was reached that the historical practice of using `+option`  
83476 was satisfactory and that there was no compelling reason to modify such widespread historical  
83477 practice.

83478 The `-o` option was adopted from the KornShell to address user needs. In addition to its  
83479 generally friendly interface, `-o` is needed to provide the *vi* command line editing mode, for  
83480 which historical practice yields no single-letter option name. (Although it might have been  
83481 possible to invent such a letter, it was recognized that other editing modes would be developed  
83482 and `-o` provides ample name space for describing such extensions.)

83483 Historical implementations are inconsistent in the format used for `-o` option status reporting.  
83484 The `+o` format without an option-argument was added to allow portable access to the options  
83485 that can be saved and then later restored using, for instance, a dot script.

83486 Historically, *sh* did trace the command `set +x`, but *ksh* did not.

83487 The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically  
83488 `<control>-D`) is entered. A user shall explicitly *exit* to leave the interactive shell.

83489 The `set -m` option was added to apply only to the UPE because it applies primarily to interactive  
83490 use, not shell script applications.

83491 The ability to do asynchronous notification became available in the 1988 version of the  
83492 KornShell. To have it occur, the user had to issue the command:

```
83493 trap "jobs -n" CLD
```

83494 The C shell provides two different levels of an asynchronous notification capability. The  
83495 environment variable *notify* is analogous to what is done in `set -b` or `set -o notify`. When set, it  
83496 notifies the user immediately of background job completions. When unset, this capability is  
83497 turned off.

83498 The other notification ability comes through the built-in utility *notify*. The syntax is:

```
83499 notify [%job ... ]
```

83500 By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when  
83501 the state of the current job changes. If given operands, *notify* asynchronously informs the user of  
83502 changes in the states of the specified jobs.

83503 To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,

83504 nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX  
83505 environment variable name).

83506 The *set -b* option was selected as a compromise.

83507 The *notify* built-in was considered to have more functionality than was required for simple  
83508 asynchronous notification.

83509 Historically, some shells applied the *-u* option to all parameters including *\$@* and *\$\**. The  
83510 standard developers felt that this was a misfeature since it is normal and common for *\$@* and *\$\**  
83511 to be used in shell scripts regardless of whether they were passed any arguments. Treating these  
83512 uses as an error when no arguments are passed reduces the value of *-u* for its intended purpose  
83513 of finding spelling mistakes in variable names and uses of unset positional parameters.

#### 83514 FUTURE DIRECTIONS

83515 A future version of this standard may remove the *-o nolog* option.

#### 83516 SEE ALSO

83517 [Section 2.15, \*hash\*](#)

83518 [XBD Section 4.26, Section 12.2](#)

#### 83519 CHANGE HISTORY

##### 83520 Issue 6

83521 The obsolescent *set* command name followed by *'-'* has been removed.

83522 The following new requirements on POSIX implementations derive from alignment with the  
83523 Single UNIX Specification:

- 83524 • The *nolog* option is added to *set -o*.

83525 IEEE PASC Interpretation 1003.2 #167 is applied, clarifying that the options default also takes  
83526 into account the description of the option.

83527 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
83528 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
83529 behavior is intended.

83530 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/8 is applied, changing the square  
83531 brackets in the example in RATIONALE to be in bold, which is the typeface used for optional  
83532 items.

##### 83533 Issue 7

83534 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first  
83535 argument is *'-'*.

83536 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83537 XSI shading is removed from the *-h* functionality.

83538 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0046 [52], XCU/TC1-2008/0047  
83539 [155,280], XCU/TC1-2008/0048 [52], XCU/TC1-2008/0049 [52], and XCU/TC1-2008/0050  
83540 [155,430] are applied.

83541 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0053 [584], XCU/TC2-2008/0054  
83542 [717], XCU/TC2-2008/0055 [717], and XCU/TC2-2008/0056 [960] are applied.

83543 **Issue 8**

- 83544 Austin Group Defect 559 is applied, changing the description of the `-u` option.
- 83545 Austin Group Defect 789 is applied, adding `-o pipefail`.
- 83546 Austin Group Defect 981 is applied, changing the description of the `-o nolog` option and the  
83547 FUTURE DIRECTIONS section.
- 83548 Austin Group Defects 1009 and 1555 are applied, changing the description of the `-a` option.
- 83549 Austin Group Defect 1016 is applied, changing the description of the `-C` option.
- 83550 Austin Group Defect 1055 is applied, adding a paragraph about the `-n` option to the  
83551 APPLICATION USAGE section.
- 83552 Austin Group Defect 1063 is applied, changing the description of the `-h` option.
- 83553 Austin Group Defect 1150 is applied, changing the description of the `-e` option.
- 83554 Austin Group Defect 1207 is applied, clarifying which option-arguments of the `-o` option are  
83555 related to the User Portability Utilities option.
- 83556 Austin Group Defect 1254 is applied, changing the descriptions of the `-b` and `-m` options.
- 83557 Austin Group Defect 1384 is applied, allowing subshells of interactive shells to ignore the `-n`  
83558 option.



**83559 NAME**

83560 shift — shift positional parameters

**83561 SYNOPSIS**

83562 shift [*n*]

**83563 DESCRIPTION**

83564 The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of  
83565 parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The  
83566 parameters represented by the numbers "\$#" down to "\$#-*n*+1" shall be unset, and the  
83567 parameter '#' is updated to reflect the new number of positional parameters.

83568 The value *n* shall be an unsigned decimal integer less than or equal to the value of the special  
83569 parameter '#'. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special  
83570 parameters are not changed.

**83571 OPTIONS**

83572 None.

**83573 OPERANDS**

83574 See the DESCRIPTION.

**83575 STDIN**

83576 Not used.

**83577 INPUT FILES**

83578 None.

**83579 ENVIRONMENT VARIABLES**

83580 None.

**83581 ASYNCHRONOUS EVENTS**

83582 Default.

**83583 STDOUT**

83584 Not used.

**83585 STDERR**

83586 The standard error shall be used only for diagnostic messages and the warning message  
83587 specified in EXIT STATUS.

**83588 OUTPUT FILES**

83589 None.

**83590 EXTENDED DESCRIPTION**

83591 None.

**83592 EXIT STATUS**

83593 If the *n* operand is invalid or is greater than "\$#", this may be treated as an error and a non-  
83594 interactive shell may exit; if the shell does not exit in this case, a non-zero exit status shall be  
83595 returned and a warning message shall be written to standard error. Otherwise, zero shall be  
83596 returned.

**83597 CONSEQUENCES OF ERRORS**

83598 Default.

83599 **APPLICATION USAGE**

83600 None.

83601 **EXAMPLES**83602 `$ set a b c d e`83603 `$ shift 2`83604 `$ echo $*`83605 `c d e`83606 **RATIONALE**

83607 None.

83608 **FUTURE DIRECTIONS**

83609 None.

83610 **SEE ALSO**83611 [Section 2.15](#)83612 **CHANGE HISTORY**83613 **Issue 6**

83614 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
83615 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
83616 behavior is intended.

83617 **Issue 7**

83618 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0051 [459] is applied.

83619 **Issue 8**

83620 Austin Group Defect 1265 is applied, updating the EXIT STATUS and STDERR sections to align  
83621 with the changes made to [Section 2.8.1](#) between Issue 6 and Issue 7.

83622 **NAME**

83623 times — write process times

83624 **SYNOPSIS**

83625 times

83626 **DESCRIPTION**83627 The *times* utility shall write the accumulated user and system times for the shell and for all of its  
83628 child processes, in the following POSIX locale format:83629 "%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,  
83630 <shell user seconds>, <shell system minutes>,  
83631 <shell system seconds>, <children user minutes>,  
83632 <children user seconds>, <children system minutes>,  
83633 <children system seconds>83634 The four pairs of times shall correspond to the members of the <sys/times.h> **tms** structure  
83635 (defined in XBD [Chapter 14](#)) as returned by *times()*: *tms\_utime*, *tms\_stime*, *tms\_cutime*, and  
83636 *tms\_cstime*, respectively.83637 **OPTIONS**

83638 None.

83639 **OPERANDS**

83640 None.

83641 **STDIN**

83642 Not used.

83643 **INPUT FILES**

83644 None.

83645 **ENVIRONMENT VARIABLES**

83646 None.

83647 **ASYNCHRONOUS EVENTS**

83648 Default.

83649 **STDOUT**

83650 See the DESCRIPTION.

83651 **STDERR**

83652 The standard error shall be used only for diagnostic messages.

83653 **OUTPUT FILES**

83654 None.

83655 **EXTENDED DESCRIPTION**

83656 None.

83657 **EXIT STATUS**

83658 0 Successful completion.

83659 &gt;0 An error occurred.

83660 **CONSEQUENCES OF ERRORS**

83661 Default.

83662 **APPLICATION USAGE**

83663 None.

83664 **EXAMPLES**83665 `$ times`83666 `0m0.43s 0m1.11s`83667 `8m44.18s 1m43.23s`83668 **RATIONALE**83669 The *times* special built-in from the Single UNIX Specification is now required for all conforming  
83670 shells.83671 **FUTURE DIRECTIONS**

83672 None.

83673 **SEE ALSO**83674 [Section 2.15](#)83675 XBD [<sys/times.h>](#)83676 **CHANGE HISTORY**83677 **Issue 6**83678 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/9 is applied, changing text in the  
83679 DESCRIPTION from: "Write the accumulated user and system times for the shell and for all of  
83680 its child processes ..." to: "The *times* utility shall write the accumulated user and system times for  
83681 the shell and for all of its child processes ...".83682 **Issue 7**

83683 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0056 [960] is applied.

83684 **NAME**

83685 trap — trap signals

83686 **SYNOPSIS**83687 trap *n* [*condition...*]83688 trap -p [*condition...*]83689 trap [*action condition...*]83690 **DESCRIPTION**

83691 If the **-p** option is not specified and the first operand is an unsigned decimal integer, the shell  
 83692 shall treat all operands as conditions, and shall reset each condition to the default value.  
 83693 Otherwise, if the **-p** option is not specified and there are operands, the first operand shall be  
 83694 treated as an action and the remaining as conditions.

83695 If *action* is '-', the shell shall reset each *condition* to the default value. If *action* is null (""), the  
 83696 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read  
 83697 and executed by the shell when one of the corresponding conditions arises. The action of *trap*  
 83698 shall override a previous action (either default action or one explicitly set). The value of "\$?"  
 83699 after the *trap* action completes shall be the value it had before the *trap* action was executed.

83700 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,  
 83701 without the SIG prefix, as listed in the tables of signal names in the <signal.h> header defined in  
 83702 XBD Chapter 14; for example, HUP, INT, QUIT, TERM. Implementations may permit names  
 83703 with the SIG prefix or ignore case in signal names as an extension. Setting a trap for SIGKILL or  
 83704 SIGSTOP produces undefined results.

83705 The EXIT condition shall occur when the shell terminates normally (exits), and may occur when  
 83706 the shell terminates abnormally as a result of delivery of a signal (other than SIGKILL) whose  
 83707 *trap* action is the default.

83708 The environment in which the shell executes a *trap* action on EXIT shall be identical to the  
 83709 environment immediately after the last command executed before the *trap* action on EXIT was  
 83710 executed.

83711 If *action* is neither '-' nor the empty string, then each time a matching *condition* arises, the *action*  
 83712 shall be executed in a manner equivalent to:

83713 eval *action*

83714 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although  
 83715 no error need be reported when attempting to do so. An interactive shell may reset or catch  
 83716 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed  
 83717 with another *trap* command.

83718 When a subshell is entered, traps that are not being ignored shall be set to the default actions,  
 83719 except in the case of a command substitution containing only a single *trap* command, when the  
 83720 traps need not be altered. Implementations may check for this case using only lexical analysis;  
 83721 for example, if `trap` and \$( trap -- ) do not alter the traps in the subshell, cases such as  
 83722 assigning var=trap and then using \$( \$var ) may still alter them. This does not imply that the  
 83723 *trap* command cannot be used within the subshell to set new traps.

83724 The *trap* command with no operands shall write to standard output a list of commands  
 83725 associated with each of a set of conditions; if the **-p** option is not specified, this set shall contain  
 83726 only the conditions that are not in the default state (including signals that were ignored on entry  
 83727 to a non-interactive shell); if the **-p** option is specified, the set shall contain all conditions, except  
 83728 that it is unspecified whether conditions corresponding to the SIGKILL and SIGSTOP signals are  
 83729 included in the set. If the command is executed in a subshell, the implementation does not

83730 perform the optional check described above for a command substitution containing only a single  
 83731 *trap* command, and no *trap* commands with operands have been executed since entry to the  
 83732 subshell, the list shall contain the commands that were associated with each condition  
 83733 immediately before the subshell environment was entered. Otherwise, the list shall contain the  
 83734 commands currently associated with each condition. The format shall be:

```
83735 "trap -- %s %s ...\n", <action>, <condition> ...
```

83736 The shell shall format the output, including the proper use of quoting, so that it is suitable for  
 83737 reinput to the shell as commands that achieve the same trapping results for the set of conditions  
 83738 included in the output, except for signals that were ignored on entry to the shell as described  
 83739 above. If this set includes conditions corresponding to the SIGKILL and SIGSTOP signals, the  
 83740 shell shall accept them when the output is reinput to the shell (where accepting them means  
 83741 they do not cause a non-zero exit status, a diagnostic message, or undefined behavior). For  
 83742 example:

```
83743 save_traps=$(trap -p)
```

```
83744 ...
83745 eval "$save_traps"
```

83746 or:

```
83747 save_traps=$(trap -p INT QUIT)
83748 trap "some command" INT QUIT
```

```
83749 ...
83750 eval "$save_traps"
```

83751 XSI XSI-conformant systems also allow numeric signal numbers for the conditions corresponding to  
 83752 the following signal names:

```
83753 1  SIGHUP
83754 2  SIGINT
83755 3  SIGQUIT
83756 6  SIGABRT
83757 9  SIGKILL
83758 14 SIGALRM
83759 15 SIGTERM
```

83760 XSI If an invalid signal name **or number** is specified, the *trap* utility shall write a warning message  
 83761 to standard error.

83762 The *trap* special built-in shall conform to XBD [Section 12.2](#).

## 83763 OPTIONS

83764 The following option shall be supported:

83765 **-p** Write to standard output a list of commands associated with each *condition* operand. The  
 83766 behavior when there are no operands is specified in the DESCRIPTION section.

83767 The shell shall format the output, including the proper use of quoting, so that it is suitable  
 83768 for reinput to the shell as commands that achieve the same trapping results for the specified  
 83769 set of conditions. If a *condition* operand is a condition corresponding to the SIGKILL or

83770 SIGSTOP signal, and *trap -p* without any operands would not include it in the set of  
 83771 conditions for which it writes output, the behavior is undefined if the output is reinput to  
 83772 the shell.

#### 83773 OPERANDS

83774 See the DESCRIPTION.

#### 83775 STDIN

83776 Not used.

#### 83777 INPUT FILES

83778 None.

#### 83779 ENVIRONMENT VARIABLES

83780 None.

#### 83781 ASYNCHRONOUS EVENTS

83782 Default.

#### 83783 STDOUT

83784 See the DESCRIPTION.

#### 83785 STDERR

83786 The standard error shall be used only for diagnostic messages and warning messages about  
 83787 XSI invalid signal names or numbers.

#### 83788 OUTPUT FILES

83789 None.

#### 83790 EXTENDED DESCRIPTION

83791 None.

#### 83792 EXIT STATUS

83793 XSI If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero  
 83794 XSI shall be returned. For both interactive and non-interactive shells, invalid signal names or  
 83795 numbers shall not be considered an error and shall not cause the shell to abort.

#### 83796 CONSEQUENCES OF ERRORS

83797 Default.

#### 83798 APPLICATION USAGE

83799 When the *-p* option is not used, since *trap* with no operands does not output commands to  
 83800 restore traps that are currently set to default, these need to be restored separately. The  
 83801 RATIONALE section shows examples and describes their drawbacks.

#### 83802 EXAMPLES

83803 Write out a list of all traps and actions:

83804 `trap`

83805 Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable  
 83806 executes when the shell terminates:

83807 `trap '$HOME'/logout' EXIT`

83808 or:

83809 `trap '$HOME'/logout' 0`

83810 Unset traps on INT, QUIT, TERM, and EXIT:

83811 `trap - INT QUIT TERM EXIT`

83812 **RATIONALE**

83813 Implementations may permit lowercase signal names as an extension. Implementations may  
 83814 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*  
 83815 utilities in this volume of POSIX.1-2024 are now consistent in their omission of the SIG prefix for  
 83816 signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the signals  
 83817 without prefixes.

83818 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but  
 83819 it has no effect. Portable POSIX applications cannot attempt to trap these signals.

83820 The output format is not historical practice. Since the output of historical *trap* commands is not  
 83821 portable (because numeric signal values are not portable) and had to change to become so, an  
 83822 opportunity was taken to format the output in a way that a shell script could use to save and  
 83823 then later reuse a trap if it wanted.

83824 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is  
 83825 allowable as an extension, but was not mandated, as other shells have not used it.

83826 The text about the environment for the EXIT trap invalidates the behavior of some historical  
 83827 versions of interactive shells which, for example, close the standard input before executing a  
 83828 trap on 0. For example, in some historical interactive shell sessions the following trap on 0  
 83829 would always print "--":

```
83830 trap 'read foo; echo "--$foo--"' 0
```

83831 The command:

```
83832 trap 'eval " $cmd"' 0
```

83833 causes the contents of the shell variable *cmd* to be executed as a command when the shell exits.  
 83834 Using:

```
83835 trap '$cmd' 0
```

83836 does not work correctly if *cmd* contains any special characters such as quoting or redirections.  
 83837 Using:

```
83838 trap " $cmd" 0
```

83839 also works (the leading <space> character protects against unlikely cases where *cmd* is a decimal  
 83840 integer or begins with '-'), but it expands the *cmd* variable when the *trap* command is executed,  
 83841 not when the exit action is executed.

83842 The **-p** option was added because without it the method used to restore traps needs to include  
 83843 special handling of traps that are set to default when *trap* with no operands is used to save the  
 83844 current traps. One example is:

```
83845 save_traps=$(trap)
83846 trap "some command" INT QUIT
83847 save_traps="trap - INT QUIT; $save_traps"
```

```
83848 ...
```

```
83849 eval "$save_traps"
```

83850 but this method relies on hard-coding the commands to reset the traps that are being set. It also  
 83851 has a race condition if INT or QUIT was not set to default when saved, since it first sets them to  
 83852 default and then restores the saved traps. A more general approach would be:

```
83853 save_traps=$(trap)
83854 for sig in EXIT $( kill -l )
```



```

83855     do
83856         case "$sig" in
83857             SIGKILL | KILL | sigkill | kill | SIGSTOP | STOP | sigstop | stop)
83858                 ;;
83859             *) trap - $sig
83860                 ;;
83861             esac
83862     done
83863     eval "$save_traps"

```

83864 This has the same race condition since it first sets all traps (that can be set) to default and then  
83865 restores those that were not previously set to default.

83866 Historically, some shells behaved the same with and without `-p` when there are no operands.  
83867 This standard requires that the set of conditions differs between the two cases: with `-p` it is all  
83868 conditions (except possibly SIGKILL and SIGSTOP); without `-p` it is only the conditions that are  
83869 not in the default state.

#### 83870 FUTURE DIRECTIONS

83871 None.

#### 83872 SEE ALSO

83873 [Section 2.15](#)

83874 [XBD Section 12.2, <signal.h>](#)

#### 83875 CHANGE HISTORY

##### 83876 Issue 6

83877 XSI-conforming implementations provide the mapping of signal names to numbers given above  
83878 (previously this had been marked obsolescent). Other implementations need not provide this  
83879 optional mapping.

83880 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
83881 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
83882 behavior is intended.

##### 83883 Issue 7

83884 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83885 Austin Group Interpretation 1003.1-2001 #116 is applied.

83886 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0052 [53,268,440],  
83887 XCU/TC1-2008/0053 [53,268,440], XCU/TC1-2008/0054 [163], XCU/TC1-2008/0055 [163], and  
83888 XCU/TC1-2008/0056 [163] are applied.

##### 83889 Issue 8

83890 Austin Group Defect 621 is applied, clarifying when the EXIT condition occurs.

83891 Austin Group Defect 1029 is applied, clarifying the execution of *trap* actions.

83892 Austin Group Defects 1211 and 1212 are applied, adding the `-p` option and clarifying that, when  
83893 `-p` is not specified, the output of *trap* with no operands does not list conditions that are in the  
83894 default state.

83895 Austin Group Defect 1265 is applied, updating the DESCRIPTION, STDERR and EXIT STATUS  
83896 sections to align with the changes made to [Section 2.8.1](#) between Issue 6 and Issue 7.

83897

Austin Group Defect 1285 is applied, inserting a blank line between the two SYNOPSIS lines.

83898 **NAME**

83899           unset — unset values and attributes of variables and functions

83900 **SYNOPSIS**

83901           unset [-fv] name...

83902 **DESCRIPTION**

83903           The *unset* utility shall unset each variable or function definition specified by *name* that does not  
83904           have the *readonly* attribute and remove any attributes other than *readonly* that have been given to  
83905           *name* (see [Section 2.15](#) *export* and *readonly*).

83906           If *-v* is specified, *name* refers to a variable name and the shell shall unset it and remove it from  
83907           the environment. Read-only variables cannot be unset.

83908           If *-f* is specified, *name* refers to a function and the shell shall unset the function definition.

83909           If neither *-f* nor *-v* is specified, *name* refers to a variable; if a variable by that name does not  
83910           exist, it is unspecified whether a function by that name, if any, shall be unset.

83911           Unsetting a variable or function that was not previously set shall not be considered an error and  
83912           does not cause the shell to abort.

83913           The *unset* special built-in shall support XBD [Section 12.2](#).

83914           Note that:

83915           VARIABLE=

83916           is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to " ". Also, the  
83917           variables that can be *unset* should not be misinterpreted to include the special parameters (see  
83918           [Section 2.5.2](#)).

83919 **OPTIONS**

83920           See the DESCRIPTION.

83921 **OPERANDS**

83922           See the DESCRIPTION.

83923 **STDIN**

83924           Not used.

83925 **INPUT FILES**

83926           None.

83927 **ENVIRONMENT VARIABLES**

83928           None.

83929 **ASYNCHRONOUS EVENTS**

83930           Default.

83931 **STDOUT**

83932           Not used.

83933 **STDERR**

83934           The standard error shall be used only for diagnostic messages.

83935 **OUTPUT FILES**

83936           None.

83937 **EXTENDED DESCRIPTION**

83938 None.

83939 **EXIT STATUS**83940 0 All *name* operands were successfully unset.83941 >0 At least one *name* could not be unset.83942 **CONSEQUENCES OF ERRORS**

83943 Default.

83944 **APPLICATION USAGE**

83945 None.

83946 **EXAMPLES**83947 Unset *VISUAL* variable:83948 `unset -v VISUAL`83949 Unset the functions **foo** and **bar**:83950 `unset -f foo bar`83951 **RATIONALE**83952 Consideration was given to omitting the `-f` option in favor of an *unfunction* utility, but the  
83953 standard developers decided to retain historical practice.83954 The `-v` option was introduced because System V historically used one name space for both  
83955 variables and functions. When *unset* is used without options, System V historically unset either a  
83956 function or a variable, and there was no confusion about which one was intended. A portable  
83957 POSIX application can use *unset* without an option to unset a variable, but not a function; the `-f`  
83958 option must be used.83959 **FUTURE DIRECTIONS**

83960 None.

83961 **SEE ALSO**83962 [Section 2.15](#)83963 [XBD Section 12.2](#)83964 **CHANGE HISTORY**83965 **Issue 6**83966 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
83967 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
83968 behavior is intended.83969 **Issue 7**


83970 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83971 **Issue 8**83972 Austin Group Defect 1075 is applied, clarifying that *unset* removes attributes, other than  
83973 *readonly*, from the variables it unsets.

83974

Chapter 3

83975



# Utilities

83976

This chapter contains the definitions of the utilities, as follows:

83977

- Mandatory utilities that are present on every conformant system

83978

- Optional utilities that are present only on systems supporting the associated option; see [Section 1.8.1](#) (on page 7) for information on the options in this volume of POSIX.1-2024

83979

## 83980 NAME

83981 admin — create and administer SCCS files (DEVELOPMENT)

## 83982 SYNOPSIS

```

83983 XSI admin -i[name] [-n] [-a login] [-d flag] [-e login] [-f flag]
83984 [-m mrlist] [-r rel] [-t[name] [-y[comment]]] newfile
83985 admin -n [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
83986 [-t[name]] [-y[comment]] newfile...
83987 admin [-a login] [-d flag] [-m mrlist] [-r rel] [-t[name]] file...
83988 admin -h file...
83989 admin -z file...

```

## 83990 DESCRIPTION

83991 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named  
 83992 file does not exist, it shall be created, and its parameters shall be initialized according to the  
 83993 specified options. Parameters not initialized by an option shall be assigned a default value. If a  
 83994 named file does exist, parameters corresponding to specified options shall be changed, and other  
 83995 parameters shall be left as is.

83996 All SCCS filenames supplied by the application shall be of the form *s.filename*. New SCCS files  
 83997 shall be given read-only permission mode. Write permission in the parent directory is required  
 83998 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)  
 83999 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode  
 84000 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file  
 84001 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This  
 84002 ensures that changes are made to the SCCS file only if no errors occur.

84003 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent  
 84004 simultaneous updates to the SCCS file; see *get*.

## 84005 OPTIONS

84006 The *admin* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the *-i*, *-t*, and *-y*  
 84007 options have optional option-arguments. These optional option-arguments shall not be  
 84008 presented as separate arguments. The following options are supported:

84009 **-n** Create a new SCCS file. When *-n* is used without *-i*, the SCCS file shall be created  
 84010 with control information but without any file data.

84011 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.  
 84012 The text constitutes the first delta of the file (see the *-r* option for the delta  
 84013 numbering scheme). If the *-i* option is used, but the *name* option-argument is  
 84014 omitted, the text shall be obtained by reading the standard input. If this option is  
 84015 omitted, the SCCS file shall be created with control information but without any  
 84016 file data. The *-i* option implies the *-n* option.

84017 **-r SID** Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that  
 84018 is, the branch and sequence numbers shall be zero or missing. The level number is  
 84019 optional, and defaults to 1.

84020 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be  
 84021 taken. In the case of existing SCCS files (neither *-i* nor *-n* is specified):

- 84022                   • A **-t** option without a *name* option-argument shall cause the removal of  
84023                   descriptive text (if any) currently in the SCCS file.
- 84024                   • A **-t** option with a *name* option-argument shall cause the text (if any) in the  
84025                   named file to replace the descriptive text (if any) currently in the SCCS file.
- 84026       **-f flag**   Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file.  
84027                   Several **-f** options may be supplied on a single *admin* command line.  
84028                   Implementations shall recognize the following flags and associated values:
- 84029                   **b**           Allow use of the **-b** option on a *get* command to create branch deltas.
- 84030                   **cceil**       Specify the highest release (that is, ceiling), a number less than or equal to  
84031                   9999, which may be retrieved by a *get* command for editing. The default  
84032                   value for an unspecified **c** flag shall be 9999.
- 84033                   **ffloor**      Specify the lowest release (that is, floor), a number greater than 0 but less  
84034                   than 9999, which may be retrieved by a *get* command for editing. The  
84035                   default value for an unspecified **f** flag shall be 1.
- 84036                   **dSID**       Specify the default delta number (SID) to be used by a *get* command.
- 84037                   **istr**       Treat the “No ID keywords” message issued by *get* or *delta* as a fatal error.  
84038                   In the absence of this flag, the message is only a warning. The message is  
84039                   issued if no SCCS identification keywords (see *get*) are found in the text  
84040                   retrieved or stored in the SCCS file. If a value is supplied, the application  
84041                   shall ensure that the keywords exactly match the given string; however,  
84042                   the string shall contain a keyword, and no embedded <newline>  
84043                   characters.
- 84044                   **j**           Allow concurrent *get* commands for editing on the same SID of an SCCS  
84045                   file. This allows multiple concurrent updates to the same version of the  
84046                   SCCS file.
- 84047                   **llist**       Specify a *list* of releases to which deltas can no longer be made (that is, *get*  
84048                   **-e** against one of these locked releases fails). Conforming applications  
84049                   shall use the following syntax to specify a *list*. Implementations may  
84050                   accept additional forms as an extension:
- 84051                   <list> ::= a | <range-list>  
84052                   <range-list> ::= <range> | <range-list>, <range>  
84053                   <range> ::= <SID>
- 84054                   The character *a* in the *list* shall be equivalent to specifying all releases for  
84055                   the named SCCS file. The non-terminal <SID> in range shall be the delta  
84056                   number of an existing delta associated with the SCCS file.
- 84057                   **n**           Cause *delta* to create a null delta in each of those releases (if any) being  
84058                   skipped when a delta is made in a new release (for example, in making  
84059                   delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas  
84060                   shall serve as anchor points so that branch deltas may later be created  
84061                   from them. The absence of this flag shall cause skipped releases to be  
84062                   nonexistent in the SCCS file, preventing branch deltas from being created  
84063                   from them in the future. During the initial creation of an SCCS file, the **n**  
84064                   flag may be ignored; that is, if the **-r** option is used to set the release  
84065                   number of the initial SID to a value greater than 1, null deltas need not be  
84066                   created for the “skipped” releases.

84067	<b>qtext</b>	Substitute user-definable <i>text</i> for all occurrences of the %Q% keyword in the SCCS file text retrieved by <i>get</i> .
84068		
84069	<b>mmod</b>	Specify the module name of the SCCS file substituted for all occurrences of the %M% keyword in the SCCS file text retrieved by <i>get</i> . If the <b>m</b> flag is not specified, the value assigned shall be the name of the SCCS file with the leading ' . ' removed.
84070		
84071		
84072		
84073	<b>ttype</b>	Specify the <i>type</i> of module in the SCCS file substituted for all occurrences of the %Y% keyword in the SCCS file text retrieved by <i>get</i> .
84074		
84075	<b>vpgm</b>	Cause <i>delta</i> to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the <b>m</b> option is also used even if its value is null.)
84076		
84077		
84078		
84079		
84080	<b>-d flag</b>	Remove (delete) the specified <i>flag</i> from an SCCS file. Several <b>-d</b> options may be supplied on a single <i>admin</i> command. See the <b>-f</b> option for allowable <i>flag</i> names. (The <b>l</b> flag gives a <i>list</i> of releases to be unlocked. See the <b>-f</b> option for further description of the <b>l</b> flag and the syntax of a <i>list</i> .)
84081		
84082		
84083		
84084	<b>-a login</b>	Specify a <i>login</i> name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-a</b> options may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If <i>login</i> or group ID is preceded by a '!', the users so specified shall be denied permission to make deltas.
84085		
84086		
84087		
84088		
84089		
84090		
84091	<b>-e login</b>	Specify a <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-e</b> options may be used on a single <i>admin</i> command line.
84092		
84093		
84094		
84095	<b>-y[comment]</b>	Insert the <i>comment</i> text into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta</i> . In the POSIX locale, omission of the <b>-y</b> option shall result in a default comment line being inserted in the form:
84096		
84097		
84098		<code>"date and time created %s %s by %s", &lt;date&gt;, &lt;time&gt;, &lt;login&gt;</code>
84099		where <date> is expressed in the format of the <i>date</i> utility's %Y/%m/%d conversion specification, <time> in the format of the <i>date</i> utility's %T conversion specification format, and <login> is the login name of the user creating the file.
84100		
84101		
84102	<b>-m mrlist</b>	Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta</i> . The application shall ensure that the <b>v</b> flag is set and the MR numbers are validated if the <b>v</b> flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the <b>v</b> flag is not set or MR validation fails.
84103		
84104		
84105		
84106		
84107	<b>-h</b>	Check the structure of the SCCS file and compare the newly computed checksum with the checksum that is stored in the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.
84108		
84109		
84110		



84111            **-z**            Recompute the SCCS file checksum and store it in the first line of the SCCS file (see  
 84112                            the **-h** option above). Note that use of this option on a truly corrupted file may  
 84113                            prevent future detection of the corruption.

#### 84114 **OPERANDS**

84115            The following operands shall be supported:

84116            *file*            A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*  
 84117                            utility shall behave as though each file in the directory were specified as a named  
 84118                            file, except that non-SCCS files (last component of the pathname does not begin  
 84119                            with **s**.) and unreadable files shall be silently ignored.

84120            *newfile*        A pathname of an SCCS file to be created.

84121            If exactly one *file* or *newfile* operand appears, and it is **'-'**, the standard input shall be read; each  
 84122                            line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-  
 84123                            SCCS files and unreadable files shall be silently ignored.

#### 84124 **STDIN**

84125            The standard input shall be a text file used only if **-i** is specified without an option-argument or  
 84126                            if a *file* or *newfile* operand is specified as **'-'**. If the first character of any standard input line is  
 84127                            <SOH> in the POSIX locale, the results are unspecified.

#### 84128 **INPUT FILES**

84129            The existing SCCS files shall be text files of an unspecified format.

84130            The application shall ensure that the file named by the **-i** option's *name* option-argument shall  
 84131                            be a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results  
 84132                            are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the  
 84133                            header for this file shall be 99 999 for this delta.

#### 84134 **ENVIRONMENT VARIABLES**

84135            The following environment variables shall affect the execution of *admin*:

84136            **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 84137                            (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 84138                            variables used to determine the values of locale categories.)

84139            **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
 84140                            internationalization variables.

84141            **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
 84142                            characters (for example, single-byte as opposed to multi-byte characters in  
 84143                            arguments and input files).

84144            **LC\_MESSAGES**

84145                            Determine the locale that should be used to affect the format and contents of  
 84146                            diagnostic messages written to standard error and the contents of the default **-y**  
 84147                            comment.

84148            **NLSPATH**     Determine the location of messages objects and message catalogs.

#### 84149 **ASYNCHRONOUS EVENTS**

84150            Default.

#### 84151 **STDOUT**

84152            Not used.

84153 **STDERR**

84154 The standard error shall be used only for diagnostic messages.

84155 **OUTPUT FILES**

84156 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a  
84157 locking *z-file*, as described in *get* (on page 2964), may be created and deleted.

84158 **EXTENDED DESCRIPTION**

84159 None.

84160 **EXIT STATUS**

84161 The following exit values shall be returned:

84162 0 Successful completion.

84163 >0 An error occurred.

84164 **CONSEQUENCES OF ERRORS**

84165 Default.

84166 **APPLICATION USAGE**

84167 It is recommended that directories containing SCCS files be writable by the owner only, and that  
84168 SCCS files themselves be read-only. The mode of the directories should allow only the owner to  
84169 modify SCCS files contained in the directories. The mode of the SCCS files prevents any  
84170 modification at all except by SCCS commands.

84171 **EXAMPLES**

84172 None.

84173 **RATIONALE**

84174 None.

84175 **FUTURE DIRECTIONS**

84176 If this utility is directed to display a pathname that contains any bytes that have the encoded  
84177 value of a <newline> character when <newline> is a terminator or separator in the output  
84178 format being used, implementations are encouraged to treat this as an error. A future version of  
84179 this standard may require implementations to treat this as an error.

84180 If this utility is directed to create a new directory entry that contains any bytes that have the  
84181 encoded value of a <newline> character, implementations are encouraged to treat this as an  
84182 error. A future version of this standard may require implementations to treat this as an error.

84183 **SEE ALSO**

84184 *delta, get, prs, what*

84185 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

84186 **CHANGE HISTORY**

84187 First released in Issue 2.

84188 **Issue 6**

84189 The normative text is reworded to avoid use of the term “must” for application requirements,  
84190 and to emphasize the term “shall” for implementation requirements.

84191 The grammar is updated.

84192 The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES  
84193 section warning that the maximum lines recorded in the file is 99 999.

84194 The Open Group Base Resolution bwg2001-009 is applied, amending the description of the **-h**  
84195 option.

84196 **Issue 8**

84197 Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

- 84198 a. Report an error if a utility is directed to display a pathname that contains any bytes that  
84199 have the encoded value of a <newline> character when <newline> is a terminator or  
84200 separator in the output format being used.
- 84201 b. Disallow the creation of filenames containing any bytes that have the encoded value of a  
84202 <newline> character.

84203 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

84204 **NAME**

84205 alias — define or display aliases

84206 **SYNOPSIS**84207 `alias [alias-name[=string] . . .]`84208 **DESCRIPTION**

84209 The *alias* utility shall create or redefine alias definitions or write the values of existing alias  
 84210 definitions to standard output. An alias definition provides a string value that shall replace a  
 84211 command name when it is encountered. For information on valid string values, and the  
 84212 processing involved, see [Section 2.3.1](#) (on page 2477).

84213 An alias definition shall affect the current shell execution environment and the execution  
 84214 environments of the subshells of the current shell. When used as specified by this volume of  
 84215 POSIX.1-2024, the alias definition shall not affect the parent process of the current shell nor any  
 84216 utility environment invoked by the shell; see [Section 2.13](#) (on page 2522).

84217 **OPTIONS**

84218 None.

84219 **OPERANDS**

84220 The following operands shall be supported:

84221 *alias-name* Write the alias definition to standard output.84222 *alias-name=string*84223 Assign the value of *string* to the alias *alias-name*.

84224 If no operands are given, all alias definitions shall be written to standard output.

84225 **STDIN**

84226 Not used.

84227 **INPUT FILES**

84228 None.

84229 **ENVIRONMENT VARIABLES**84230 The following environment variables shall affect the execution of *alias*:

84231 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 84232 (See [XBD Section 8.2](#) (on page 169) for the precedence of internationalization  
 84233 variables used to determine the values of locale categories.)

84234 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 84235 internationalization variables.

84236 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 84237 characters (for example, single-byte as opposed to multi-byte characters in  
 84238 arguments).

84239 *LC\_MESSAGES*

84240 Determine the locale that should be used to affect the format and contents of  
 84241 diagnostic messages written to standard error.

84242 *XSI* *NLSPATH* Determine the location of messages objects and message catalogs.84243 **ASYNCHRONOUS EVENTS**

84244 Default.

84245 **STDOUT**

84246 The format for displaying aliases (when no operands or only *name* operands are specified) shall  
84247 be:

84248 `"%s=%s\n", name, value`

84249 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the  
84250 shell. See the description of shell quoting in [Section 2.2](#) (on page 2472).

84251 **STDERR**

84252 The standard error shall be used only for diagnostic messages.

84253 **OUTPUT FILES**

84254 None.

84255 **EXTENDED DESCRIPTION**

84256 None.

84257 **EXIT STATUS**

84258 The following exit values shall be returned:

84259 0 Successful completion.

84260 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.

84261 **CONSEQUENCES OF ERRORS**

84262 Default.

84263 **APPLICATION USAGE**

84264 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

84265 Care should be taken to avoid alias values that end with a character that could be treated as part  
84266 of an operator token, as it is unspecified whether the character that follows the alias name in the  
84267 input can be used as part of the same token (see [Section 2.3.1](#), on page 2477). For example, with:

84268 `$ alias foo='echo 0'`  
84269 `$ foo>&2`

84270 the shell can either pass the argument '0' to *echo* and redirect fd 1 to fd 2, or pass no arguments  
84271 to *echo* and redirect fd 0 to fd 2. Changing it to:

84272 `$ alias foo='echo "0"'`

84273 avoids this problem. The alternative of adding a `<space>` after the '0' would also avoid the  
84274 problem, but in addition it would alter the way the alias works, as described in [Section 2.3.1](#) (on  
84275 page 2477).

84276 Likewise, given:

84277 `$ alias foo='some_command &'`  
84278 `$ foo&`

84279 the shell may combine the two '&' characters into an `&&` (and) operator. Since the alias cannot  
84280 pass arguments to *some\_command* and thus can be expected to be invoked without arguments,  
84281 adding a `<space>` after the '&' would be an acceptable way to prevent this. Alternatively, the  
84282 alias could be specified as a grouping command:

84283 `$ alias foo='{ some_command & }'`

84284 Problems can occur for tokens other than operators as well, if the alias is used in unusual ways.  
84285 For example, with:

```
84286 $ alias foo='echo $'
84287 $ foo((123))
```

84288 some shells combine the '\$' and the "((123))" to form an arithmetic expansion, but others  
84289 do not (resulting in a syntax error).

#### 84290 EXAMPLES

84291 1. Create a short alias for a commonly used *ls* command:

```
84292 alias lf="ls -CF"
```

84293 2. Create a simple ``redo'' command to repeat previous entries in the command history file:

```
84294 alias r='fc -s'
```

84295 3. Use 1K units for *du*:

```
84296 alias du=du\ -k
```

84297 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:

```
84298 alias nohup="nohup "
```

84299 5. Add the **-F** option to interactive uses of *ls*, even when executed as *xargs ls* or  
84300 *xargs -0 ls*:

```
84301 alias ls='ls -F'
84302 alias xargs='xargs '
84303 alias -- -0='-0 '
84304 find . [...] -print | xargs ls          # breaks on filenames with \n
84305                                     # (two aliases expanded)
84306 find . [...] -print0 | xargs -0 ls     # minimizes \n issues (three
84307                                     # aliases expanded)
```

#### 84308 RATIONALE

84309 The *alias* description is based on historical KornShell implementations. Known differences exist  
84310 between that and the C shell. The KornShell version was adopted to be consistent with all the  
84311 other KornShell features in this volume of POSIX.1-2024, such as command line editing.

84312 Since *alias* affects the current shell execution environment, it is generally provided as a shell  
84313 regular built-in.

84314 Historical versions of the KornShell have allowed aliases to be exported to scripts that are  
84315 invoked by the same shell. This is triggered by the *alias -x* flag; it is allowed by this volume of  
84316 POSIX.1-2024 only when an explicit extension such as *-x* is used. The standard developers  
84317 considered that aliases were of use primarily to interactive users and that they should normally  
84318 not affect shell scripts called by those users; functions are available to such scripts.

84319 Historical versions of the KornShell had not written aliases in a quoted manner suitable for  
84320 reentry to the shell, but this volume of POSIX.1-2024 has made this a requirement for all similar  
84321 output. Therefore, consistency was chosen over this detail of historical practice.

#### 84322 FUTURE DIRECTIONS

84323 None.

#### 84324 SEE ALSO

84325 [Section 2.9.5](#) (on page 2511)

84326 [XBD Chapter 8](#) (on page 167)

84327 **CHANGE HISTORY**

84328 First released in Issue 4.

84329 **Issue 6**

84330 This utility is marked as part of the User Portability Utilities option.

84331 The APPLICATION USAGE section is added.

84332 **Issue 7**

84333 The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

84335 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84336 The first example is changed to remove the creation of an alias for a standard utility that alters its behavior to be non-conforming.

84338 **Issue 8**

84339 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

84341 Austin Group Defect 953 is applied, clarifying that the details of how alias replacement is performed are in the cross-referenced section ([Section 2.3.1](#), on page 2477) and updating the APPLICATION USAGE section.

84344 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

84345 Austin Group Defect 1630 is applied, adding a new item in EXAMPLES.

84346 **NAME**

84347 ar — create and maintain library archives

84348 **SYNOPSIS**84349 SD ar -d [-v] *archive file...*84350 XSI ar -m [-v] *archive file...*84351 ar -m -a [-v] *posname archive file...*84352 ar -m -b [-v] *posname archive file...*84353 ar -m -i [-v] *posname archive file...*84354 XSI ar -p [-v] [-s] *archive [file...]*84355 XSI ar -q [-cv] *archive file...*84356 ar -r [-cuv] *archive file...*84357 XSI ar -r -a [-cuv] *posname archive file...*84358 ar -r -b [-cuv] *posname archive file...*84359 ar -r -i [-cuv] *posname archive file...*84360 XSI ar -t [-v] [-s] *archive [file...]*84361 XSI ar -x [-v] [-sCT] *archive [file...]*84362 **DESCRIPTION**84363 The *ar* utility is part of the Software Development Utilities option.

84364 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once  
 84365 an archive has been created, new files can be added, and existing files in an archive can be  
 84366 extracted, deleted, or replaced. When an archive consists entirely of valid object files, the  
 84367 implementation shall format the archive so that it is usable as a library for link editing (see *c17*).  
 84368 When some of the archived files are not valid object files, the suitability of the archive for library  
 84369 XSI use is undefined. If an archive consists entirely of printable files, the entire archive shall be  
 84370 printable.

84371 When *ar* creates an archive, it creates administrative information indicating whether a symbol  
 84372 table is present in the archive. When there is at least one object file that *ar* recognizes as such in  
 84373 the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is  
 84374 used by the link editor to search the archive. Whenever the *ar* utility is used to create or update  
 84375 the contents of such an archive, the symbol table shall be rebuilt. The *-s* option shall force the  
 84376 symbol table to be rebuilt.

84377 All *file* operands can be pathnames. However, files within archives shall be named by a filename,  
 84378 which is the last component of the pathname used when the file was entered into the archive.  
 84379 The comparison of *file* operands to the names of files in archives shall be performed by  
 84380 comparing the last component of the operand to the name of the file in the archive.

84381 It is unspecified whether multiple files in the archive may be identically named. In the case of  
 84382 XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive  
 84383 having a name that is the same as the last component of the operand.



84384 **OPTIONS**

84385 The *ar* utility shall conform to XBD [Section 12.2](#) (on page 215), except for Guideline 9.

84386 The following options shall be supported:

- 84387 XSI **-a** Position new files in the archive after the file named by the *posname* operand.
- 84388 XSI **-b** Position new files in the archive before the file named by the *posname* operand.
- 84389 **-c** Suppress the diagnostic message that is written to standard error by default when the archive *archive* is created.
- 84391 XSI **-C** Prevent extracted files from replacing like-named files in the file system. This option is useful when **-T** is also used, to prevent truncated filenames from replacing files with the same prefix.
- 84392
- 84393
- 84394 **-d** Delete one or more *files* from *archive*.
- 84395 XSI **-i** Position new files in the archive before the file in the archive named by the *posname* operand (equivalent to **-b**).
- 84396
- 84397 XSI **-m** Move the named files in the archive. The **-a**, **-b**, or **-i** options with the *posname* operand indicate the position; otherwise, move the names files in the archive to the end of the archive.
- 84398
- 84399
- 84400 **-p** Write the contents of the *files* in the archive named by *file* operands from *archive* to the standard output. If no *file* operands are specified, the contents of all files in the archive shall be written in the order of the archive.
- 84401
- 84402
- 84403 XSI **-q** Append the named files to the end of the archive. In this case *ar* does not check whether the added files are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece by piece.
- 84404
- 84405
- 84406 **-r** Replace or add *files* to *archive*. If the archive named by *archive* does not exist, a new archive shall be created and a diagnostic message shall be written to standard error (unless the **-c** option is specified). If no *files* are specified and the *archive* exists, the results are undefined. Files that replace existing files in the archive shall not change the order of the archive. Files that do not replace existing files in the archive shall be appended to the archive unless a **-a**, **-b**, or **-i** option specifies another position.
- 84407
- 84408
- 84409
- 84410
- 84411 XSI
- 84412 XSI **-s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an option that modifies the archive contents. This option is useful to restore the archive symbol table after it has been stripped; see *strip*.
- 84413
- 84414
- 84415 **-t** Write a table of contents of *archive* to the standard output. Only the files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.
- 84416
- 84417
- 84418 XSI **-T** Allow filename truncation of extracted files whose archive names are longer than the file system can support. By default, extracting a file with a name that is too long shall be an error; a diagnostic message shall be written and the file shall not be extracted.
- 84419
- 84420
- 84421
- 84422 **-u** Update older files in the archive. When used with the **-r** option, files in the archive shall be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file in the archive.
- 84423
- 84424

84425        **-v**           Give verbose output. When used with the option characters **-d**, **-r**, or **-x**, write a  
84426                   detailed file-by-file description of the archive creation and maintenance activity, as  
84427                   described in the STDOUT section.

84428                   When used with **-p**, write the name of the file in the archive to the standard output  
84429                   before writing the file in the archive itself to the standard output, as described in  
84430                   the STDOUT section.

84431                   When used with **-t**, include a long listing of information about the files in the  
84432                   archive, as described in the STDOUT section.

84433        **-x**           Extract the files in the archive named by the *file* operands from *archive*. The  
84434                   contents of the archive shall not be changed. If no *file* operands are given, all files  
84435                   in the archive shall be extracted. The modification time of each file extracted shall  
84436                   be set to the time the file is extracted from the archive.

#### 84437 **OPERANDS**

84438           The following operands shall be supported:

84439        *archive*       A pathname of the archive.

84440        *file*           A pathname. Only the last component shall be used when comparing against the  
84441                   names of files in the archive. If two or more *file* operands have the same last  
84442                   pathname component (basename), the results are unspecified. The  
84443                   implementation's archive format shall not truncate valid filenames of files added  
84444                   to or replaced in the archive.

84445 XSI        *posname*     The name of a file in the archive, used for relative positioning; see options **-m** and  
84446                   **-r**.

#### 84447 **STDIN**

84448           Not used.

#### 84449 **INPUT FILES**

84450           The archive named by *archive* shall be a file in the format created by *ar -r*.

#### 84451 **ENVIRONMENT VARIABLES**

84452           The following environment variables shall affect the execution of *ar*:

84453        **LANG**           Provide a default value for the internationalization variables that are unset or null.  
84454                   (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
84455                   variables used to determine the values of locale categories.)

84456        **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
84457                   internationalization variables.

84458        **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
84459                   characters (for example, single-byte as opposed to multi-byte characters in  
84460                   arguments and input files).

84461        **LC\_MESSAGES**

84462                   Determine the locale that should be used to affect the format and contents of  
84463                   diagnostic messages written to standard error.

84464        **LC\_TIME**       Determine the format and content for date and time strings written by *ar -tv*.

84465 XSI        **NLSPATH**     Determine the location of messages objects and message catalogs.

84466 *TMPDIR* Determine the pathname that overrides the default directory for temporary files, if  
84467 any.

84468 *TZ* Determine the timezone used to calculate date and time strings written by *ar -tv*.  
84469 If *TZ* is unset or null, an unspecified default timezone shall be used.

#### 84470 ASYNCHRONOUS EVENTS

84471 Default.

#### 84472 STDOUT

84473 If the *-d* option is used with the *-v* option, the standard output format shall be:

84474 "d - %s\n", <file>

84475 where *file* is the operand specified on the command line.

84476 If the *-p* option is used with the *-v* option, *ar* shall precede the contents of each file with:

84477 "\n<%s>\n\n", <file>

84478 where *file* is the operand specified on the command line, if *file* operands were specified, and the  
84479 name of the file in the archive if they were not.

84480 If the *-r* option is used with the *-v* option:

84481 • If *file* is already in the archive, the standard output format shall be:

84482 "r - %s\n", <file>

84483 where <file> is the operand specified on the command line.

84484 • If *file* is not already in the archive, the standard output format shall be:

84485 "a - %s\n", <file>

84486 where <file> is the operand specified on the command line.

84487 If the *-t* option is used, *ar* shall write the names of the files in the archive to the standard output  
84488 in the format:

84489 "%s\n", <file>

84490 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
84491 name of the file in the archive if they were not.

84492 If the *-t* option is used with the *-v* option, the standard output format shall be:

84493 "%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>,  
84494 <group ID>, <number of bytes in member>,  
84495 <abbreviated month>, <day-of-month>, <hour>,  
84496 <minute>, <year>, <file>

84497 where:

84498 <file> Shall be the operand specified on the command line, if *file* operands were specified,  
84499 or the name of the file in the archive if they were not.

84500 <member mode>

84501 Shall be formatted the same as the <file mode> string defined in the STDOUT  
84502 section of *ls*, except that the first character, the <entry type>, is not used; the string  
84503 represents the file mode of the file in the archive at the time it was added to or  
84504 replaced in the archive.

84505 The following represent the last-modification time of a file when it was most recently added to  
84506 or replaced in the archive:

84507 <*abbreviated month*>

84508 Equivalent to the format of the %b conversion specification format in *date*.

84509 <*day-of-month*>

84510 Equivalent to the format of the %e conversion specification format in *date*.

84511 <*hour*>

Equivalent to the format of the %H conversion specification format in *date*.

84512 <*minute*>

Equivalent to the format of the %M conversion specification format in *date*.

84513 <*year*>

Equivalent to the format of the %Y conversion specification format in *date*.

84514 When *LC\_TIME* does not specify the POSIX locale, a different format and order of presentation  
84515 of these fields relative to each other may be used in a format appropriate in the specified locale.

84516 If the *-x* option is used with the *-v* option, the standard output format shall be:

84517 "x - %s\n", <*file*>

84518 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
84519 name of the file in the archive if they were not.

#### 84520 **STDERR**

84521 The standard error shall be used only for diagnostic messages. The diagnostic message about  
84522 creating a new archive when *-c* is not specified shall not modify the exit status.

#### 84523 **OUTPUT FILES**

84524 Archives are files with unspecified formats.

#### 84525 **EXTENDED DESCRIPTION**

84526 None.

#### 84527 **EXIT STATUS**

84528 The following exit values shall be returned:

84529 0 Successful completion.

84530 >0 An error occurred.

#### 84531 **CONSEQUENCES OF ERRORS**

84532 Default.

#### 84533 **APPLICATION USAGE**

84534 None.

#### 84535 **EXAMPLES**

84536 None.

#### 84537 **RATIONALE**

84538 The archive format is not described. It is recognized that there are several known *ar* formats,  
84539 which are not compatible. The *ar* utility is included, however, to allow creation of archives that  
84540 are intended for use only on one machine. The archive is specified as a file, and it can be moved  
84541 as a file. This does allow an archive to be moved from one machine to another machine that uses  
84542 the same implementation of *ar*.

84543 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable “archives”. This is a not  
84544 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the  
84545 compilers, based on a historical model.

84546 In historical implementations, the `-q` option (available on XSI-conforming systems) is known to  
84547 execute quickly because *ar* does not check on whether the added members are already in the  
84548 archive. This is useful to bypass the searching otherwise done when creating a large archive  
84549 piece-by-piece. These remarks may but need not remain true for a brand new implementation of  
84550 this utility; hence, these remarks have been moved into the RATIONALE.

84551 BSD implementations historically required applications to provide the `-s` option whenever the  
84552 archive was supposed to contain a symbol table. As in this volume of POSIX.1-2024, System V  
84553 historically creates or updates an archive symbol table whenever an object file is removed from,  
84554 added to, or updated in the archive.

84555 The OPERANDS section requires what might seem to be true without specifying it: the archive  
84556 cannot truncate the filenames below `{NAME_MAX}`. Some historical implementations do so,  
84557 however, causing unexpected results for the application. Therefore, this volume of POSIX.1-2024  
84558 makes the requirement explicit to avoid misunderstandings.

84559 According to the System V documentation, the options `-dmpqrtx` are not required to begin with  
84560 a `<hyphen-minus>` ('-'). This volume of POSIX.1-2024 requires that a conforming application  
84561 use the leading `<hyphen-minus>`.

84562 The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as  
84563 an example:

84564 A file created by *ar* begins with the "magic" string `!<arch>\n`". The rest of the archive  
84565 is made up of objects, each of which is composed of a header for a file, a possible filename,  
84566 and the file contents. The header is portable between machine architectures, and, if the file  
84567 contents are printable, the archive is itself printable.

84568 The header is made up of six ASCII fields, followed by a two-character trailer. The fields  
84569 are the object name (16 characters), the file last modification time (12 characters), the user  
84570 and group IDs (each 6 characters), the file mode (8 characters), and the file size (10  
84571 characters). All numeric fields are in decimal, except for the file mode, which is in octal.

84572 The modification time is the file `st_mtime` field. The user and group IDs are the file `st_uid`  
84573 and `st_gid` fields. The file mode is the file `st_mode` field. The file size is the file `st_size` field.  
84574 The two-byte trailer is the string `"`<newline>`".

84575 Only the name field has any provision for overflow. If any filename is more than 16  
84576 characters in length or contains an embedded space, the string `"#1/"` followed by the  
84577 ASCII length of the name is written in the name field. The file size (stored in the archive  
84578 header) is incremented by the length of the name. The name is then written immediately  
84579 following the archive header.

84580 Any unused characters in any of these fields are written as `<space>` characters. If any fields  
84581 are their particular maximum number of characters in length, there is no separation  
84582 between the fields.

84583 Objects in the archive are always an even number of bytes long; files that are an odd  
84584 number of bytes long are padded with a `<newline>`, although the size in the header does  
84585 not reflect this.

84586 The *ar* utility description requires that (when all its members are valid object files) *ar* produce an  
84587 object code library, which the linkage editor can use to extract object modules. If the linkage  
84588 editor needs a symbol table to permit random access to the archive, *ar* must provide it; however,  
84589 *ar* does not require a symbol table.

84590 The BSD `-o` option was omitted. It is a rare conforming application that uses *ar* to extract object

84591 code from a library with concern for its modification time, since this can only be of importance  
 84592 to *make*. Hence, since this functionality is not deemed important for applications portability, the  
 84593 modification time of the extracted files is set to the current time.

84594 There is at least one known implementation (for a small computer) that can accommodate only  
 84595 object files for that system, disallowing mixed object and other files. The ability to handle any  
 84596 type of file is not only historical practice for most implementations, but is also a reasonable  
 84597 expectation.

84598 Consideration was given to changing the output format of *ar -tv* to the same format as the  
 84599 output of *ls -l*. This would have made parsing the output of *ar* the same as that of *ls*. This was  
 84600 rejected in part because the current *ar* format is commonly used and changes would break  
 84601 historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a  
 84602 <slash>. Changing this to be the user name and group name would not be correct if the archive  
 84603 were moved to a machine that contained a different user database. Since *ar* cannot know  
 84604 whether the archive was generated on the same machine, it cannot tell what to report.

84605 The text on the *-ur* option combination is historical practice—since one filename can easily  
 84606 represent two different files (for example, */a/foo* and */b/foo*), it is reasonable to replace the file in  
 84607 the archive even when the modification time in the archive is identical to that in the file system.

#### 84608 FUTURE DIRECTIONS

84609 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 84610 value of a <newline> character when <newline> is a terminator or separator in the output  
 84611 format being used, implementations are encouraged to treat this as an error. A future version of  
 84612 this standard may require implementations to treat this as an error.

84613 If this utility is directed to create a new directory entry that contains any bytes that have the  
 84614 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 84615 error. A future version of this standard may require implementations to treat this as an error.

#### 84616 SEE ALSO

84617 *c17, date, pax, strip*

84618 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215), [<unistd.h>](#), description of  
 84619 {POSIX\_NO\_TRUNC}

#### 84620 CHANGE HISTORY

84621 First released in Issue 2.

#### 84622 Issue 5

84623 The FUTURE DIRECTIONS section is added.

#### 84624 Issue 6

84625 This utility is marked as part of the Software Development Utilities option.

84626 The STDOUT description is changed for the *-v* option to align with the IEEE P1003.2b draft  
 84627 standard.

84628 The normative text is reworded to avoid use of the term “must” for application requirements.

84629 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

84630 IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use  
 84631 “file” to refer to a file in the file system hierarchy, “archive” to refer to the archive being  
 84632 operated upon by the *ar* utility, and “file in the archive” to refer to a copy of a file that is  
 84633 contained in the archive.

84634 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/10 is applied, making corrections to the

84635 SYNOPSIS. The change was needed since the **-a**, **-b**, and **-i** options are mutually-exclusive, and  
84636 *posname* is required if any of these options is specified.

84637 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/11 is applied, correcting the description  
84638 of the two-byte trailer in RATIONALE which had missed out a backquote. The correct trailer is a  
84639 backquote followed by a <newline>.

84640 **Issue 7**

84641 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
84642 apply.

84643 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84644 The description of the **-t** option is changed to say ``Only the files specified ...''.

84645 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0057 [584] is applied.

84646 **Issue 8**

84647 Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

84648 a. Report an error if a utility is directed to display a pathname that contains any bytes that  
84649 have the encoded value of a <newline> character when <newline> is a terminator or  
84650 separator in the output format being used.

84651 b. Disallow the creation of filenames containing any bytes that have the encoded value of a  
84652 <newline> character.

84653 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

84654 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

84655 **NAME**

84656           asa — interpret carriage-control characters

84657 **SYNOPSIS**84658 FR       asa [*file...*]84659 **DESCRIPTION**84660           The *asa* utility shall write its input files to standard output, mapping carriage-control characters  
84661           from the text files to line-printer control sequences in an implementation-defined manner.84662           The first character of every line shall be removed from the input, and the following actions are  
84663           performed.

84664           If the character removed is:

84665           &lt;space&gt;     The rest of the line is output without change.

84666           0           A &lt;newline&gt; is output, then the rest of the input line.

84667           1           One or more implementation-defined characters that causes an advance to the next  
84668           page shall be output, followed by the rest of the input line.84669           +           The <newline> of the previous line shall be replaced with one or more  
84670           implementation-defined characters that causes printing to return to column  
84671           position 1, followed by the rest of the input line. If the '+' is the first character in  
84672           the input, it shall be equivalent to <space>.84673           The action of the *asa* utility is unspecified upon encountering any character other than those  
84674           listed above as the first character in a line.84675 **OPTIONS**

84676           None.

84677 **OPERANDS**84678           *file*         A pathname of a text file used for input. If no *file* operands are specified, the  
84679           standard input shall be used.84680 **STDIN**84681           The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
84682           operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
84683           the standard input shall not be used. See the INPUT FILES section.84684 **INPUT FILES**

84685           The input files shall be text files.

84686 **ENVIRONMENT VARIABLES**84687           The following environment variables shall affect the execution of *asa*:84688           LANG         Provide a default value for the internationalization variables that are unset or null.  
84689                         (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
84690                         variables used to determine the values of locale categories.)84691           LC\_ALL        If set to a non-empty string value, override the values of all the other  
84692           internationalization variables.84693           LC\_CTYPE     Determine the locale for the interpretation of sequences of bytes of text data as  
84694           characters (for example, single-byte as opposed to multi-byte characters in  
84695           arguments and input files).



- 84696 *LC\_MESSAGES*  
84697 Determine the locale that should be used to affect the format and contents of  
84698 diagnostic messages written to standard error.
- 84699 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 84700 **ASYNCHRONOUS EVENTS**  
84701 Default.
- 84702 **STDOUT**  
84703 The standard output shall be the text from the input file modified as described in the  
84704 DESCRIPTION section.
- 84705 **STDERR**  
84706 None.
- 84707 **OUTPUT FILES**  
84708 None.
- 84709 **EXTENDED DESCRIPTION**  
84710 None.
- 84711 **EXIT STATUS**  
84712 The following exit values shall be returned:  
84713 0 All input files were output successfully.  
84714 >0 An error occurred.
- 84715 **CONSEQUENCES OF ERRORS**  
84716 Default.
- 84717 **APPLICATION USAGE**  
84718 None.
- 84719 **EXAMPLES**  
84720 1. The following command:  
84721 *asa file*  
84722 permits the viewing of *file* (created by a program using FORTRAN-style carriage-control  
84723 characters) on a terminal.  
84724 2. The following command:  
84725 *a.out | asa | lp*  
84726 formats the FORTRAN output of **a.out** and directs it to the printer.
- 84727 **RATIONALE**  
84728 The *asa* utility is needed to map “standard” FORTRAN 77 output into a form acceptable to  
84729 contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.  
84730 This utility is generally used only by FORTRAN programs. The standard developers decided to  
84731 retain *asa* to avoid breaking the historical large base of FORTRAN applications that put carriage-  
84732 control characters in their output files. There is no requirement that a system have a FORTRAN  
84733 compiler in order to run applications that need *asa*.  
84734 Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII  
84735 <carriage-return> in response to a '+'. It is suggested that implementations treat characters  
84736 other than 0, 1, and '+' as <space> in the absence of any compelling reason to do otherwise.

84737 However, the action is listed here as ``unspecified'', permitting an implementation to provide  
84738 extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

84739 **FUTURE DIRECTIONS**

84740 None.

84741 **SEE ALSO**

84742 *lp*

84743 XBD [Chapter 8](#) (on page 167)

84744 **CHANGE HISTORY**

84745 First released in Issue 4.

84746 **Issue 6**

84747 This utility is marked as part of the FORTRAN Runtime Utilities option.

84748 The normative text is reworded to avoid use of the term ``must'' for application requirements.

84749 **Issue 7**

84750 Austin Group Interpretation 1003.1-2001 #092 is applied.

84751 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84752 **Issue 8**

84753 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

84754 **NAME**

84755 at — execute commands at a later time

84756 **SYNOPSIS**84757 at [-m] [-f *file*] [-q *queuename*] -t *time\_arg*84758 at [-m] [-f *file*] [-q *queuename*] *timespec...*84759 at -r *at\_job\_id...*84760 at -l -q *queuename*84761 at -l [*at\_job\_id...*]84762 **DESCRIPTION**84763 The *at* utility shall read commands from standard input and group them together as an *at-job*, to  
84764 be executed at a later time.84765 The *at-job* shall be executed in a separate invocation of the shell, running in a separate process  
84766 group with no controlling terminal, except that the environment variables, current working  
84767 directory, file creation mask, and other implementation-defined execution-time attributes in  
84768 effect when the *at* utility is executed shall be retained and used when the *at-job* is executed.84769 When the *at-job* is submitted, the *at\_job\_id* and scheduled time shall be written to standard error.  
84770 The *at\_job\_id* is an identifier that shall be a string consisting solely of alphanumeric characters  
84771 and the <period> character. The *at\_job\_id* shall be assigned by the system when the job is  
84772 scheduled such that it uniquely identifies a particular job.84773 User notification and the processing of the job's standard output and standard error are  
84774 described under the **-m** option.84775 XSI Users shall be permitted to use *at* if their name appears in the file **at.allow** which is located in an  
84776 implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in  
84777 an implementation-defined directory, shall be checked to determine whether the user shall be  
84778 denied access to *at*. If neither file exists, only a process with appropriate privileges shall be  
84779 allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The  
84780 **at.allow** and **at.deny** files shall consist of one user name per line.84781 **OPTIONS**84782 The *at* utility shall conform to XBD [Section 12.2](#) (on page 215).

84783 The following options shall be supported:

84784 **-f file** Specify the pathname of a file to be used as the source of the *at-job*, instead of  
84785 standard input.84786 **-l** (The letter ell.) Report all jobs scheduled for the invoking user if no *at\_job\_id*  
84787 operands are specified. If *at\_job\_ids* are specified, report only information for these  
84788 jobs. The output shall be written to standard output.84789 **-m** Send mail to the invoking user after the *at-job* has run, announcing its completion.  
84790 Standard output and standard error produced by the *at-job* shall be mailed to the  
84791 user as well, unless redirected elsewhere. Mail shall be sent even if the job  
84792 produces no output.84793 If **-m** is not used, the job's standard output and standard error shall be provided to  
84794 the user by means of mail, unless they are redirected elsewhere; if there is no such  
84795 output to provide, the implementation need not notify the user of the job's  
84796 completion.

84797	<b>-q</b> <i>queuename</i>	
84798		Specify in which queue to schedule a job for submission. When used with the <b>-l</b>
84799		option, limit the search to that particular queue. By default, at-jobs shall be
84800		scheduled in queue <i>a</i> . In contrast, queue <i>b</i> shall be reserved for batch jobs; see
84801		<i>batch</i> . The meanings of all other <i>queuename</i> s are implementation-defined. If <b>-q</b> <i>b</i> is
84802		specified along with either of the <b>-t</b> <i>time_arg</i> or <i>timespec</i> arguments, the results are
84803		unspecified.
84804	<b>-r</b>	Remove the jobs with the specified <i>at_job_id</i> operands that were previously
84805		scheduled by the <i>at</i> utility.
84806	<b>-t</b> <i>time_arg</i>	Submit the job to be run at the time specified by the <i>time</i> option-argument, which
84807		the application shall ensure has the format as specified by the <i>touch -t time</i> utility.

## 84808 OPERANDS

84809 The following operands shall be supported:

84810	<i>at_job_id</i>	The name reported by a previous invocation of the <i>at</i> utility at the time the job was
84811		scheduled.
84812	<i>timespec</i>	Submit the job to be run at the date and time specified. All of the <i>timespec</i> operands
84813		are interpreted as if they were separated by <space> characters and concatenated,
84814		and shall be parsed as described in the grammar at the end of this section. The date
84815		and time shall be interpreted as being in the timezone of the user (as determined
84816		by the <i>TZ</i> variable), unless a timezone name appears as part of <i>time</i> , below.
84817		In the POSIX locale, the following describes the three parts of the time specification
84818		string. All of the values from the <i>LC_TIME</i> categories in the POSIX locale shall be
84819		recognized in a case-insensitive manner.
84820	<i>time</i>	The time can be specified as one, two, or four digits. One-digit and
84821		two-digit numbers shall be taken to be hours; four-digit numbers to
84822		be hours and minutes. The time can alternatively be specified as two
84823		numbers separated by a <colon>, meaning <i>hour:minute</i> . If the
84824		<i>LC_TIME</i> category of the locale supports 12-hour time format (see
84825		XBD Section 7.3.5, on page 152), an AM/PM indication in the form of
84826		one of the values from the <b>am_pm</b> keywords in the <i>LC_TIME</i> locale
84827		category can follow the time; otherwise, a 24-hour clock time shall be
84828		understood. A timezone name can also follow to further qualify the
84829		time. The acceptable timezone names are implementation-defined,
84830		except that they shall be case-insensitive and the string <b>utc</b> is
84831		supported to indicate the time is in Coordinated Universal Time. In
84832		the POSIX locale, the <i>time</i> field can also be one of the following
84833		tokens:
84834	<b>midnight</b>	Indicates the time 12:00 am (00:00).
84835	<b>noon</b>	Indicates the time 12:00 pm.
84836	<b>now</b>	Indicates the current day and time. Invoking <i>at</i> <b>&lt;now&gt;</b>
84837		shall submit an at-job for potentially immediate
84838		execution (that is, subject only to unspecified
84839		scheduling delays).
84840	<i>date</i>	An optional <i>date</i> can be specified as either a month name (one of the
84841		values from the <b>mon</b> or <b>abmon</b> keywords in the <i>LC_TIME</i> locale
84842		category) followed by a day number (and possibly year number

84843 preceded by a comma), or a day of the week (one of the values from  
 84844 the **day** or **abday** keywords in the *LC\_TIME* locale category). In the  
 84845 POSIX locale, two special days shall be recognized:

84846 **today** Indicates the current day.

84847 **tomorrow** Indicates the day following the current day.

84848 If no *date* is given, **today** shall be assumed if the given time is greater  
 84849 than the current time, and **tomorrow** shall be assumed if it is less. If  
 84850 the given month is less than the current month (and no year is given),  
 84851 next year shall be assumed.

84852 *increment* The optional *increment* shall be a number preceded by a <plus-sign>  
 84853 ('+') and suffixed by one of the following: **minutes**, **hours**, **days**,  
 84854 **weeks**, **months**, or **years**. (The singular forms shall also be accepted.)  
 84855 The keyword **next** shall be equivalent to an increment number of +1.  
 84856 For example, the following are equivalent commands:

```
84857 at 2pm + 1 week
84858 at 2pm next week
```

84859 The following grammar describes the precise format of *timespec* in the POSIX locale. The general  
 84860 conventions for this style of grammar are described in [Section 1.3](#) (on page 2461). This formal  
 84861 syntax shall take precedence over the preceding text syntax description. The longest possible  
 84862 token or delimiter shall be recognized at a given point. When used in a *timespec*, white space  
 84863 shall also delimit tokens.

```
84864 %token hr24clock_hr_min
84865 %token hr24clock_hour
84866 /*
84867 An hr24clock_hr_min is a one, two, or four-digit number. A one-digit
84868 or two-digit number constitutes an hr24clock_hour. An hr24clock_hour
84869 may be any of the single digits [0,9], or may be double digits, ranging
84870 from [00,23]. If an hr24clock_hr_min is a four-digit number, the
84871 first two digits shall be a valid hr24clock_hour, while the last two
84872 represent the number of minutes, from [00,59].
84873 */
```

```
84874 %token wallclock_hr_min
84875 %token wallclock_hour
84876 /*
84877 A wallclock_hr_min is a one, two-digit, or four-digit number.
84878 A one-digit or two-digit number constitutes a wallclock_hour.
84879 A wallclock_hour may be any of the single digits [1,9], or may
84880 be double digits, ranging from [01,12]. If a wallclock_hr_min
84881 is a four-digit number, the first two digits shall be a valid
84882 wallclock_hour, while the last two represent the number of
84883 minutes, from [00,59].
84884 */
```

```
84885 %token minute
84886 /*
84887 A minute is a one or two-digit number whose value can be [0,9]
84888 or [00,59].
84889 */
```

```

84890 %token day_number
84891 /*
84892     A day_number is a number in the range appropriate for the particular
84893     month and year specified by month_name and year_number, respectively.
84894     If no year_number is given, the current year is assumed if the given
84895     date and time are later this year. If no year_number is given and
84896     the date and time have already occurred this year and the month is
84897     not the current month, next year is the assumed year.
84898 */

84899 %token year_number
84900 /*
84901     A year_number is a four-digit number representing the year A.D., in
84902     which the at_job is to be run.
84903 */

84904 %token inc_number
84905 /*
84906     The inc_number is the number of times the succeeding increment
84907     period is to be added to the specified date and time.
84908 */

84909 %token timezone_name
84910 /*
84911     The name of an optional timezone suffix to the time field, in an
84912     implementation-defined format.
84913 */

84914 %token month_name
84915 /*
84916     One of the values from the mon or abmon keywords in the LC_TIME
84917     locale category.
84918 */

84919 %token day_of_week
84920 /*
84921     One of the values from the day or abday keywords in the LC_TIME
84922     locale category.
84923 */

84924 %token am_pm
84925 /*
84926     One of the values from the am_pm keyword in the LC_TIME locale
84927     category.
84928 */

84929 %start timespec
84930 %%
84931 timespec      : time
84932                | time date
84933                | time increment
84934                | time date increment
84935                | nowspec
84936                ;
84937 nowspec      : "now"

```

```

84938         | "now" increment
84939         ;

84940     time      : hr24clock_hr_min
84941         | hr24clock_hr_min timezone_name
84942         | hr24clock_hour ":" minute
84943         | hr24clock_hour ":" minute timezone_name
84944         | wallclock_hr_min am_pm
84945         | wallclock_hr_min am_pm timezone_name
84946         | wallclock_hour ":" minute am_pm
84947         | wallclock_hour ":" minute am_pm timezone_name
84948         | "noon"
84949         | "midnight"
84950         ;

84951     date      : month_name day_number
84952         | month_name day_number "," year_number
84953         | day_of_week
84954         | "today"
84955         | "tomorrow"
84956         ;

84957     increment : "+" inc_number inc_period
84958         | "next" inc_period
84959         ;

84960     inc_period : "minute" | "minutes"
84961         | "hour" | "hours"
84962         | "day" | "days"
84963         | "week" | "weeks"
84964         | "month" | "months"
84965         | "year" | "years"
84966         ;

```

#### 84967 STDIN

84968 The standard input shall be a text file consisting of commands acceptable to the shell command  
84969 language described in [Chapter 2](#) (on page 2472). The standard input shall only be used if no `-f`  
84970 `file` option is specified.

#### 84971 INPUT FILES

84972 See the STDIN section.

84973 XSI The text files `at.allow` and `at.deny`, which are located in an implementation-defined directory,  
84974 shall contain zero or more user names, one per line, of users who are, respectively, authorized or  
84975 denied access to the `at` and `batch` utilities.

#### 84976 ENVIRONMENT VARIABLES

84977 The following environment variables shall affect the execution of `at`:

84978 `LANG` Provide a default value for the internationalization variables that are unset or null.  
84979 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
84980 variables used to determine the values of locale categories.)

84981 `LC_ALL` If set to a non-empty string value, override the values of all the other  
84982 internationalization variables.

84983		<b>LC_CTYPE</b>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
84984			
84985			
84986		<b>LC_MESSAGES</b>	
84987			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
84988			
84989			
84990	XSI	<b>NLSPATH</b>	Determine the location of messages objects and message catalogs.
84991		<b>LC_TIME</b>	Determine the format and contents for date and time strings written and accepted by <i>at</i> .
84992			
84993		<b>SHELL</b>	Determine a name of a command interpreter to be used to invoke the <i>at</i> -job. If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; or any of the preceding accompanied by a warning diagnostic about which was chosen.
84994			
84995			
84996			
84997			
84998		<b>TZ</b>	Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <i>-t time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it shall override <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
84999			
85000			
85001			
85002			
85003		<b>ASYNCHRONOUS EVENTS</b>	
85004			Default.
85005		<b>STDOUT</b>	
85006			When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.
85007			
85008			In the POSIX locale, the following shall be written to the standard output for each job when jobs are listed in response to the <i>-l</i> option:
85009			
85010			<code>"%s\t%s\n", at_job_id, &lt;date&gt;</code>
85011			where <i>date</i> shall be equivalent in format to the output of:
85012			<code>date +"%a %b %e %T %Y"</code>
85013			The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the <i>TZ</i> variable).
85014			
85015		<b>STDERR</b>	
85016			In the POSIX locale, the following shall be written to standard error when a job has been successfully submitted:
85017			
85018			<code>"job %s at %s\n", at_job_id, &lt;date&gt;</code>
85019			where <i>date</i> has the same format as that described in the STDOUT section. Neither this, nor warning messages concerning the selection of the command interpreter, shall be considered a diagnostic that changes the exit status.
85020			
85021			
85022			Diagnostic messages, if any, shall be written to standard error.



85023 **OUTPUT FILES**

85024 None.

85025 **EXTENDED DESCRIPTION**

85026 None.

85027 **EXIT STATUS**

85028 The following exit values shall be returned:

85029 0 Neither the `-l` option nor the `-r` option was specified and a job was successfully submitted;  
 85030 or, the `-l` option was specified with no `at_job_id` operands and there were no jobs to be  
 85031 listed; or, the `-l` option was specified and all job listings were successfully output; or, the `-r`  
 85032 option was specified and all of the specified jobs were successfully removed.

85033 &gt;0 An error occurred.

85034 **CONSEQUENCES OF ERRORS**

85035 If neither the `-l` option nor the `-r` option was specified, the job shall not be scheduled.  
 85036 Otherwise, the default actions specified in [Section 1.4](#) (on page 2462) apply.

85037 **APPLICATION USAGE**

85038 The format of the `at` command line shown here is guaranteed only for the POSIX locale. Other  
 85039 cultures may be supported with substantially different interfaces, although implementations are  
 85040 encouraged to provide comparable levels of functionality.

85041 Since the commands run in a separate shell invocation, running in a separate process group with  
 85042 no controlling terminal, open file descriptors, traps, and priority inherited from the invoking  
 85043 environment are lost.

85044 Some implementations do not allow substitution of different shells using `SHELL`. System V  
 85045 systems, for example, have used the login shell value for the user in `/etc/passwd`. To select  
 85046 reliably another command interpreter, the user must include it as part of the script, such as:

```
85047 $ at 1800
85048 myshell myscript
85049 EOT
85050 job ... at ...
85051 $
```

85052 **EXAMPLES**

85053 1. This sequence can be used at a terminal:

```
85054 at -m 0730 tomorrow
85055 sort < file >outfile
85056 EOT
```

85057 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
85058 command procedure (the sequence of output redirection specifications is significant):

```
85059 at now + 1 hour <<!
85060 diff file1 file2 2>&1 >outfile | mailx -s "outfile update" mygroup
85061 !
```

85062 Note that this always sends mail when there has been an attempt to update **outfile** and  
 85063 the body of the message will be empty unless an error occurred.

85064 3. The following shows how to capture both standard error and standard output:

```
85065 at now + 1 hour <<EOF
```

```

85066     {
85067         run-batch-processing |
85068             mailx -s "batch processing output" mygroup
85069     } 2>&1 | mailx -E -s "errors during batch processing" mygroup
85070 EOF

```

85071 4. To have a job reschedule itself, *at* can be invoked from within the at-job. For example, this  
 85072 daily processing script named **my.daily** runs every day (although *crontab* is a more  
 85073 appropriate vehicle for such work):

```

85074 # my.daily runs every day
85075 daily processing
85076 at now tomorrow < my.daily

```

85077 5. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as  
 85078 there are no ambiguities. Examples of various times and operand presentation include:

```

85079 at 0815am Jan 24
85080 at 8 :15amjan24
85081 at now "+ 1day"
85082 at 5 pm FRIday
85083 at '17
85084     utc+
85085     30minutes'

```

## 85086 RATIONALE

85087 The *at* utility reads from standard input the commands to be executed at a later time. It may be  
 85088 useful to redirect standard output and standard error within the specified commands.

85089 The *-t time* option was added as a new capability to support an internationalized way of  
 85090 specifying a time for execution of the submitted job.

85091 Early proposals added a “jobname” concept as a way of giving submitted jobs names that are  
 85092 meaningful to the user submitting them. The historical, system-specified *at\_job\_id* gives no  
 85093 indication of what the job is. Upon further reflection, it was decided that the benefit of this was  
 85094 not worth the change in historical interface.

85095 The *-q* option historically has been an undocumented option, used mainly by the *batch* utility.

85096 The System V *-m* option was added to provide a method for informing users that an at-job had  
 85097 completed. Otherwise, users are only informed when output to standard error or standard  
 85098 output are not redirected.

85099 The behavior of *at <now>* was changed in an early proposal from being unspecified to  
 85100 submitting a job for potentially immediate execution. Historical BSD *at* implementations support  
 85101 this. Historical System V implementations give an error in that case, but a change to the System  
 85102 V versions should have no backwards-compatibility ramifications.

85103 On BSD-based systems, a *-u user* option has allowed those with appropriate privileges to access  
 85104 the work of other users. Since this is primarily a system administration feature and is not  
 85105 universally implemented, it has been omitted. Similarly, a specification for the output format for  
 85106 a user with appropriate privileges viewing the queues of other users has been omitted.

85107 The *-f file* option from System V is used instead of the BSD method of using the last operand as  
 85108 the pathname. The BSD method is ambiguous—does:

```
85109 at 1200 friday
```

85110 mean the same thing if there is a file named **friday** in the current directory?

85111 The *at\_job\_id* is composed of a limited character set in historical practice, and it is mandated here  
 85112 to invalidate systems that might try using characters that require shell quoting or that could not  
 85113 be easily parsed by shell scripts.

85114 The *at* utility varies between System V and BSD systems in the way timezones are used. On  
 85115 System V systems, the *TZ* variable affects the at-job submission times and the times displayed  
 85116 for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved  
 85117 with the current specification. If the user wishes to have the timezone default to that of the  
 85118 system, they merely need to issue the *at* command immediately following an unsetting or null  
 85119 assignment to *TZ*. For example:

85120 `TZ= at noon . . .`

85121 gives the desired BSD result.

85122 While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with  
 85123 respect to the digit strings, a lexical analyzer would probably be written to look for and return  
 85124 digit strings in those cases. The parser could then check whether the digit string returned is a  
 85125 valid *day\_number*, *year\_number*, and so on, based on the context.

#### 85126 FUTURE DIRECTIONS

85127 None.

#### 85128 SEE ALSO

85129 *batch*, *crontab*

85130 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

#### 85131 CHANGE HISTORY

85132 First released in Issue 2.

#### 85133 Issue 6

85134 This utility is marked as part of the User Portability Utilities option.

85135 The following new requirements on POSIX implementations derive from alignment with the  
 85136 Single UNIX Specification:

- 85137 • If `-m` is not used, the job's standard output and standard error are provided to the user by  
 85138 mail.

85139 The effects of using the `-q` and `-t` options as defined in the IEEE P1003.2b draft standard are  
 85140 specified.

85141 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 85142 Issue 7

85143 The *at* utility is moved from the User Portability Utilities option to the Base. User Portability  
 85144 Utilities is now an option for interactive utilities.

85145 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
 85146 by the *at* utility.

85147 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

#### 85148 Issue 8

85149 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

85150 Austin Group Defect 1307 is applied, changing the *timespec* operand in relation to locales that do  
 85151 not support the 12-hour clock format.

85152 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

- 85153 Austin Group Defect 1368 is applied, changing the EXAMPLES section.
- 85154 Austin Group Defect 1377 is applied, correcting a typographic error in the description of the `-q`  
85155 option.
- 85156 Austin Group Defect 1495 is applied, changing the EXIT STATUS and CONSEQUENCES OF  
85157 ERRORS sections.

85158 **NAME**

85159 awk — pattern scanning and processing language

85160 **SYNOPSIS**85161 awk [-F *sepstring*] [-v *assignment*]... *program* [*argument*...]85162 awk [-F *sepstring*] -f *progfile* [-f *progfile*]... [-v *assignment*]...85163 [*argument*...]85164 **DESCRIPTION**

85165 The *awk* utility shall execute programs written in the *awk* programming language, which is  
 85166 specialized for textual data manipulation. An *awk* program is a sequence of patterns and  
 85167 corresponding actions. When input is read that matches a pattern, the action associated with that  
 85168 pattern is carried out.

85169 Input shall be interpreted as a sequence of records. By default, a record is a line, less its  
 85170 terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of  
 85171 input shall be matched in turn against each pattern in the program. For each pattern matched,  
 85172 the associated action shall be executed.

85173 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field  
 85174 is a string of non-<blank> non-<newline> characters. This default <blank> and <newline> field  
 85175 delimiter can be changed by using the **FS** built-in variable or the **-F *sepstring*** option. The *awk*  
 85176 utility shall denote the first field in a record \$1, the second \$2, and so on. The symbol \$0 shall  
 85177 refer to the entire record; setting any other field causes the re-evaluation of \$0. Assigning to \$0  
 85178 shall reset the values of all other fields and the **NF** built-in variable.

85179 **OPTIONS**85180 The *awk* utility shall conform to XBD [Section 12.2](#) (on page 215).

85181 The following options shall be supported:

85182 **-F *sepstring*** Define the input field separator. This option shall be equivalent to:85183 **-v *FS=sepstring***

85184 except that if **-F *sepstring*** and **-v *FS=sepstring*** are both used, it is unspecified  
 85185 whether the **FS** assignment resulting from **-F *sepstring*** is processed in command  
 85186 line order or is processed after the last **-v *FS=sepstring***. See the description of the  
 85187 **FS** built-in variable, and how it is used, in the EXTENDED DESCRIPTION section.

85188 **-f *progfile*** Specify the pathname of the file *progfile* containing an *awk* program. A pathname of  
 85189 '-' shall denote the standard input. If multiple instances of this option are  
 85190 specified, the concatenation of the files specified as *progfile* in the order specified  
 85191 shall be the *awk* program. The *awk* program can alternatively be specified in the  
 85192 command line as a single argument.

85193 **-v *assignment***

85194 The application shall ensure that the *assignment* argument is in the same form as an  
 85195 *assignment* operand. The specified variable assignment shall occur prior to  
 85196 executing the *awk* program, including the actions associated with **BEGIN** patterns  
 85197 (if any). Multiple occurrences of this option can be specified.

85198 **OPERANDS**

85199 The following operands shall be supported:

85200 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*  
 85201 program. The application shall supply the *program* operand as a single argument to  
 85202 *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.

85203 *argument* Either of the following two types of *argument* can be intermixed:

85204 *file* A pathname of a file that contains the input to be read, which is  
 85205 matched against the set of patterns in the program. If no *file* operands  
 85206 or their equivalents, achieved by modifying the *awk* variables **ARGV**  
 85207 and **ARGC**, are specified, or if a *file* operand is '-', the standard  
 85208 input shall be used.

85209 *assignment* An operand that begins with an <underscore> or alphabetic  
 85210 character from the portable character set (see the table in XBD [Section](#)  
 85211 [6.1](#), on page 117), followed by a sequence of underscores, digits, and  
 85212 alphabetic characters from the portable character set, followed by the '='  
 85213 character, shall specify a variable assignment rather than a pathname.  
 85214 The characters before the '=' represent the name of an *awk* variable;  
 85215 if that name is an *awk* reserved word (see [Grammar](#), on page 2624)  
 85216 the behavior is undefined. The characters following the <equals-  
 85217 sign> shall be interpreted as if they appeared in the *awk* program  
 85218 preceded and followed by a double-quote ('"') character, as a  
 85219 **STRING** token (see [Grammar](#), on page 2624), except that if the last  
 85220 character is an unescaped <backslash>, it shall be interpreted as a  
 85221 literal <backslash> rather than as the first character of the sequence  
 85222 "\\". The variable shall be assigned the value of that **STRING**  
 85223 token and, if appropriate, shall be considered a *numeric string* (see  
 85224 [Expressions in awk](#), on page 2608), the variable shall also be assigned  
 85225 its numeric value. Each such variable assignment shall occur just  
 85226 prior to the processing of the following *file*, if any. Thus, an  
 85227 assignment before the first *file* argument shall be executed after the  
 85228 **BEGIN** actions (if any), while an assignment after the last *file*  
 85229 argument shall occur before the **END** actions (if any). If there are no  
 85230 *file* arguments or their equivalents, achieved by modifying the *awk*  
 85231 variables **ARGV** and **ARGC**, assignments shall be executed before  
 85232 processing the standard input.

### 85233 STDIN

85234 The standard input shall be used only if no *file* operands or their equivalents, achieved by  
 85235 modifying the *awk* variables **ARGV** and **ARGC**, are specified; or if a *file* operand, or its  
 85236 equivalent, is '-'; or if a *progfile* option-argument is '-'; see the INPUT FILES section. If the  
 85237 *awk* program contains no actions and no patterns, but is otherwise a valid *awk* program,  
 85238 standard input and any *file* operands shall not be read and *awk* shall exit with a return status of  
 85239 zero.

### 85240 INPUT FILES

85241 Input files to the *awk* program from any of the following sources shall be text files:

- 85242 • Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV**  
 85243 and **ARGC**
- 85244 • Standard input in the absence of any *file* operands, or their equivalents
- 85245 • Arguments to the **getline** function

85246 Whether the variable **RS** is set to a value other than a <newline> or not, for these files,  
 85247 implementations shall support records terminated with the specified separator up to  
 85248 {LINE\_MAX} bytes and may support longer records.

85249 If **-f progfile** is specified, the application shall ensure that the files named by each of the *progfile*

85250 option-arguments are text files and their concatenation, in the same order as they appear in the  
85251 arguments, is an *awk* program.

#### 85252 ENVIRONMENT VARIABLES

85253 The following environment variables shall affect the execution of *awk*:

85254 *LANG* Provide a default value for the internationalization variables that are unset or null.  
85255 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
85256 variables used to determine the values of locale categories.)

85257 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
85258 internationalization variables.

#### 85259 *LC\_COLLATE*

85260 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
85261 character collating elements within regular expressions and in comparisons of  
85262 string values.

85263 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
85264 characters (for example, single-byte as opposed to multi-byte characters in  
85265 arguments and input files), the behavior of character classes within regular  
85266 expressions, the identification of characters as letters, and the mapping of  
85267 uppercase and lowercase characters for the **toupper** and **tolower** functions.

#### 85268 *LC\_MESSAGES*

85269 Determine the locale that should be used to affect the format and contents of  
85270 diagnostic messages written to standard error.

#### 85271 *LC\_NUMERIC*

85272 Determine the radix character used when interpreting numeric input, performing  
85273 conversions between numeric and string values, and formatting numeric output.  
85274 Regardless of locale, the <period> character (the decimal-point character of the  
85275 POSIX locale) is the decimal-point character recognized in processing *awk*  
85276 programs (including assignments in command line arguments).

85277 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

85278 *PATH* Determine the search path when looking for commands executed by *system(expr)*,  
85279 or input and output pipes; see XBD Chapter 8 (on page 167).

85280 In addition, all environment variables shall be visible via the *awk* variable **ENVIRON**.

#### 85281 ASYNCHRONOUS EVENTS

85282 Default.

#### 85283 STDOUT

85284 The nature of the output files depends on the *awk* program.

#### 85285 STDERR

85286 The standard error shall be used only for diagnostic messages.

#### 85287 OUTPUT FILES

85288 The nature of the output files depends on the *awk* program.

## 85289 EXTENDED DESCRIPTION

## 85290 Overall Program Structure

85291 An *awk* program is composed of pairs of the form:85292 *pattern { action }*

85293 Either the pattern or the action (including the enclosing brace characters) can be omitted.

85294 A missing pattern shall match any record of input, and a missing action shall be equivalent to:

85295 *{ print }*

85296 Execution of the *awk* program shall start by first executing the actions associated with all **BEGIN**  
 85297 patterns in the order they occur in the program. Then each *file* operand (or standard input if no  
 85298 files were specified) shall be processed in turn by reading data from the file until a record  
 85299 separator is seen (<newline> by default). Before the first reference to a field in the record is  
 85300 evaluated, the record shall be split into fields, according to the rules in [Regular Expressions](#) (on  
 85301 page 2615), using the value of **FS** that was current at the time the record was read. Each pattern  
 85302 in the program then shall be evaluated in the order of occurrence, and the action associated with  
 85303 each pattern that matches the current record executed. The action for a matching pattern shall be  
 85304 executed before evaluating subsequent patterns. Finally, the actions associated with all **END**  
 85305 patterns shall be executed in the order they occur in the program.

## 85306 Expressions in awk

85307 Expressions describe computations used in *patterns* and *actions*. In the following table, valid  
 85308 expression operations are given in groups from highest precedence first to lowest precedence  
 85309 last, with equal-precedence operators grouped between horizontal lines. In expression  
 85310 evaluation, where the grammar is formally ambiguous, higher precedence operators shall be  
 85311 evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent  
 85312 any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side  
 85313 of an assignment operator). The precise syntax of expressions is given in [Grammar](#) (on page  
 85314 2624).

85315 **Table 3-1** Expressions in Decreasing Precedence in *awk*

Syntax	Name	Type of Result	Associativity
( <i>expr</i> )	Grouping	Type of <i>expr</i>	N/A
<i>\$expr</i>	Field reference	Uninitialized or String	N/A
<i>lvalue</i> ++	Post-increment	Numeric	N/A
<i>lvalue</i> --	Post-decrement	Numeric	N/A
++ <i>lvalue</i>	Pre-increment	Numeric	N/A
-- <i>lvalue</i>	Pre-decrement	Numeric	N/A
<i>expr</i> ^ <i>expr</i>	Exponentiation	Numeric	Right
! <i>expr</i>	Logical not	Numeric	N/A
+ <i>expr</i>	Unary plus	Numeric	N/A
- <i>expr</i>	Unary minus	Numeric	N/A
<i>expr</i> * <i>expr</i>	Multiplication	Numeric	Left
<i>expr</i> / <i>expr</i>	Division	Numeric	Left
<i>expr</i> % <i>expr</i>	Modulus	Numeric	Left



	Syntax	Name	Type of Result	Associativity
85330				
85331	$expr + expr$	Addition	Numeric	Left
85332	$expr - expr$	Subtraction	Numeric	Left
85333	$expr expr$	String concatenation	String	Left
85334	$expr < expr$	Less than	Numeric	None
85335	$expr <= expr$	Less than or equal to	Numeric	None
85336	$expr != expr$	Not equal to	Numeric	None
85337	$expr == expr$	Equal to	Numeric	None
85338	$expr > expr$	Greater than	Numeric	None
85339	$expr >= expr$	Greater than or equal to	Numeric	None
85340	$expr \sim expr$	ERE match	Numeric	None
85341	$expr !\sim expr$	ERE non-match	Numeric	None
85342	$expr$ in array	Array membership	Numeric	Left
85343	( $index$ ) in array	Multi-dimension array membership	Numeric	Left
85344				
85345	$expr \&\& expr$	Logical AND	Numeric	Left
85346	$expr \ \  expr$	Logical OR	Numeric	Left
85347	$expr1 ? expr2 : expr3$	Conditional expression	Type of selected $expr2$ or $expr3$	Right
85348				
85349	$lvalue \hat{=} expr$	Exponentiation assignment	Numeric	Right
85350	$lvalue \% = expr$	Modulus assignment	Numeric	Right
85351	$lvalue * = expr$	Multiplication assignment	Numeric	Right
85352	$lvalue / = expr$	Division assignment	Numeric	Right
85353	$lvalue + = expr$	Addition assignment	Numeric	Right
85354	$lvalue - = expr$	Subtraction assignment	Numeric	Right
85355	$lvalue = expr$	Assignment	Type of $expr$	Right

85356 Each expression shall have either a string value, a numeric value, or both. Except as stated for  
 85357 specific contexts, the value of an expression shall be implicitly converted to the type needed for  
 85358 the context in which it is used. A string value shall be converted to a numeric value either by the  
 85359 equivalent of the following calls to functions defined by the ISO C standard:

```
85360 setlocale(LC_NUMERIC, "");
85361 numeric_value = atof(string_value);
```

85362 or by converting the initial portion of the string to type **double** representation as follows:

85363 The input string is decomposed into two parts: an initial, possibly empty, sequence of  
 85364 white-space characters (as specified by *isspace()*) and a subject sequence interpreted as a  
 85365 floating-point constant.

85366 The expected form of the subject sequence is an optional '+' or '-' sign, then a non-  
 85367 empty sequence of digits optionally containing a radix character, then an optional  
 85368 exponent part. An exponent part consists of 'e' or 'E', followed by an optional sign,  
 85369 followed by one or more decimal digits.

85370 The sequence starting with the first digit or the radix character (whichever occurs first) is  
 85371 interpreted as a floating constant of the C language, except that the radix character shall be  
 85372 used in place of a <period>, and if neither an exponent part nor a radix character appears,  
 85373 a radix character is assumed to follow the last digit in the string. If the subject sequence  
 85374 begins with a <hyphen-minus>, the value resulting from the conversion is negated.

85375 A numeric value that is exactly equal to the value of an integer (see [Section 1.1.2](#), on page 2457)

85376 shall be converted to a string by the equivalent of a call to the **sprintf** function (see [String](#)  
 85377 [Functions](#), on page 2621) with the string "%d" as the *fmt* argument and the numeric value being  
 85378 converted as the first and only *expr* argument. Any other numeric value shall be converted to a  
 85379 string by the equivalent of a call to the **sprintf** function with the value of the variable  
 85380 **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only  
 85381 *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a  
 85382 floating-point format specification. This volume of POSIX.1-2024 specifies no explicit  
 85383 conversions between numbers and strings. An application can force an expression to be treated  
 85384 as a number by adding zero to it, or can force it to be treated as a string by concatenating the null  
 85385 string (" ") to it.

85386 A string value shall be considered a *numeric string* if it comes from one of the following:

- 85387 1. Field variables
- 85388 2. Input from the *getline()* function
- 85389 3. **FILENAME**
- 85390 4. **ARGV** array elements
- 85391 5. **ENVIRON** array elements
- 85392 6. Array elements created by the *split()* function
- 85393 7. A command line variable assignment
- 85394 8. Variable assignment from another numeric string variable

85395 and an implementation-dependent condition corresponding to either case (a) or (b) below is  
 85396 met.

- 85397 a. After the equivalent of the following calls to functions defined by the ISO C standard,  
 85398 *string\_value\_end* would differ from *string\_value*, and any characters before the terminating  
 85399 null character in *string\_value\_end* would be <blank> characters:

```
85400 char *string_value_end;  
85401 setlocale(LC_NUMERIC, "");  
85402 numeric_value = strtod(string_value, &string_value_end);
```

- 85403 b. After all the following conversions have been applied, the resulting string would lexically  
 85404 be recognized as a **NUMBER** token as described by the lexical conventions in [Grammar](#)  
 85405 (on page 2624):

- 85406 — All leading and trailing <blank> characters are discarded.
- 85407 — If the first non-<blank> is '+' or '-', it is discarded.
- 85408 — Each occurrence of the radix character from the current locale is changed to a  
 85409 <period>.

85410 In case (a) the numeric value of the *numeric string* shall be the value that would be returned by  
 85411 the *strtod()* call. In case (b) if the first non-<blank> is '-', the numeric value of the *numeric*  
 85412 *string* shall be the negation of the numeric value of the recognized **NUMBER** token; otherwise,  
 85413 the numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER**  
 85414 token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that  
 85415 term is used in this section.

85416 When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall  
 85417 be treated as false and any other value shall be treated as true. Otherwise, a string value of the  
 85418 null string shall be treated as false and any other value shall be treated as true. A Boolean

85419 context shall be one of the following:

- 85420 • The first subexpression of a conditional expression
- 85421 • An expression operated on by logical NOT, logical AND, or logical OR
- 85422 • The second expression of a **for** statement
- 85423 • The expression of an **if** statement
- 85424 • The expression of the **while** clause in either a **while** or **do...while** statement
- 85425 • An expression used as a pattern (as in Overall Program Structure)

85426 All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C  
85427 standard (see [Section 1.1.2](#), on page 2457).

85428 The value of the expression:

85429 `expr1 ^ expr2`

85430 shall be equivalent to the value returned by the ISO C standard function call:

85431 `pow(expr1, expr2)`

85432 The expression:

85433 `lvalue ^= expr`

85434 shall be equivalent to the ISO C standard expression:

85435 `lvalue = pow(lvalue, expr)`

85436 except that `lvalue` shall be evaluated only once. The value of the expression:

85437 `expr1 % expr2`

85438 shall be equivalent to the value returned by the ISO C standard function call:

85439 `fmod(expr1, expr2)`

85440 The expression:

85441 `lvalue %= expr`

85442 shall be equivalent to the ISO C standard expression:

85443 `lvalue = fmod(lvalue, expr)`

85444 except that `lvalue` shall be evaluated only once.

85445 Variables and fields shall be set by the assignment statement:

85446 `lvalue = expression`

85447 and the type of *expression* shall determine the resulting variable type. The assignment includes  
85448 the arithmetic assignments ("`+=`", "`-=`", "`*=`", "`/=`", "`%=`", "`^=`", "`++`", "`--`") all of which  
85449 shall produce a numeric result. The left-hand side of an assignment and the target of increment  
85450 and decrement operators can be one of a variable, an array with index, or a field selector.

85451 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not  
85452 be declared. They shall initially be empty, and their sizes shall change dynamically. The  
85453 subscripts, or element identifiers, are strings, providing a type of associative array capability. An  
85454 array name followed by a subscript within square brackets can be used as an lvalue and thus as  
85455 an expression, as described in the grammar; see [Grammar](#) (on page 2624). Unsubscripted array  
85456 names can be used in only the following contexts:

- 85457 • A parameter in a function definition or function call
- 85458 • The **NAME** token following any use of the keyword **in** as specified in the grammar (see
- 85459 [Grammar](#), on page 2624); if the name used in this context is not an array name, the
- 85460 behavior is undefined
- 85461 • The **NAME** token following the keyword **Delete** without a subscript as specified in the
- 85462 grammar (see [Grammar](#), on page 2624); if the name used in this context is not an array
- 85463 name, the behavior is undefined.

85464 A valid array *index* shall consist of one or more <comma>-separated expressions, similar to the  
 85465 way in which multi-dimensional arrays are indexed in some programming languages. Because  
 85466 *awk* arrays are really one-dimensional, such a <comma>-separated list shall be converted to a  
 85467 single string by concatenating the string values of the separate expressions, each separated from  
 85468 the other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be  
 85469 equivalent:

```
85470 var[expr1, expr2, ... exprn]
```

```
85471 var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]
```

85472 The application shall ensure that a multi-dimensioned *index* used with the **in** operator is  
 85473 parenthesized. The **in** operator, which tests for the existence of a particular array element, shall  
 85474 not cause that element to exist. Any other reference to a nonexistent array element shall  
 85475 automatically create it.

85476 Comparisons (with the '<', '<=', '!=', '==', '>', and '>=' operators) shall be made  
 85477 numerically:

- 85478 • if both operands are numeric,
- 85479 • if one is numeric and the other has a string value that is a numeric string,
- 85480 • if both have string values that are numeric strings, or
- 85481 • if one is numeric and the other has the uninitialized value.

85482 Otherwise, operands shall be converted to strings as required and a string comparison shall be  
 85483 made as follows:

- 85484 • For the '!=' and '==' operators, the strings shall be compared to check if they are  
 85485 identical (not to check if they collate equally).
- 85486 • For the other operators, the strings shall be compared using the locale-specific collation  
 85487 sequence.

85488 The value of the comparison expression shall be 1 if the relation is true, or 0 if the relation is  
 85489 false.

## 85490 Variables and Special Variables

85491 Variables can be used in an *awk* program by referencing them. With the exception of function  
 85492 parameters (see [User-Defined Functions](#), on page 2624), they are not explicitly declared.  
 85493 Function parameter names shall be local to the function; all other variable names shall be global.  
 85494 The same name shall not be used as both a function parameter name and as the name of a  
 85495 function or a special *awk* variable. The same name shall not be used both as a variable name with  
 85496 global scope and as the name of a function. The same name shall not be used within the same  
 85497 scope both as a scalar variable and as an array. Uninitialized variables, including scalar  
 85498 variables, array elements, and field variables, shall have an uninitialized value. An uninitialized  
 85499 value shall have both a numeric value of zero and a string value of the empty string. Evaluation

85500 of variables with an uninitialized value, to either string or numeric, shall be determined by the  
85501 context in which they are used.

85502 Field variables shall be designated by a '\$' followed by a number or numerical expression. The  
85503 effect of the field number *expression* evaluating to anything other than a non-negative integer is  
85504 unspecified; uninitialized variables or string values need not be converted to numeric values in  
85505 this context. New field variables can be created by assigning a value to them. References to  
85506 nonexistent fields (that is, fields after \$NF), shall evaluate to the uninitialized value. Such  
85507 references shall not create new fields. However, assigning to a nonexistent field (for example,  
85508 \$(NF+2)=5) shall increase the value of NF; create any intervening fields with the uninitialized  
85509 value; and cause the value of \$0 to be recomputed, with the fields being separated by the value  
85510 of OFS. Each field variable shall have a string value or an uninitialized value when created.  
85511 Field variables shall have the uninitialized value when created from \$0 using FS and the variable  
85512 does not contain any characters. If appropriate, the field variable shall be considered a numeric  
85513 string (see [Expressions in awk](#), on page 2608).

85514 Implementations shall support the following other special variables that are set by *awk*:

85515 **ARGC** A number determining when the iteration described for ARGV stops. When an  
85516 *awk* program starts, ARGC shall be initialized to the number of elements in the  
85517 ARGV array. ARGC can be updated by the *awk* program and by assignment  
85518 operands. If ARGC is set to a value less than 1, the behavior is unspecified. It is  
85519 unspecified whether alterations to ARGC can be made using the -v option.

85520 **ARGV** An array containing, initially, the command name (see [Section 2.9.1](#), on page 2500)  
85521 used to invoke *awk* in ARGV[0] and the command line arguments, if any,  
85522 excluding options and the *program* operand, in ARGV[1] through ARGV[ARGC-1].  
85523 The elements in ARGV can be assigned new values or deleted, and new elements  
85524 can be added. Note that alterations to ARGV cannot be made using either the  
85525 *assignment* operand or the -v option, because an operand with a '[' before '=' is  
85526 treated as a *file* operand, not an *assignment* operand, and applications are required  
85527 to ensure that the -v option-argument has the same form as an *assignment* operand.  
85528 (See the OPTIONS and OPERANDS sections.)

85529 After processing the BEGIN actions, if any, *awk* begins iterating over the elements  
85530 of ARGV, processing them as if they were *argument* operands. It shall behave as if  
85531 the implementation maintains an internal counter that is initialized to 1 and  
85532 increments by 1 at the end of each iteration. For each iteration, the following shall  
85533 occur:

- 85534 • If the internal counter is greater than or equal to the current value of ARGC  
85535 and no *file* operands have been processed, *awk* shall set FILENAME to '-'  
85536 and process standard input as if it was given as a file operand. The internal  
85537 counter shall not be incremented at the end of this iteration.
- 85538 • Otherwise, if the internal counter is greater than or equal to the current value  
85539 of ARGC, the iterations shall stop and processing of the END actions, if any,  
85540 shall begin. Any ARGV elements with index values greater than or equal to  
85541 ARGC shall not be processed as *argument* operands.
- 85542 • Otherwise, if the element ARGV[*internal counter value*] does not exist, it is  
85543 unspecified whether that element is created. No other action shall be taken.
- 85544 • Otherwise, if ARGV[*internal counter value*] is a null string, no action shall be  
85545 taken.

85546		• Otherwise, if ARGV[ <i>internal counter value</i> ] matches the format of an
85547		assignment operand (see OPERANDS), awk shall process the assignment.
85548		• Otherwise, ARGV[ <i>internal counter value</i> ] shall be treated as a file operand,
85549		FILENAME shall be set to that value, and the named file, or standard input if
85550		the value is '-', shall be processed as an input file.
85551		Since only non-null elements are processed, setting an element of ARGV to the
85552		null string or deleting it means that it shall not be treated as an argument operand.
85553	<b>CONVFMT</b>	The printf format for converting numbers to strings (except for output statements,
85554		where OFMT is used); "%.6g" by default.
85555	<b>ENVIRON</b>	An array representing the value of the environment, as described in the exec
85556		functions defined in the System Interfaces volume of POSIX.1-2024. The indices of
85557		the array shall be strings consisting of the names of the environment variables, and
85558		the value of each array element shall be a string consisting of the value of that
85559		variable. If appropriate, the environment variable shall be considered a numeric
85560		string (see Expressions in awk, on page 2608); the array element shall also have its
85561		numeric value.
85562		In all cases where the behavior of awk is affected by environment variables
85563		(including the environment of any commands that awk executes via the system
85564		function or via pipeline redirections with the print statement, the printf statement,
85565		or the getline function), the environment used shall be the environment at the time
85566		awk began executing; it is implementation-defined whether any modification of
85567		ENVIRON affects this environment.
85568	<b>FILENAME</b>	The pathname used to open the current input file, or '-' if the file is standard
85569		input. Inside a BEGIN action FILENAME shall be unset. Inside an END action the
85570		value shall be the name of the last input file processed. If an application changes
85571		the value of FILENAME, the results are unspecified.
85572	<b>FNR</b>	The ordinal number of the current record in the current file. Inside a BEGIN action
85573		the value shall be zero. Inside an END action the value shall be the number of the
85574		last record processed in the last file processed.
85575	<b>FS</b>	Input field separator regular expression; a <space> by default.
85576	<b>NF</b>	The number of fields in the current record. Inside a BEGIN action, the use of NF is
85577		undefined unless a getline function without a var argument is executed previously.
85578		Inside an END action, NF shall retain the value it had for the last record read,
85579		unless a subsequent, redirected, getline function without a var argument is
85580		performed prior to entering the END action.
85581	<b>NR</b>	The ordinal number of the current record from the start of input. Inside a BEGIN
85582		action the value shall be zero. Inside an END action the value shall be the number
85583		of the last record processed. Records skipped by the nextfile statement shall not
85584		be included.
85585	<b>OFMT</b>	The printf format for converting numbers to strings in output statements (see
85586		Output Statements, on page 2619); "%.6g" by default. The result of the conversion
85587		is unspecified if the value of OFMT is not a floating-point format specification.
85588	<b>OFS</b>	The print statement output field separator; <space> by default.

85589	<b>ORS</b>	The <b>print</b> statement output record separator; a <newline> by default.
85590	<b>RLENGTH</b>	The length of the string matched by the <b>match</b> function.
85591	<b>RS</b>	The first character of the string value of <b>RS</b> shall be the input record separator; a <newline> by default. If <b>RS</b> contains more than one character, the results are unspecified. If <b>RS</b> is null, then records are separated by sequences consisting of a <newline> plus one or more blank lines, leading or trailing blank lines shall not result in empty records at the beginning or end of the input, and a <newline> shall always be a field separator, no matter what the value of <b>FS</b> is.
85592		
85593		
85594		
85595		
85596		
85597	<b>RSTART</b>	The starting position of the string matched by the <b>match</b> function, numbering from 1. This shall always be equivalent to the return value of the <b>match</b> function.
85598		
85599	<b>SUBSEP</b>	The subscript separator string for multi-dimensional arrays; the default value is implementation-defined.
85600		

### 85601 **Regular Expressions**

85602 The *awk* utility shall make use of the extended regular expression notation (see XBD [Section 9.4](#),  
85603 on page 187) except that it shall allow the use of C-language conventions for escaping special  
85604 characters within the EREs, as specified in the table in XBD [Chapter 5](#) (on page 113) for '\\',  
85605 '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v' and in the following table for other sequences;  
85606 these escape sequences shall be recognized both inside and outside bracket expressions. Note  
85607 that records need not be separated by <newline> characters and string constants can contain  
85608 <newline> characters, so even the "\\n" sequence is valid in *awk* EREs. Using a <slash>  
85609 character within the lexical token **ERE** (except as one of the two delimiters) requires the escaping  
85610 shown in the following table.

85611

Table 3-2 Escape Sequences in *awk*

Escape Sequence	Description	Meaning
\ "	<backslash> <quotation-mark>	In the lexical token <b>STRING</b> , <quotation-mark> character. Otherwise undefined.
\ /	<backslash> <slash>	In the lexical token <b>ERE</b> , <slash> character. Otherwise undefined.
\ ddd	A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined. If the digits produce a value greater than octal 377, the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
\ ., \ [, \ (, \ *, \ +, \ ?, \ {, \  , \ ^, \ \$	A <backslash> character followed by a character that has a special meaning in EREs (see XBD Section 9.4), other than <backslash>.	In the lexical token <b>ERE</b> when not inside a bracket expression, the sequence shall represent itself. Otherwise undefined.
\\	Two <backslash> characters.	In the lexical token <b>ERE</b> , the sequence shall represent itself. In the lexical token <b>STRING</b> , it shall represent a single <backslash>.
\ c	A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 113) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').	Undefined

85640

85641

85642

85643

85644

85645

85646

85647

85648

85649

85650

85651

85652

85653

A regular expression can be matched against a specific field or string by using one of the two regular expression matching operators, '~' and '!~'. These operators shall interpret their right-hand operand as a regular expression and their left-hand operand as a string. If the regular expression matches the string, the '~' expression shall evaluate to a value of 1, and the '!~' expression shall evaluate to a value of 0. (The regular expression matching operation is as defined by the term *matched* in XBD Section 9.1 (on page 179), where a match occurs on any part of the string unless the regular expression is limited with the <circumflex> or <dollar-sign> special characters.) If the regular expression does not match the string, the '~' expression shall evaluate to a value of 0, and the '!~' expression shall evaluate to a value of 1. If the right-hand operand is any expression other than the lexical token **ERE**, the string value of the expression shall be interpreted as an extended regular expression, including the escape conventions described above. Note that these escape conventions shall also be applied in determining the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a string literal is used in this context.

85654

85655

85656

When an **ERE** token appears as an expression in any context other than as the right-hand of the '~' or '!~' operator or as one of the built-in function arguments described below, the value of the resulting expression shall be the equivalent of:

85657

```
$0 ~ /ere/
```



85658 The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function  
 85659 (see [String Functions](#), on page 2621) shall be interpreted as extended regular expressions. These  
 85660 can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner  
 85661 as the right-hand side of the '*~*' or '!~' operator.

85662 An extended regular expression can be used to separate fields by assigning a string containing  
 85663 the expression to the built-in variable **FS**, either directly or as a consequence of using the **-F**  
 85664 *sepstring* option. The default value of the **FS** variable shall be a single `<space>`. The following  
 85665 describes **FS** behavior:

- 85666 1. If **FS** is a null string, the behavior is unspecified.
- 85667 2. If **FS** is a single character:
  - 85668 a. If **FS** is `<space>`, skip leading and trailing `<blank>` and `<newline>` characters;  
 85669 fields shall be delimited by sets of one or more `<blank>` or `<newline>` characters.
  - 85670 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single  
 85671 occurrence of *c*.
- 85672 3. Otherwise, the string value of **FS** shall be considered to be an extended regular  
 85673 expression. Each occurrence of a sequence of one or more characters matching the  
 85674 extended regular expression shall delimit fields.

85675 When ERE matching is performed against input records; that is, the match is against \$0 and the  
 85676 current value of \$0 resulted from processing an input record, record separator characters (the  
 85677 first character of the value of the variable **RS**, `<newline>` by default) cannot be embedded in the  
 85678 expression, and no expression shall match the record separator character. If the record separator  
 85679 is not `<newline>`, `<newline>` characters embedded in the expression can be matched. When ERE  
 85680 matching is not performed against input records, it shall be based on text strings; any character  
 85681 (including `<newline>` and the record separator) can be embedded in the pattern, and an  
 85682 appropriate pattern shall match any character. However, in all *awk* ERE matching, the use of one  
 85683 or more NUL characters in the pattern, input record, or text string produces undefined results.

## 85684 **Patterns**

85685 A *pattern* is any valid *expression*, a range specified by two expressions separated by a comma, or  
 85686 one of the two special patterns **BEGIN** or **END**.

## 85687 **Special Patterns**

85688 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern  
 85689 shall be matched once and its associated action executed before the first record of input is read—  
 85690 except possibly by use of the **getline** function (see [Input/Output and General Functions](#), on  
 85691 page 2623) in a prior **BEGIN** action—and before command line assignment is done. Each **END**  
 85692 pattern shall be matched once and its associated action executed after the last record of input has  
 85693 been read, or if there is no further input file to process following a **nextfile** statement. These two  
 85694 patterns shall have associated actions.

85695 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns  
 85696 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order  
 85697 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern  
 85698 in a program.

85699 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action  
 85700 contains no **getline** function, *awk* shall exit without reading its input when the last statement in  
 85701 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern

85702 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the  
85703 statements in the **END** actions are executed.

### 85704 **Expression Patterns**

85705 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the  
85706 result is true, the pattern shall be considered to match, and the associated action (if any) shall be  
85707 executed. If the result is false, the action shall not be executed.

### 85708 **Pattern Ranges**

85709 A pattern range consists of two expressions separated by a comma; in this case, the action shall  
85710 be performed for all records between a match of the first expression and the following match of  
85711 the second expression, inclusive. At this point, the pattern range can be repeated starting at  
85712 input records subsequent to the end of the matched range.

### 85713 **Actions**

85714 An action is a sequence of statements as shown in the grammar in [Grammar](#) (on page 2624).  
85715 Any single statement can be replaced by a statement list enclosed in curly braces. The  
85716 application shall ensure that statements in a statement list are separated by <newline> or  
85717 <semicolon> characters. Statements in a statement list shall be executed sequentially in the order  
85718 that they appear.

85719 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero  
85720 or non-null, the following statement shall be executed; otherwise, if **else** is present, the statement  
85721 following the **else** shall be executed.

85722 The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard  
85723 (see [Section 1.1.2](#), on page 2457), except that the Boolean expressions shall be treated as  
85724 described in [Expressions in awk](#) (on page 2608), and except in the case of:

85725 `for (variable in array)`

85726 which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of  
85727 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**  
85728 statement occurs outside of a loop, the behavior is undefined.

85729 The **delete** statement shall remove either a specified individual array element or, if no element is  
85730 specified, all array elements. Thus, the following code:

85731 `for (index in array)`  
85732  `delete array[index]`

85733 is equivalent to:

85734 `delete array`

85735 Both delete all elements of the array.

85736 The **next** statement shall cause all further processing of the current input record to be  
85737 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or  
85738 **END** action.

85739 The **nextfile** statement shall cause all further processing of the current input file to be  
85740 abandoned. The behavior is undefined if a **nextfile** statement appears or is invoked in a **BEGIN**  
85741 or **END** action, or in a user-defined function.

85742 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program

85743 source and then terminate the program without reading further input. An **exit** statement inside  
 85744 an **END** action shall terminate the program without further execution of **END** actions. If an  
 85745 expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*,  
 85746 unless subsequent errors are encountered or a subsequent **exit** statement with an expression is  
 85747 executed.

## 85748 Output Statements

85749 Both **print** and **printf** statements shall write to standard output by default. The output shall be  
 85750 written to the location specified by *output\_redirection* if one is supplied, as follows:

```
85751 > expression
85752 >> expression
85753 | expression
```

85754 In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into  
 85755 which to write (for '>' or '>>') or as a command to be executed (for '|' or '|'). Using the first two  
 85756 forms, if the file of that name is not currently open, it shall be opened, creating it if necessary  
 85757 and using the first form, truncating the file. The output then shall be appended to the file. As  
 85758 long as the file remains open, subsequent calls in which *expression* evaluates to the same string  
 85759 value shall simply append output to the file. The file remains open until the **close** function (see  
 85760 [Input/Output and General Functions](#), on page 2623) is called with an expression that evaluates  
 85761 to the same string value.

85762 The third form shall write output onto a stream piped to the input of a command. The stream  
 85763 shall be created if no stream is currently open with the value of *expression* as its command name.  
 85764 The stream created shall be equivalent to one created by a call to the *popen()* function defined in  
 85765 the System Interfaces volume of POSIX.1-2024 with the value of *expression* as the *command*  
 85766 argument and a value of *w* as the *mode* argument. As long as the stream remains open,  
 85767 subsequent calls in which *expression* evaluates to the same string value shall write output to the  
 85768 existing stream. The stream shall remain open until the **close** function (see [Input/Output and  
 85769 General Functions](#), on page 2623) is called with an expression that evaluates to the same string  
 85770 value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in  
 85771 the System Interfaces volume of POSIX.1-2024.

85772 As described in detail by the grammar in [Grammar](#) (on page 2624), these output statements shall  
 85773 take a <comma>-separated list of *expressions* referred to in the grammar by the non-terminal  
 85774 symbols **expr\_list**, **print\_expr\_list**, or **print\_expr\_list\_opt**. This list is referred to here as the  
 85775 *expression list*, and each member is referred to as an *expression argument*.

85776 The **print** statement shall write the value of each expression argument onto the indicated output  
 85777 stream separated by the current output field separator (see variable **OFS** above), and terminated  
 85778 by the output record separator (see variable **ORS** above). All expression arguments shall be  
 85779 taken as strings, being converted if necessary; this conversion shall be as described in  
 85780 [Expressions in awk](#) (on page 2608), with the exception that the **printf** format in **OFMT** shall be  
 85781 used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole  
 85782 input record (\$0).

85783 The **printf** statement shall produce output based on a notation similar to the File Format  
 85784 Notation used to describe file formats in this volume of POSIX.1-2024 (see [XBD Chapter 5](#), on  
 85785 page 113). Output shall be produced as specified with the first *expression* argument as the string  
 85786 *format* and subsequent *expression* arguments as the strings *arg1* to *argn*, inclusive, with the  
 85787 following exceptions:

- 85788 1. The *format* shall be an actual character string rather than a graphical representation.  
 85789 Therefore, it cannot contain empty character positions. The <space> in the *format* string,  
 85790 in any context other than a *flag* of a conversion specification, shall be treated as an  
 85791 ordinary character that is copied to the output.
- 85792 2. If the character set contains a ' $\Delta$ ' character and that character appears in the *format*  
 85793 string, it shall be treated as an ordinary character that is copied to the output.
- 85794 3. The *escape sequences* beginning with a <backslash> character shall be treated as sequences  
 85795 of ordinary characters that are copied to the output. Note that these same sequences shall  
 85796 be interpreted lexically by *awk* when they appear in literal strings, but they shall not be  
 85797 treated specially by the **printf** statement.
- 85798 4. A *field width* or *precision* can be specified as the '\*' character instead of a digit string. In  
 85799 this case the next argument from the expression list shall be fetched and its numeric value  
 85800 taken as the field width or precision.
- 85801 5. The implementation shall not precede or follow output from the *d* or *u* conversion  
 85802 specifier characters with <blank> characters not specified by the *format* string.
- 85803 6. The implementation shall not precede output from the *o* conversion specifier character  
 85804 with leading zeros not specified by the *format* string.
- 85805 7. For the *c* conversion specifier character: if the argument has a numeric value, the  
 85806 character whose encoding is that value shall be output. If the value is zero or is not the  
 85807 encoding of any character in the character set, the behavior is undefined. If the argument  
 85808 does not have a numeric value, the first character of the string value shall be output; if the  
 85809 string does not contain any characters, the behavior is undefined.
- 85810 8. For each conversion specification that consumes an argument, the next expression  
 85811 argument shall be evaluated. With the exception of the *c* conversion specifier character,  
 85812 the value shall be converted (according to the rules specified in [Expressions in awk](#), on  
 85813 page 2608) to the appropriate type for the conversion specification.
- 85814 9. If there are insufficient expression arguments to satisfy all the conversion specifications in  
 85815 the *format* string, the behavior is undefined.
- 85816 10. If any character sequence in the *format* string begins with a '%' character, but does not  
 85817 form a valid conversion specification, the behavior is unspecified.

85818 Both **print** and **printf** can output at least {LINE\_MAX} bytes.

## 85819 Functions

85820 The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and  
 85821 general.

85822 Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an  
 85823 array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is  
 85824 passed as a parameter that the function uses as an array. Function parameters shall be passed by  
 85825 value if scalar and by reference if array name.

85826 **Arithmetic Functions**

85827 The arithmetic functions, except for **int**, shall be based on the ISO C standard (see [Section 1.1.2](#),  
 85828 on page 2457). The behavior is undefined in cases where the ISO C standard specifies that an  
 85829 error be returned or that the behavior is undefined. Although the grammar (see [Grammar](#), on  
 85830 page 2624) permits built-in functions to appear with no arguments or parentheses, unless the  
 85831 argument or parentheses are indicated as optional in the following list (by displaying them  
 85832 within the " [ ] " brackets), such use is undefined.

85833 **atan2**(*y,x*) Return arctangent of *y/x* in radians in the range  $[-\pi,\pi]$ .

85834 **cos**(*x*) Return cosine of *x*, where *x* is in radians.

85835 **sin**(*x*) Return sine of *x*, where *x* is in radians.

85836 **exp**(*x*) Return the exponential function of *x*.

85837 **log**(*x*) Return the natural logarithm of *x*.

85838 **sqrt**(*x*) Return the square root of *x*.

85839 **int**(*x*) Return the argument truncated to an integer. Truncation shall be toward 0 when  
 85840 *x*>0.

85841 **rand**() Return a floating point pseudo-random number *n*, such that  $0\leq n<1$ .

85842 **srand**(*expr*) Set the seed value for **rand** to *expr* or use the seconds since the Epoch if *expr* is  
 85843 omitted. The previous seed value shall be returned. The behavior is unspecified if  
 85844 *expr* is not an integer expression or if the value of *expr* is not within the range 0  
 85845 through  $2^{31}-1$  (2 147 483 647), inclusive. The initial seed value is unspecified if **rand**  
 85846 is called without calling **srand** first. The **srand** function uses the argument as a  
 85847 seed for a new sequence of pseudo-random numbers to be returned by subsequent  
 85848 calls to **rand**. If **srand** is then called with the same seed value, the sequence of  
 85849 pseudo-random numbers shall be repeated.

85850 **String Functions**

85851 The string functions in the following list shall be supported. Although the grammar (see  
 85852 [Grammar](#), on page 2624) permits built-in functions to appear with no arguments or parentheses,  
 85853 unless the argument or parentheses are indicated as optional in the following list (by displaying  
 85854 them within the " [ ] " brackets), such use is undefined.

85855 **gsub**(*ere, repl[, in]*)

85856 Behave like **sub** (see below), except that it shall replace all occurrences of the  
 85857 regular expression (like the *ed* utility global substitute) in \$0 or in the *in* argument,  
 85858 when specified.

85859 **index**(*s, t*) Return the position, in characters, numbering from 1, in string *s* where string *t* first  
 85860 occurs, or zero if it does not occur at all.

85861 **length**(*[arg]*)

85862 If *arg* is an array, return the number of elements in the array; otherwise, return the  
 85863 length, in characters, of *arg* taken as a string, or of the whole record, \$0, if there is  
 85864 no argument.

85865 **match**(*s, ere*) Return the position, in characters, numbering from 1, in string *s* where the  
 85866 extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART  
 85867 shall be set to the starting position (which is the same as the returned value), zero  
 85868 if no match is found; RLENGTH shall be set to the length of the matched string, -1

85869 if no match is found.

85870 **split**(*s*, *a*[], *fs* )

85871 Split the string *s* into array elements *a*[1], *a*[2], . . . , *a*[*n*], and return *n*. All elements

85872 of the array shall be deleted before the split is performed. The separation shall be

85873 done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array

85874 element shall have a string value when created and, if appropriate, the array

85875 element shall be considered a numeric string (see [Expressions in awk](#), on page

85876 2608). The effect of a null string as the value of *fs* is unspecified.

85877 **sprintf**(*fmt*, *expr*, *expr*, . . . )

85878 Format the expressions according to the **printf** format given by *fmt* and return the

85879 resulting string.

85880 **sub**(*ere*, *repl*[], *in* )

85881 Substitute the string *repl* in place of the first instance of the extended regular

85882 expression *ERE* in string *in* and return the number of substitutions. An

85883 <ampersand> ('&') appearing in the string *repl* shall be replaced by the string

85884 from *in* that matches the ERE. An <ampersand> preceded with a <backslash> shall

85885 be interpreted as the literal <ampersand> character. An occurrence of two

85886 consecutive <backslash> characters shall be interpreted as just a single literal

85887 <backslash> character. Any other occurrence of a <backslash> (for example,

85888 preceding any other character) shall be treated as a literal <backslash> character.

85889 Note that if *repl* is a string literal (the lexical token **STRING**; see [Grammar](#), on page

85890 2624), the handling of the <ampersand> character occurs after any lexical

85891 processing, including any lexical <backslash>-escape sequence processing. If *in* is

85892 specified and it is not an lvalue (see [Expressions in awk](#), on page 2608), the

85893 behavior is undefined. If *in* is omitted, *awk* shall use the current record (\$0) in its

85894 place.

85895 **substr**(*s*, *m*[], *n* )

85896 Return the at most *n*-character substring of *s* that begins at position *m*, numbering

85897 from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string,

85898 the length of the substring shall be limited by the length of the string *s*.

85899 **tolower**(*s*) Return a string based on the string *s*. Each character in *s* that is an uppercase letter

85900 specified to have a **tolower** mapping by the *LC\_CTYPE* category of the current

85901 locale shall be replaced in the returned string by the lowercase letter specified by

85902 the mapping. Other characters in *s* shall be unchanged in the returned string.

85903 **toupper**(*s*) Return a string based on the string *s*. Each character in *s* that is a lowercase letter

85904 specified to have a **toupper** mapping by the *LC\_CTYPE* category of the current

85905 locale is replaced in the returned string by the uppercase letter specified by the

85906 mapping. Other characters in *s* are unchanged in the returned string.

85907 All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued

85908 expression that is a regular expression as defined in [Regular Expressions](#) (on page 2615).

85909 **Input/Output and General Functions**

85910 The input/output and general functions are:

85911 **close**(*expression*)

85912 Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with  
 85913 the same string-valued *expression*. The limit on the number of open *expression*  
 85914 arguments is implementation-defined. If the close was successful, the function  
 85915 shall return zero; otherwise, it shall return non-zero.

85916 **fflush**([*expression*])

85917 Write any unwritten data to the file or piped stream opened by a **print** or **printf**  
 85918 statement with the same string-valued *expression*. If no argument, or if *expression*  
 85919 evaluates to the null string, then write all such data for all such open files and  
 85920 piped streams, and standard output.

85921 If **fflush** is successful, it shall return 0; otherwise, it shall return non-zero.85922 *expression* | **getline** [*var*]

85923 Read a record of input from a stream piped from the output of a command. The  
 85924 stream shall be created if no stream is currently open with the value of *expression* as  
 85925 its command name. The stream created shall be equivalent to one created by a call  
 85926 to the *popen*(*r*) function with the value of *expression* as the *command* argument and a  
 85927 value of *r* as the *mode* argument. As long as the stream remains open, subsequent  
 85928 calls in which *expression* evaluates to the same string value shall read subsequent  
 85929 records from the stream. The stream shall remain open until the **close** function is  
 85930 called with an expression that evaluates to the same string value. At that time, the  
 85931 stream shall be closed as if by a call to the *pclose*(*r*) function. If *var* is omitted, \$0 and  
 85932 **NR** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 85933 a numeric string (see [Expressions in awk](#), on page 2608).

85934 The **getline** operator can form ambiguous constructs when there are  
 85935 unparenthesized operators (including concatenate) to the left of the '|' (to the  
 85936 beginning of the expression containing **getline**). In the context of the '\$' operator,  
 85937 '|' shall behave as if it had a lower precedence than '\$'. The result of evaluating  
 85938 other operators is unspecified, and conforming applications shall parenthesize  
 85939 properly all such usages.

85940 **getline** Set \$0 to the next input record from the current input file. This form of **getline** shall  
 85941 set the **NR**, **NR**, and **FNR** variables.

85942 **getline** *var* Set variable *var* to the next input record from the current input file and, if  
 85943 appropriate, *var* shall be considered a numeric string (see [Expressions in awk](#), on  
 85944 page 2608). This form of **getline** shall set the **FNR** and **NR** variables.

85945 **getline** [*var*] < *expression*

85946 Read the next record of input from a named file. The *expression* shall be evaluated  
 85947 to produce a string that is used as a pathname. If the file of that name is not  
 85948 currently open, it shall be opened. As long as the stream remains open, subsequent  
 85949 calls in which *expression* evaluates to the same string value shall read subsequent  
 85950 records from the file. The file shall remain open until the **close** function is called  
 85951 with an expression that evaluates to the same string value. If *var* is omitted, \$0 and  
 85952 **NR** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 85953 a numeric string (see [Expressions in awk](#), on page 2608).

85954 The **getline** operator can form ambiguous constructs when there are  
 85955 unparenthesized binary operators (including concatenate) to the right of the '<'

85956 (up to the end of the expression containing the **getline**). The result of evaluating  
85957 such a construct is unspecified, and conforming applications shall parenthesize  
85958 properly all such usages.

85959 **system**(*expression*)

85960 Execute the command given by *expression* in a manner equivalent to the *system()*  
85961 function defined in the System Interfaces volume of POSIX.1-2024 and return the  
85962 exit status of the command.

85963 All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

85964 Where strings are used as the name of a file or pipeline, the application shall ensure that the  
85965 strings are textually identical. The terminology “same string value” implies that “equivalent  
85966 strings”, even those that differ only by <space> characters, represent different files.

## 85967 User-Defined Functions

85968 The *awk* language also provides user-defined functions. Such functions can be defined as:

```
85969 function name([parameter, ...]) { statements }
```

85970 A function can be referred to anywhere in an *awk* program; in particular, its use can precede its  
85971 definition. The scope of a function is global.

85972 The number of parameters in the function definition need not match the number of parameters  
85973 in the function call. Excess formal parameters can be used as local variables. If fewer arguments  
85974 are supplied in a function call than are in the function definition, the extra parameters that are  
85975 used in the function body as scalars shall evaluate to the uninitialized value until they are  
85976 otherwise initialized, and the extra parameters that are used in the function body as arrays shall  
85977 be treated as uninitialized arrays where each element evaluates to the uninitialized value until  
85978 otherwise initialized.

85979 When invoking a function, no white space can be placed between the function name and the  
85980 opening parenthesis. Function calls can be nested and recursive calls can be made upon  
85981 functions. Upon return from any nested or recursive function call, the values of all of the calling  
85982 function’s parameters shall be unchanged, except for array parameters passed by reference. The  
85983 **return** statement can be used to return a value. If a **return** statement appears outside of a  
85984 function definition, the behavior is undefined.

85985 In the function definition, <newline> characters shall be optional before the opening brace and  
85986 after the closing brace. Function definitions can appear anywhere in the program where a  
85987 *pattern-action* pair is allowed.

## 85988 Grammar

85989 The grammar in this section and the lexical conventions in the following section shall together  
85990 describe the syntax for *awk* programs. The general conventions for this style of grammar are  
85991 described in [Section 1.3](#) (on page 2461). A valid program can be represented as the non-terminal  
85992 symbol *program* in the grammar. This formal syntax shall take precedence over the preceding  
85993 text syntax description.

```
85994 %token NAME NUMBER STRING ERE
85995 %token FUNC_NAME /* Name followed by '(' without white space. */

85996 /* Keywords */
85997 %token Begin End
85998 /* 'BEGIN' 'END' */
```



```

85999 %token      Break   Continue  Delete   Do      Else
86000 /*          'break' 'continue' 'delete' 'do' 'else' */

86001 %token      Exit    For      Function  If      In      Next
86002 /*          'exit' 'for' 'function' 'if' 'in' 'next' */

86003 %token      Nextfile Print   Printf   Return  While
86004 /*          'nextfile' 'print' 'printf' 'return' 'while' */

86005 /* Reserved function names */
86006 %token BUILTIN_FUNC_NAME
86007 /* One token for the following:
86008 * atan2 cos sin exp log sqrt int rand srand
86009 * gsub index length match split sprintf sub
86010 * substr tolower toupper close fflush system
86011 */
86012 %token GETLINE
86013 /* Syntactically different from other built-ins. */

86014 /* Two-character tokens. */
86015 %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
86016 /*      '+='      '-='      '*='      '/='      '%='      '^=' */

86017 %token OR    AND    NO_MATCH  EQ    LE    GE    NE    INCR  DECR  APPEND
86018 /*      '||' ' &&' '!~' '==' '<=' '>=' '!=' '++' '--' '>>' */

86019 /* One-character tokens. */
86020 %token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
86021 %token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='

86022 %start program
86023 %%

86024 program      : item_list
86025               | item_list item
86026               ;

86027 item_list    : /* empty */
86028               | item_list item terminator
86029               ;

86030 item         : action
86031               | pattern action
86032               | normal_pattern
86033               | Function NAME      '(' param_list_opt ')'
86034                 newline_opt action
86035               | Function FUNC_NAME '(' param_list_opt ')'
86036                 newline_opt action
86037               ;

86038 param_list_opt : /* empty */
86039                | param_list
86040                ;

86041 param_list     : NAME
86042                | param_list ',' NAME
86043                ;

```

```

86044 pattern          : normal_pattern
86045                   | special_pattern
86046                   ;

86047 normal_pattern    : expr
86048                   | expr ',' newline_opt expr
86049                   ;

86050 special_pattern    : Begin
86051                   | End
86052                   ;

86053 action             : '{' newline_opt                '}'
86054                   | '{' newline_opt terminated_statement_list '}'
86055                   | '{' newline_opt unterminated_statement_list '}'
86056                   ;

86057 terminator         : terminator NEWLINE
86058                   | ';'
86059                   | NEWLINE
86060                   ;

86061 terminated_statement_list : terminated_statement
86062                   | terminated_statement_list terminated_statement
86063                   ;

86064 unterminated_statement_list : unterminated_statement
86065                   | terminated_statement_list unterminated_statement
86066                   ;

86067 terminated_statement : action newline_opt
86068                   | If '(' expr ')' newline_opt terminated_statement
86069                   | If '(' expr ')' newline_opt terminated_statement
86070                   | Else newline_opt terminated_statement
86071                   | While '(' expr ')' newline_opt terminated_statement
86072                   | For '(' simple_statement_opt ';'
86073                   |   expr_opt ';' simple_statement_opt ')' newline_opt
86074                   |   terminated_statement
86075                   | For '(' NAME In NAME ')' newline_opt
86076                   |   terminated_statement
86077                   | ';' newline_opt
86078                   | terminatable_statement NEWLINE newline_opt
86079                   | terminatable_statement ';'      newline_opt
86080                   ;

86081 unterminated_statement : terminatable_statement
86082                   | If '(' expr ')' newline_opt unterminated_statement
86083                   | If '(' expr ')' newline_opt terminated_statement
86084                   |   Else newline_opt unterminated_statement
86085                   | While '(' expr ')' newline_opt unterminated_statement
86086                   | For '(' simple_statement_opt ';'
86087                   |   expr_opt ';' simple_statement_opt ')' newline_opt
86088                   |   unterminated_statement
86089                   | For '(' NAME In NAME ')' newline_opt
86090                   |   unterminated_statement
86091                   ;

```

```

86092     terminatable_statement : simple_statement
86093         | Break
86094         | Continue
86095         | Next
86096         | Nextfile
86097         | Exit expr_opt
86098         | Return expr_opt
86099         | Do newline_opt terminated_statement While '(' expr ')'
86100         ;

86101     simple_statement_opt : /* empty */
86102         | simple_statement
86103         ;

86104     simple_statement : Delete NAME '[' expr_list ']'
86105         | Delete NAME
86106         | expr
86107         | print_statement
86108         ;

86109     print_statement : simple_print_statement
86110         | simple_print_statement output_redirection
86111         ;

86112     simple_print_statement : Print print_expr_list_opt
86113         | Print '(' multiple_expr_list ')'
86114         | Printf print_expr_list
86115         | Printf '(' multiple_expr_list ')'
86116         ;

86117     output_redirection : '>' expr
86118         | APPEND expr
86119         | '|' expr
86120         ;

86121     expr_list_opt : /* empty */
86122         | expr_list
86123         ;

86124     expr_list : expr
86125         | multiple_expr_list
86126         ;

86127     multiple_expr_list : expr ',' newline_opt expr
86128         | multiple_expr_list ',' newline_opt expr
86129         ;

86130     expr_opt : /* empty */
86131         | expr
86132         ;

86133     expr : unary_expr
86134         | non_unary_expr
86135         ;

86136     unary_expr : '+' expr
86137         | '-' expr

```

```

86138 unary_expr '^' expr
86139 unary_expr '*' expr
86140 unary_expr '/' expr
86141 unary_expr '%' expr
86142 unary_expr '+' expr
86143 unary_expr '-' expr
86144 unary_expr non_unary_expr
86145 unary_expr '<' expr
86146 unary_expr LE expr
86147 unary_expr NE expr
86148 unary_expr EQ expr
86149 unary_expr '>' expr
86150 unary_expr GE expr
86151 unary_expr '~' expr
86152 unary_expr NO_MATCH expr
86153 unary_expr In NAME
86154 unary_expr AND newline_opt expr
86155 unary_expr OR newline_opt expr
86156 unary_expr '?' expr ':' expr
86157 unary_input_function
86158 ;

86159 non_unary_expr : '(' expr ')'
86160                '!' expr
86161                non_unary_expr '^' expr
86162                non_unary_expr '*' expr
86163                non_unary_expr '/' expr
86164                non_unary_expr '%' expr
86165                non_unary_expr '+' expr
86166                non_unary_expr '-' expr
86167                non_unary_expr non_unary_expr
86168                non_unary_expr '<' expr
86169                non_unary_expr LE expr
86170                non_unary_expr NE expr
86171                non_unary_expr EQ expr
86172                non_unary_expr '>' expr
86173                non_unary_expr GE expr
86174                non_unary_expr '~' expr
86175                non_unary_expr NO_MATCH expr
86176                non_unary_expr In NAME
86177                '(' multiple_expr_list ')' In NAME
86178                non_unary_expr AND newline_opt expr
86179                non_unary_expr OR newline_opt expr
86180                non_unary_expr '?' expr ':' expr
86181                NUMBER
86182                STRING
86183                lvalue
86184                ERE
86185                lvalue INCR
86186                lvalue DECR
86187                INCR lvalue
86188                DECR lvalue

```

```

86189         | lvalue POW_ASSIGN expr
86190         | lvalue MOD_ASSIGN expr
86191         | lvalue MUL_ASSIGN expr
86192         | lvalue DIV_ASSIGN expr
86193         | lvalue ADD_ASSIGN expr
86194         | lvalue SUB_ASSIGN expr
86195         | lvalue '=' expr
86196         | FUNC_NAME '(' expr_list_opt ')'
86197         | /* no white space allowed before '(' */
86198         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
86199         | BUILTIN_FUNC_NAME
86200         | non_unary_input_function
86201         ;

86202     print_expr_list_opt : /* empty */
86203         | print_expr_list
86204         ;

86205     print_expr_list    : print_expr
86206         | print_expr_list ',' newline_opt print_expr
86207         ;

86208     print_expr         : unary_print_expr
86209         | non_unary_print_expr
86210         ;

86211     unary_print_expr  : '+' print_expr
86212         | '-' print_expr
86213         | unary_print_expr '^'      print_expr
86214         | unary_print_expr '*'      print_expr
86215         | unary_print_expr '/'      print_expr
86216         | unary_print_expr '%'      print_expr
86217         | unary_print_expr '+'      print_expr
86218         | unary_print_expr '-'      print_expr
86219         | unary_print_expr          non_unary_print_expr
86220         | unary_print_expr '~'      print_expr
86221         | unary_print_expr NO_MATCH print_expr
86222         | unary_print_expr In NAME
86223         | unary_print_expr AND newline_opt print_expr
86224         | unary_print_expr OR  newline_opt print_expr
86225         | unary_print_expr '?' print_expr ':' print_expr
86226         ;

86227     non_unary_print_expr : '(' expr ')'
86228         | '!' print_expr
86229         | non_unary_print_expr '^'      print_expr
86230         | non_unary_print_expr '*'      print_expr
86231         | non_unary_print_expr '/'      print_expr
86232         | non_unary_print_expr '%'      print_expr
86233         | non_unary_print_expr '+'      print_expr
86234         | non_unary_print_expr '-'      print_expr
86235         | non_unary_print_expr          non_unary_print_expr
86236         | non_unary_print_expr '~'      print_expr
86237         | non_unary_print_expr NO_MATCH print_expr

```

```

86238         | non_unary_print_expr In NAME
86239         | '(' multiple_expr_list ')' In NAME
86240         | non_unary_print_expr AND newline_opt print_expr
86241         | non_unary_print_expr OR newline_opt print_expr
86242         | non_unary_print_expr '?' print_expr ':' print_expr
86243         | NUMBER
86244         | STRING
86245         | lvalue
86246         | ERE
86247         | lvalue INCR
86248         | lvalue DECR
86249         | INCR lvalue
86250         | DECR lvalue
86251         | lvalue POW_ASSIGN print_expr
86252         | lvalue MOD_ASSIGN print_expr
86253         | lvalue MUL_ASSIGN print_expr
86254         | lvalue DIV_ASSIGN print_expr
86255         | lvalue ADD_ASSIGN print_expr
86256         | lvalue SUB_ASSIGN print_expr
86257         | lvalue '=' print_expr
86258         | FUNC_NAME '(' expr_list_opt ')'
86259         | /* no white space allowed before '(' */
86260         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
86261         | BUILTIN_FUNC_NAME
86262         ;

86263     lvalue      : NAME
86264                 | NAME '[' expr_list ']'
86265                 | '$' expr
86266                 ;

86267     non_unary_input_function : simple_get
86268                             | simple_get '<' expr
86269                             | non_unary_expr '|' simple_get
86270                             ;

86271     unary_input_function : unary_expr '|' simple_get
86272                         ;

86273     simple_get      : GETLINE
86274                   | GETLINE lvalue
86275                   ;

86276     newline_opt    : /* empty */
86277                   | newline_opt NEWLINE
86278                   ;

```

86279 This grammar has several ambiguities that shall be resolved as follows:

- 86280 • Operator precedence and associativity shall be as described in [Table 3-1](#) (on page 2608).
- 86281 • In case of ambiguity, an **else** shall be associated with the most immediately preceding **if**
- 86282 that would satisfy the grammar.

86283 • In some contexts, a <slash> (' / ') that is used to surround an ERE could also be the  
 86284 division operator. This shall be resolved in such a way that wherever the division operator  
 86285 could appear, a <slash> is assumed to be the division operator. (There is no unary division  
 86286 operator.)

86287 Each expression in an *awk* program shall conform to the precedence and associativity rules, even  
 86288 when this is not needed to resolve an ambiguity. For example, because '\$' has higher  
 86289 precedence than '++', the string "\$x++--" is not a valid *awk* expression, even though it is  
 86290 unambiguously parsed by the grammar as "\$ (x++) --".

86291 One convention that might not be obvious from the formal grammar is where <newline>  
 86292 characters are acceptable. There are several obvious placements such as terminating a statement,  
 86293 and a <backslash> can be used to escape <newline> characters between any lexical tokens. In  
 86294 addition, <newline> characters without <backslash> characters can follow a comma, an open  
 86295 brace, logical AND operator ("&&"), logical OR operator ("||"), the **do** keyword, the **else**  
 86296 keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
86297 { print $1,  
86298         $2 }
```

### 86299 Lexical Conventions

86300 The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as  
 86301 follows:

- 86302 1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a  
 86303 given point.
- 86304 2. A comment shall consist of any characters beginning with the <number-sign> character  
 86305 and terminated by, but excluding the next occurrence of, a <newline>. Comments shall  
 86306 have no effect, except to delimit lexical tokens.
- 86307 3. The <newline> shall be recognized as the token **NEWLINE**.
- 86308 4. A <backslash> character immediately followed by a <newline> shall have no effect.
- 86309 5. The token **STRING** shall represent a string constant. A string constant shall begin with  
 86310 the character '"'. Within a string constant, a <backslash> character shall be considered  
 86311 to begin an escape sequence as specified in the table in XBD [Chapter 5](#) (on page 113)  
 86312 ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequences  
 86313 in [Table 3-2](#) (on page 2616) shall be recognized. A <newline> shall not occur within a  
 86314 string constant. A string constant shall be terminated by the first unescaped occurrence of  
 86315 the character '"' after the one that begins the string constant. The value of the string  
 86316 shall be the sequence of all unescaped characters and values of escape sequences  
 86317 between, but not including, the two delimiting '"' characters.
- 86318 6. The token **ERE** represents an extended regular expression constant. An ERE constant  
 86319 shall begin with the <slash> character. Within an ERE constant, a <backslash> character  
 86320 shall be considered to begin an escape sequence as specified in the table in XBD [Chapter 5](#)  
 86321 (on page 113). In addition, the escape sequences in [Table 3-2](#) (on page 2616) shall be  
 86322 recognized. The application shall ensure that a <newline> does not occur within an ERE  
 86323 constant. An ERE constant shall be terminated by the first unescaped occurrence of the  
 86324 <slash> character after the one that begins the ERE constant. The extended regular  
 86325 expression represented by the ERE constant shall be the sequence of all unescaped  
 86326 characters and values of escape sequences between, but not including, the two delimiting  
 86327 <slash> characters.

- 86328 7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE**  
86329 tokens.
- 86330 8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall  
86331 either be equivalent to the **decimal-floating-constant** token as specified by the ISO C  
86332 standard, or it shall be a sequence of decimal digits and shall be evaluated as an integer  
86333 constant in decimal. In addition, implementations may accept numeric constants with the  
86334 form and numeric value equivalent to the **hexadecimal-constant** and **hexadecimal-**  
86335 **floating-constant** tokens as specified by the ISO C standard. Note that these forms do not  
86336 use the radix character from the current locale; they always use a <period>.
- 86337 If the value is too large or too small to be representable (see [Section 1.1.2](#), on page 2457),  
86338 the behavior is undefined.
- 86339 9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see  
86340 XBD [Section 6.1](#), on page 117), beginning with an <underscore> or alphabetic character,  
86341 shall be considered a word.
- 86342 10. The following words are keywords that shall be recognized as individual tokens; the  
86343 name of the token is the same as the keyword:

86344	<b>BEGIN</b>	<b>delete</b>	<b>END</b>	<b>function</b>	<b>in</b>	<b>print</b>	<b>while</b>
86345	<b>break</b>	<b>do</b>	<b>exit</b>	<b>getline</b>	<b>next</b>	<b>printf</b>	
86346	<b>continue</b>	<b>else</b>	<b>for</b>	<b>if</b>	<b>nextfile</b>	<b>return</b>	

- 86347 11. The following words are names of built-in functions and shall be recognized as the token  
86348 **BUILTIN\_FUNC\_NAME**:

86349	<b>atan2</b>	<b>fflush</b>	<b>int</b>	<b>match</b>	<b>split</b>	<b>srand</b>	<b>system</b>
86350	<b>close</b>	<b>gsub</b>	<b>length</b>	<b>rand</b>	<b>sprintf</b>	<b>sub</b>	<b>tolower</b>
86351	<b>cos</b>	<b>index</b>	<b>log</b>	<b>sin</b>	<b>sqrt</b>	<b>substr</b>	<b>toupper</b>
86352	<b>exp</b>						

86353 The above-listed keywords and names of built-in functions are considered reserved  
86354 words.

- 86355 12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in  
86356 function and is not followed immediately (without any delimiters) by the ' (' character.
- 86357 13. The token **FUNC\_NAME** shall consist of a word that is not a keyword or a name of a  
86358 built-in function, followed immediately (without any delimiters) by the ' (' character.  
86359 The ' (' character shall not be included as part of the token.
- 86360 14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
86361 <b>ADD_ASSIGN</b>	+=	<b>NO_MATCH</b>	!~
86362 <b>SUB_ASSIGN</b>	-=	<b>EQ</b>	==
86363 <b>MUL_ASSIGN</b>	*=	<b>LE</b>	<=
86364 <b>DIV_ASSIGN</b>	/=	<b>GE</b>	>=
86365 <b>MOD_ASSIGN</b>	%=	<b>NE</b>	!=
86366 <b>POW_ASSIGN</b>	^=	<b>INCR</b>	++
86367 <b>OR</b>		<b>DECR</b>	--
86368 <b>AND</b>	&&	<b>APPEND</b>	>>



86370 15. The following single characters shall be recognized as tokens whose names are the  
86371 character:

86372 <newline> { } ( ) [ ] , ; + - \* % ^ ! > < | ? : ~ \$ =

86373 There is a lexical ambiguity between the token **ERE** and the tokens **'/'** and **DIV\_ASSIGN**.  
86374 When an input sequence begins with a <slash> character in any syntactic context where the  
86375 token **'/'** or **DIV\_ASSIGN** could appear as the next token in a valid program, the longer of  
86376 those two tokens that can be recognized shall be recognized. In any other syntactic context  
86377 where the token **ERE** could appear as the next token in a valid program, the token **ERE** shall be  
86378 recognized.

#### 86379 EXIT STATUS

86380 The following exit values shall be returned:

86381 0 All input files were processed successfully.

86382 >0 An error occurred.

86383 The exit status can be altered within the program by using an **exit** expression.

#### 86384 CONSEQUENCES OF ERRORS

86385 If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a  
86386 diagnostic message to standard error and terminate without any further action.

86387 If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk*  
86388 program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

#### 86389 APPLICATION USAGE

86390 Since <backslash> has a special meaning both in the *assignment* option-argument to the **-v**  
86391 option and in the *assignment* operand, applications that need to pass strings to *awk* without  
86392 special interpretation of <backslash> should not use these methods but should instead make use  
86393 of the **ARGV** or **ENVIRON** array.

86394 The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in  
86395 the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with  
86396 bytes.

86397 Because the concatenation operation is represented by adjacent expressions rather than an  
86398 explicit operator, it is often necessary to use parentheses to enforce the proper evaluation  
86399 precedence.

86400 When using *awk* to process pathnames, it is recommended that **LC\_ALL**, or at least **LC\_CTYPE**  
86401 and **LC\_COLLATE**, are set to **POSIX** or **C** in the environment, since pathnames can contain byte  
86402 sequences that do not form valid characters in some locales, in which case the utility's behavior  
86403 would be undefined. In the **POSIX** locale each byte is a valid single-byte character, and therefore  
86404 this problem is avoided.

86405 Since the **"=="** operator checks if strings are identical, not whether they collate equally,  
86406 applications needing to check whether strings collate equally can use:

86407 a <= b && a >= b

86408 To specify a *file* operand naming a file with a name containing an <equals-sign>, users can use  
86409 **"/** as the first two characters of a relative file pathname that starts with an <underscore> or  
86410 an alphabetic character to keep the *file* operand from being interpreted as an *assignment* operand.  
86411 Similarly, **"/-** can be used to access a file named **'-'** in the current directory rather than use  
86412 standard input.

86413 **EXAMPLES**

86414 The *awk* program specified in the command line is most easily specified within single-quotes (for  
 86415 example, '*program*') for applications using *sh*, because *awk* programs commonly contain  
 86416 characters that are special to the shell, including double-quotes. In the cases where an *awk*  
 86417 program contains single-quote characters, it is usually easiest to specify most of the program as  
 86418 strings within single-quotes concatenated by the shell with quoted single-quote characters. For  
 86419 example:

```
86420 awk '/'\''/ { print "quote:", $0 }
```

86421 prints all lines from the standard input containing a single-quote character, prefixed with *quote*..

86422 The following are examples of simple *awk* programs:

- 86423 1. Write to the standard output all input lines for which field 3 is greater than 5:

```
86424     $3 > 5
```

- 86425 2. Write every tenth line:

```
86426     (NR % 10) == 0
```

- 86427 3. Write any line with a substring matching the regular expression:

```
86428     /(G|D)(2[0-9][[:alpha:]]*)/
```

- 86429 4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits  
 86430 and characters. This example uses character classes **digit** and **alpha** to match language-  
 86431 independent digit and alphabetic characters respectively:

```
86432     /(G|D)([[:digit:]][[:alpha:]]*)/
```

- 86433 5. Write any line in which the second field matches the regular expression and the fourth  
 86434 field does not:

```
86435     $2 ~ /xyz/ && $4 !~ /xyz/
```

- 86436 6. Write any line in which the second field contains a <backslash>:

```
86437     $2 ~ /\\"/>

```

- 86438 7. Write any line in which the second field contains a <backslash>. Note that  
 86439 <backslash>-escapes are interpreted twice; once in lexical processing of the string and  
 86440 once in processing the regular expression:

```
86441     $2 ~ "\\\"/>

```

- 86442 8. Write the second to the last and the last field in each line. Separate the fields by a <colon>:

```
86443     {OFS=":";print $(NF-1), $NF}
```

- 86444 9. Write the line number and number of fields in each line. The three strings representing  
 86445 the line number, the <colon>, and the number of fields are concatenated and that string is  
 86446 written to standard output:

```
86447     {print NR ":" NF}
```

- 86448 10. Write lines longer than 72 characters:

```
86449     length($0) > 72
```

- 86450 11. Write the first two fields in opposite order separated by **OFS**:

```
86451     { print $2, $1 }
```

- 86452 12. Same, with input fields separated by a <comma> or <space> and <tab> characters, or  
86453 both:
- ```
86454 BEGIN { FS = ", [ \t]* | [ \t]+" }
86455       { print $2, $1 }
```
- 86456 13. Add up the first column, print sum, and average:
- ```
86457       { s += $1 }
86458 END   { print "sum is ", s, " average is", s/NR }
```
- 86459 14. Write fields in reverse order, one per line (many lines out for each line in):
- ```
86460     { for (i = NF; i > 0; --i) print $i }
```
- 86461 15. Write all lines between occurrences of the strings **start** and **stop**:
- ```
86462     /start/, /stop/
```
- 86463 16. Write all lines whose first field is different from the previous one:
- ```
86464     $1 != prev { print; prev = $1 }
```
- 86465 17. Simulate *echo*:
- ```
86466 BEGIN {
86467     for (i = 1; i < ARGV; ++i)
86468         printf("%s%s", ARGV[i], i==ARGC-1?"\n":" ")
86469 }
```
- 86470 18. Write the path prefixes contained in the *PATH* environment variable, one per line:
- ```
86471 BEGIN {
86472     n = split (ENVIRON["PATH"], path, ":")
86473     for (i = 1; i <= n; ++i)
86474         print path[i]
86475 }
```
- 86476 19. If there is a file named **input** containing page headers of the form:
- ```
86477 Page #
```
- 86478 and a file named **program** that contains:
- ```
86479 /Page/   { $2 = n++; }
86480         { print }
```
- 86481 then the command line:
- ```
86482 awk -f program n=5 input
```
- 86483 prints the file **input**, filling in page numbers starting at 5.

#### 86484 RATIONALE

86485 This description is based on the new *awk*, “*nawk*”, (see the referenced *The AWK Programming*  
86486 *Language*), which introduced a number of new features to the historical *awk*:

- 86487 1. New keywords: **delete**, **do**, **function**, **return**
- 86488 2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**

- 86489 3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
- 86490 4. New expression operators: **?**, **:**, **..**, **^**
- 86491 5. The **FS** variable and the third argument to **split**, now treated as extended regular
- 86492 expressions.
- 86493 6. The operator precedence, changed to more closely match the C language. Two examples
- 86494 of code that operate differently are:
- ```
86495 while ( n /= 10 > 1) ...
86496 if (!"wk" ~ /bwk/) ...
```

86497 Several features have been added based on newer implementations of *awk*:

- 86498 • Multiple instances of **-f profile** are permitted.
- 86499 • The new option **-v assignment**.
- 86500 • The new predefined variable **ENVIRON**.
- 86501 • New built-in functions **toupper** and **tolower**.
- 86502 • More formatting capabilities are added to **printf** to match the ISO C standard.

86503 Earlier versions of this standard required implementations to support multiple adjacent

86504 `<semicolon>`s, lines with one or more `<semicolon>` before a rule (*pattern-action* pairs), and lines

86505 with only `<semicolon>`(s). These are not required by this standard and are considered poor

86506 programming practice, but can be accepted by an implementation of *awk* as an extension.

86507 The overall *awk* syntax has always been based on the C language, with a few features from the

86508 shell command language and other sources. Because of this, it is not completely compatible with

86509 any other language, which has caused confusion for some users. It is not the intent of the

86510 standard developers to address such issues. A few relatively minor changes toward making the

86511 language more compatible with the ISO C standard were made; most of these changes are based

86512 on similar changes in recent implementations, as described above. There remain several C-

86513 language conventions that are not in *awk*. One of the notable ones is the `<comma>` operator,

86514 which is commonly used to specify multiple expressions in the C language **for** statement. Also,

86515 there are various places where *awk* is more restrictive than the C language regarding the type of

86516 expression that can be used in a given context. These limitations are due to the different features

86517 that the *awk* language does provide.

86518 Regular expressions in *awk* have been extended somewhat from historical implementations to

86519 make them a pure superset of extended regular expressions, as defined by POSIX.1-2024 (see

86520 XBD Section 9.4, on page 187). The main extensions are internationalization features and

86521 interval expressions. Historical implementations of *awk* have long supported

86522 `<backslash>`-escape sequences as an extension to extended regular expressions, and this

86523 extension has been retained despite inconsistency with other utilities. The number of escape

86524 sequences recognized in both extended regular expressions and strings has varied (generally

86525 increasing with time) among implementations. The set specified by POSIX.1-2024 includes most

86526 sequences known to be supported by popular implementations and by the ISO C standard. One

86527 sequence that is not supported is hexadecimal value escapes beginning with `'\x'`. This would

86528 allow values expressed in more than 9 bits to be used within *awk* as in the ISO C standard.

86529 However, because this syntax has a non-deterministic length, it does not permit the subsequent

86530 character to be a hexadecimal digit. This limitation can be dealt with in the C language by the

86531 use of lexical string concatenation. In the *awk* language, concatenation could also be a solution

86532 for strings, but not for extended regular expressions (either lexical ERE tokens or strings used

86533 dynamically as regular expressions). Because of this limitation, the feature has not been added to

86534 POSIX.1-2024.

86535 When a string variable is used in a context where an extended regular expression normally  
86536 appears (where the lexical token ERE is used in the grammar) the string does not contain the  
86537 literal <slash> characters.

86538 Some versions of *awk* allow the form:

```
86539 func name(args, ... ) { statements }
```

86540 This has been deprecated by the authors of the language, who asked that it not be specified.

86541 Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN**  
86542 action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This  
86543 behavior has not been documented, and it was not believed that it was necessary to standardize  
86544 it.

86545 The specification of conversions between string and numeric values is much more detailed than  
86546 in the documentation of historical implementations or in the referenced *The AWK Programming*  
86547 *Language*. Although most of the behavior is designed to be intuitive, the details are necessary to  
86548 ensure compatible behavior from different implementations. This is especially important in  
86549 relational expressions since the types of the operands determine whether a string or numeric  
86550 comparison is performed. From the perspective of an application developer, it is usually  
86551 sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating  
86552 a null string) when the type of an expression does not obviously match what is needed. The  
86553 intent has been to specify historical practice in almost all cases. The one exception is that, in  
86554 historical implementations, variables and constants maintain both string and numeric values  
86555 after their original value is converted by any use. This means that referencing a variable or  
86556 constant can have unexpected side-effects. For example, with historical implementations the  
86557 following program:

```
86558 {  
86559     a = "+2"  
86560     b = 2  
86561     if (NR % 2)  
86562         c = a + b  
86563     if (a == b)  
86564         print "numeric comparison"  
86565     else  
86566         print "string comparison"  
86567 }
```

86568 would perform a numeric comparison (and output numeric comparison) for each odd-  
86569 numbered line, but perform a string comparison (and output string comparison) for each even-  
86570 numbered line. POSIX.1-2024 ensures that comparisons will be numeric if necessary. With  
86571 historical implementations, the following program:

```
86572 BEGIN {  
86573     OFMT = "%e"  
86574     print 3.14  
86575     OFMT = "%f"  
86576     print 3.14  
86577 }
```

86578 would output "3.140000e+00" twice, because in the second **print** statement the constant  
86579 "3.14" would have a string value from the previous conversion. POSIX.1-2024 requires that the  
86580 output of the second **print** statement be "3.140000". The behavior of historical

86581 implementations was seen as too unintuitive and unpredictable.

86582 It was pointed out that with the rules contained in early drafts, the following script would print  
86583 nothing:

```
86584 BEGIN {
86585     y[1.5] = 1
86586     OFMT = "%e"
86587     print y[1.5]
86588 }
```

86589 Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to  
86590 affecting output conversions of numbers to strings and **CONVFMT** is used for internal  
86591 conversions, such as comparisons or array indexing. The default value is the same as that for  
86592 **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it  
86593 will receive the historical behavior associated with internal string conversions.

86594 The POSIX *awk* lexical and syntactic conventions are specified more formally than in other  
86595 sources. Again the intent has been to specify historical practice. One convention that may not be  
86596 obvious from the formal grammar as in other verbal descriptions is where <newline> characters  
86597 are acceptable. There are several obvious placements such as terminating a statement, and a  
86598 <backslash> can be used to escape <newline> characters between any lexical tokens. In addition,  
86599 <newline> characters without <backslash> characters can follow a comma, an open brace, a  
86600 logical AND operator ("&&"), a logical OR operator ("||"), the **do** keyword, the **else** keyword,  
86601 and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
86602 { print $1,
86603     $2 }
```

86604 The requirement that *awk* add a trailing <newline> to the program argument text is to simplify  
86605 the grammar, making it match a text file in form. There is no way for an application or test suite  
86606 to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

86607 POSIX.1-2024 requires several changes from historical implementations in order to support  
86608 internationalization. Probably the most subtle of these is the use of the decimal-point character,  
86609 defined by the *LC\_NUMERIC* category of the locale, in representations of floating-point  
86610 numbers. This locale-specific character is used in recognizing numeric input, in converting  
86611 between strings and numeric values, and in formatting output. However, regardless of locale,  
86612 the <period> character (the decimal-point character of the POSIX locale) is the decimal-point  
86613 character recognized in processing *awk* programs (including assignments in command line  
86614 arguments). This is essentially the same convention as the one used in the ISO C standard. The  
86615 difference is that the C language includes the *setlocale()* function, which permits an application  
86616 to modify its locale. Because of this capability, a C application begins executing with its locale set  
86617 to the C locale, and only executes in the environment-specified locale after an explicit call to  
86618 *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as  
86619 inappropriate for POSIX.1-2024. It is possible to execute an *awk* program explicitly in any desired  
86620 locale by setting the environment in the shell.

86621 The undefined behavior resulting from NULs in extended regular expressions allows future  
86622 extensions for the GNU *gawk* program to process binary data.

86623 The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic  
86624 errors) is undefined because it was considered overly limiting on implementations to specify. In  
86625 most cases such errors can be expected to produce a diagnostic and a non-zero exit status.  
86626 However, some implementations may choose to extend the language in ways that make use of  
86627 certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but

86628 otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect  
 86629 in some implementations. Also, different implementations might detect a given error during an  
 86630 initial parsing of the program (before reading any input files) while others might detect it when  
 86631 executing the program after reading some input. Implementors should be aware that diagnosing  
 86632 errors as early as possible and producing useful diagnostics can ease debugging of applications,  
 86633 and thus make an implementation more usable.

86634 The unspecified behavior from using multi-character **RS** values is to allow possible future  
 86635 extensions based on extended regular expressions used for record separators. Historical  
 86636 implementations take the first character of the string and ignore the others.

86637 Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future  
 86638 extension that would split up a string into an array of individual characters.

86639 In the context of the **getline** function, equally good arguments for different precedences of the |  
 86640 and < operators can be made. Historical practice has been that:

86641 `getline < "a" "b"`

86642 is parsed as:

86643 `( getline < "a" ) "b"`

86644 although many would argue that the intent was that the file **ab** should be read. However:

86645 `getline < "x" + 1`

86646 parses as:

86647 `getline < ( "x" + 1 )`

86648 Similar problems occur with the | version of **getline**, particularly in combination with \$. For  
 86649 example:

86650  `$"echo hi" | getline`

86651 (This situation is particularly problematic when used in a **print** statement, where the |**getline**  
 86652 part might be a redirection of the **print**.)

86653 Since in most cases such constructs are not (or at least should not) be used (because they have a  
 86654 natural ambiguity for which there is no conventional parsing), the meaning of these constructs  
 86655 has been made explicitly unspecified. (The effect is that a conforming application that runs into  
 86656 the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual  
 86657 uses of such constructs.

86658 Grammars can be written that would cause an error under these circumstances. Where  
 86659 backwards-compatibility is not a large consideration, implementors may wish to use such  
 86660 grammars.

86661 Some historical implementations have allowed some built-in functions to be called without an  
 86662 argument list, the result being a default argument list chosen in some "reasonable" way. Use of  
 86663 **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely  
 86664 known or widely used; this particular form is documented in various places (for example, most  
 86665 historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as  
 86666 legitimate practice. With this exception, default argument lists have always been undocumented  
 86667 and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-  
 86668 defined functions. They add no useful functionality and preclude possible future extensions that  
 86669 might need to name functions without calling them. Not standardizing them seems the simplest  
 86670 course. The standard developers considered that **length** merited special treatment, however,  
 86671 since it has been documented in the past and sees possibly substantial use in historical

86672 programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the  
86673 obsolescent marking for XSI-conforming implementations and many otherwise conforming  
86674 applications depend on this feature.

86675 In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive  
86676 <backslash> characters should be used in the string to ensure a single <backslash> will precede  
86677 the <ampersand> when the resultant string is passed to the function. (For example, to specify  
86678 one literal <ampersand> in the replacement string, use **gsub(ERE, "\\&")**.)

86679 Historically, the only special character in the *repl* argument of **sub** and **gsub** string functions was  
86680 the <ampersand> ('&') character and preceding it with the <backslash> character was used to  
86681 turn off its special meaning.

86682 The description in the ISO POSIX-2:1993 standard introduced behavior such that the  
86683 <backslash> character was another special character and it was unspecified whether there were  
86684 any other special characters. This description introduced several portability problems, some of  
86685 which are described below, and so it has been replaced with the more historical description.  
86686 Some of the problems include:

- 86687 • Historically, to create the replacement string, a script could use **gsub(ERE, "\\&")**, but  
86688 with the ISO POSIX-2:1993 standard wording, it was necessary to use **gsub(ERE,**  
86689 **"\\\\&")**. The <backslash> characters are doubled here because all string literals are  
86690 subject to lexical analysis, which would reduce each pair of <backslash> characters to a  
86691 single <backslash> before being passed to **gsub**.
- 86692 • Since it was unspecified what the special characters were, for portable scripts to guarantee  
86693 that characters are printed literally, each character had to be preceded with a <backslash>.  
86694 (For example, a portable script had to use **gsub(ERE, "\\h\\i")** to produce a replacement  
86695 string of "hi".)

86696 The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe  
86697 historical practice because of the way numeric strings are compared as numbers. The current  
86698 rules cause the following code:

```
86699 if (0 == "000")
86700     print "strange, but true"
86701 else
86702     print "not true"
```

86703 to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this  
86704 is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

86705 To fix this problem, the definition of *numeric string* was enhanced to include only those values  
86706 obtained from specific circumstances (mostly external sources) where it is not possible to  
86707 determine unambiguously whether the value is intended to be a string or a numeric.

86708 Variables that are assigned to a numeric string shall also be treated as a numeric string. (For  
86709 example, the notion of a numeric string can be propagated across assignments.) In comparisons,  
86710 all variables having the uninitialized value are to be treated as a numeric operand evaluating to  
86711 the numeric value zero.

86712 Uninitialized variables include all types of variables including scalars, array elements, and  
86713 fields. The definition of an uninitialized value in [Variables and Special Variables](#) (on page 2612)  
86714 is necessary to describe the value placed on uninitialized variables and on fields that are valid  
86715 (for example, < \$NF) but have no characters in them and to describe how these variables are to  
86716 be used in comparisons. A valid field, such as \$1, that has no characters in it can be obtained  
86717 from an input line of "\t\t" when FS=' \t '. Historically, the comparison (\$1<10) was done



86718 numerically after evaluating `$1` to the value zero.

86719 The phrase "... also shall have the numeric value of the numeric string" was removed from  
86720 several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary  
86721 implementation detail. It is not necessary for POSIX.1-2024 to specify that these objects be  
86722 assigned two different values. It is only necessary to specify that these objects may evaluate to  
86723 two different values depending on context.

86724 Historical implementations of *awk* did not parse hexadecimal integer or floating constants like  
86725 "0xa" and "0xap0". Due to an oversight, the 2001 through 2004 editions of this standard  
86726 required support for hexadecimal floating constants. This was due to the reference to *atof()*.  
86727 This version of the standard allows but does not require implementations to use *atof()* and  
86728 includes a description of how floating-point numbers are recognized as an alternative to match  
86729 historic behavior. The intent of this change is to allow implementations to recognize floating-  
86730 point constants according to either the ISO/IEC 9899:1990 standard or ISO/IEC 9899:1999  
86731 standard, and to allow (but not require) implementations to recognize hexadecimal integer  
86732 constants.

86733 Historical implementations of *awk* did not support floating-point infinities and NaNs in *numeric*  
86734 *strings*; e.g., "-INF" and "NaN". However, implementations that use the *atof()* or *strtod()*  
86735 functions to do the conversion picked up support for these values if they used a  
86736 ISO/IEC 9899:1999 standard version of the function instead of a ISO/IEC 9899:1990 standard  
86737 version. Due to an oversight, the 2001 through 2004 editions of this standard did not allow  
86738 support for infinities and NaNs, but in this revision support is allowed (but not required). This is  
86739 a silent change to the behavior of *awk* programs; for example, in the POSIX locale the expression:

86740 (`"-INF" + 0 < 0`)

86741 formerly had the value 0 because "-INF" converted to 0, but now it may have the value 0 or 1.

86742 Deleting all elements of an array one element at a time, via:

```
86743 for (index in array)
86744     delete array[index]
```

86745 is usually not efficient. This standard requires `delete array` to have the same effects, and this  
86746 was supported in most implementations as a more efficient operation. It is also possible to use  
86747 `split("", array)` to achieve the same effect and efficiency.

## 86748 FUTURE DIRECTIONS

86749 If this utility is directed to create a new directory entry that contains any bytes that have the  
86750 encoded value of a <newline> character, implementations are encouraged to treat this as an  
86751 error. A future version of this standard may require implementations to treat this as an error.

86752 A future version of this standard may require **srand** to accept any numeric value and calculate  
86753 the seed by taking the provided value, converting it to an integer, and calculating the integer  
86754 value modulo  $2^n$  where  $n$  is an implementation-defined value greater than or equal to 32.

86755 A future version of this standard may require the initial seed for the **rand** function (the seed  
86756 value used if **srand** is not called) to be an integer between 0 and  $2^n-1$  inclusive where  $n$  is an  
86757 implementation-defined value greater than or equal to 32. Additionally, the initial seed value  
86758 may be required to be a (pseudo-)random value such that two invocations of *awk* are unlikely to  
86759 emit the same sequence of random values (unless the seed is explicitly set to the same value via  
86760 **srand**).

86761 A future version of this standard may define a new **posix\_srand** function that enables  
86762 application authors to set the seed to a (pseudo-)random value generated by the system.  
86763 Alternatively, the specification of the **srand** function may be altered to provide some means to

- 86764 set the default seed value to a (pseudo-)random value.
- 86765 **SEE ALSO**
- 86766 [Section 1.3](#) (on page 2461), [grep](#), [lex](#), [sed](#)
- 86767 [XBD Chapter 5](#) (on page 113), [Section 6.1](#) (on page 117), [Chapter 8](#) (on page 167), [Chapter 9](#) (on page 179), [Section 12.2](#) (on page 215)
- 86768
- 86769 [XSH atof\(\)](#), [exec](#), [isspace\(\)](#), [popen\(\)](#), [setlocale\(\)](#), [strtod\(\)](#)
- 86770 **CHANGE HISTORY**
- 86771 First released in Issue 2.
- 86772 **Issue 5**
- 86773 The FUTURE DIRECTIONS section is added.
- 86774 **Issue 6**
- 86775 The *awk* utility is aligned with the IEEE P1003.2b draft standard.
- 86776 The normative text is reworded to avoid use of the term “must” for application requirements.
- 86777 IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence “An occurrence of two consecutive `<backslash>` characters shall be interpreted as just a single literal `<backslash>` character.” into the description of the **sub** string function.
- 86778
- 86779
- 86780 **Issue 7**
- 86781 PASC Interpretation 1003.2-1992 #107 (SD5-XCU-ERN-73) is applied, updating the description of the **OFS** variable.
- 86782
- 86783 Austin Group Interpretation 1003.1-2001 #189 is applied.
- 86784 Austin Group Interpretation 1003.1-2001 #201 is applied, permitting implementations to support infinities and NaNs.
- 86785
- 86786 SD5-XCU-ERN-79 is applied, restoring the horizontal lines to [Table 3-1](#) (on page 2608), and SD5-XCU-ERN-80 is applied, changing the order of some table entries.
- 86787
- 86788 SD5-XCU-ERN-87 is applied, updating the descriptive text of the Grammar.
- 86789 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 86790 The EXTENDED DESCRIPTION is changed to make the support of hexadecimal integer and floating constants optional.
- 86791
- 86792 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0057 [224], XCU/TC1-2008/0058 [454], XCU/TC1-2008/0059 [224], XCU/TC1-2008/0060 [224], XCU/TC1-2008/0061 [254], XCU/TC1-2008/0062 [254], XCU/TC1-2008/0063 [224], and XCU/TC1-2008/0064 [454] are applied.
- 86793
- 86794
- 86795
- 86796 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0058 [584], XCU/TC2-2008/0059 [963], XCU/TC2-2008/0060 [226], XCU/TC2-2008/0061 [663], XCU/TC2-2008/0062 [963], XCU/TC2-2008/0063 [226], and XCU/TC2-2008/0064 [963] are applied.
- 86797
- 86798
- 86799 **Issue 8**
- 86800 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a `<newline>` character.
- 86801
- 86802 Austin Group Defects 544 and 1136 are applied, requiring implementations to accept the **delete** statement with an unsubscripted array name.
- 86803
- 86804 Austin Group Defect 607 is applied, adding the **nextfile** statement.

- 86805 Austin Group Defect 634 is applied, adding the **fflush** function.
- 86806 Austin Group Defects 974 and 1451 are applied, clarifying the **ARGC**, **ARGV** and **FILENAME**  
86807 variables, and adding to APPLICATION USAGE.
- 86808 Austin Group Defect 983 is applied, changing the descriptions of the **rand** and **srand** functions  
86809 and the FUTURE DIRECTIONS section.
- 86810 Austin Group Defect 1070 is applied, requiring the "!=" and "==" operators to perform string  
86811 comparisons by checking if the strings are identical (and not by checking if they collate equally).
- 86812 Austin Group Defect 1105 is applied, clarifying the requirements for <backslash> escaping.
- 86813 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 86814 Austin Group Defect 1198 is applied, requiring comparisons to be performed numerically when  
86815 both operands have string values that are numeric strings.
- 86816 Austin Group Defect 1277 is applied, clarifying that using a <slash> character within an ERE  
86817 requires escaping only if it is within the lexical token **ERE**.
- 86818 Austin Group Defect 1320 is applied, clarifying the condition under which ERE matching is  
86819 against input records.
- 86820 Austin Group Defect 1395 is applied, changing the requirements for string to number  
86821 conversion.
- 86822 Austin Group Defect 1468 is applied, clarifying the behavior when **FS** is an ERE that can match  
86823 the null string.
- 86824 Austin Group Defect 1566 is applied, specifying the behavior of the **length** function when  
86825 passed an array argument.

86826 **NAME**

86827 `basename` — return non-directory portion of a pathname

86828 **SYNOPSIS**

86829 `basename string [suffix]`

86830 **DESCRIPTION**

86831 The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.254](#) (on page 68).  
 86832 The string *string* shall be converted to the filename corresponding to the last pathname  
 86833 component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be  
 86834 done by performing actions equivalent to the following steps in order:

- 86835 1. If *string* is a null string, it is unspecified whether the resulting string is '.' or a null  
 86836 string. In either case, skip steps 2 through 6.
- 86837 2. If *string* is `"/"`, it is implementation-defined whether steps 3 to 6 are skipped or  
 86838 processed.
- 86839 3. If *string* consists entirely of `<slash>` characters, *string* shall be set to a single `<slash>`  
 86840 character. In this case, skip steps 4 to 6.
- 86841 4. If there are any trailing `<slash>` characters in *string*, they shall be removed.
- 86842 5. If there are any `<slash>` characters remaining in *string*, the prefix of *string* up to and  
 86843 including the last `<slash>` character in *string* shall be removed.
- 86844 6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is  
 86845 identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed  
 86846 from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an  
 86847 error if *suffix* is not found in *string*.

86848 The resulting string shall be written to standard output.

86849 **OPTIONS**

86850 None.

86851 **OPERANDS**

86852 The following operands shall be supported:

86853 *string* A string.

86854 *suffix* A string.

86855 **STDIN**

86856 Not used.

86857 **INPUT FILES**

86858 None.

86859 **ENVIRONMENT VARIABLES**

86860 The following environment variables shall affect the execution of *basename*:

86861 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 86862 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 86863 variables used to determine the values of locale categories.)

86864 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 86865 internationalization variables.

86866 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 86867 characters (for example, single-byte as opposed to multi-byte characters in  
 86868 arguments).

86869 **LC\_MESSAGES**  
 86870 Determine the locale that should be used to affect the format and contents of  
 86871 diagnostic messages written to standard error.

86872 XSI **NLSPATH** Determine the location of messages objects and message catalogs.

86873 **ASYNCHRONOUS EVENTS**  
 86874 Default.

86875 **STDOUT**  
 86876 The *basename* utility shall write a line to the standard output in the following format:  
 86877 "%s\n", <resulting string>

86878 **STDERR**  
 86879 The standard error shall be used only for diagnostic messages.

86880 **OUTPUT FILES**  
 86881 None.

86882 **EXTENDED DESCRIPTION**  
 86883 None.

86884 **EXIT STATUS**  
 86885 The following exit values shall be returned:  
 86886 0 Successful completion.  
 86887 >0 An error occurred.

86888 **CONSEQUENCES OF ERRORS**  
 86889 Default.

86890 **APPLICATION USAGE**  
 86891 The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
 86892 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters  
 86893 to the beginning of a pathname unless they can ensure that there are more or less than two or are  
 86894 prepared to deal with the implementation-defined consequences.

86895 **EXAMPLES**  
 86896 If the string *string* is a valid pathname:  
 86897 `$(basename -- "string")`  
 86898 produces a filename that could be used to open the file named by *string* in the directory returned  
 86899 by:  
 86900 `$(dirname -- "string")`  
 86901 If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be  
 86902 a valid filename. The *basename* utility is not expected to make any judgements about the validity  
 86903 of *string* as a pathname; it just follows the specified algorithm to produce a result string.

86904 The following shell script compiles `/usr/src/cmd/cat.c` and moves the output to a file named `cat`  
 86905 in the current directory when invoked with the argument `/usr/src/cmd/cat` or with the argument  
 86906 `/usr/src/cmd/cat.c`:  
 86907 `c17 -- "$(dirname -- "$1")/$(basename -- "$1" .c) .c" &&`  
 86908 `mv a.out "$(basename -- "$1" .c) "`

86909 The EXAMPLES section of the *basename()* function (see XSH [basename\(\)](#)) includes a table  
 86910 showing examples of the results of processing several sample pathnames by the *basename()* and

86911 *dirname()* functions and by the *basename* and *dirname* utilities.

#### 86912 RATIONALE

86913 The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid  
86914 pathname:

86915 `$(basename -- "string")`

86916 would be a valid filename for the file in the directory:

86917 `$(dirname -- "string")`

86918 This would not work for the early proposal versions of these utilities due to the way it specified  
86919 handling of trailing `<slash>` characters.

86920 Since the definition of *pathname* specifies implementation-defined behavior for pathnames  
86921 starting with two `<slash>` characters, this volume of POSIX.1-2024 specifies similar  
86922 implementation-defined behavior for the *basename* and *dirname* utilities.

#### 86923 FUTURE DIRECTIONS

86924 If this utility is directed to display a pathname that contains any bytes that have the encoded  
86925 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
86926 format being used, implementations are encouraged to treat this as an error. A future version of  
86927 this standard may require implementations to treat this as an error.

#### 86928 SEE ALSO

86929 [Section 2.5](#) (on page 2478), *dirname*

86930 [XBD Section 3.254](#) (on page 68), [Chapter 8](#) (on page 167)

86931 XSH *basename()*, *dirname()*

#### 86932 CHANGE HISTORY

86933 First released in Issue 2.

#### 86934 Issue 6

86935 IEEE PASC Interpretation 1003.2 #164 is applied.

86936 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 86937 Issue 7

86938 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0065 [192,538], XCU/TC1-2008/0066  
86939 [192,538], and XCU/TC1-2008/0067 [192,430,538] are applied.

86940 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0065 [612] is applied.

#### 86941 Issue 8

86942 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
86943 directed to display a pathname that contains any bytes that have the encoded value of a  
86944 `<newline>` character when `<newline>` is a terminator or separator in the output format being  
86945 used.

86946 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

86947 **NAME**

86948           batch — schedule commands to be executed in a batch queue

86949 **SYNOPSIS**86950           *batch*86951 **DESCRIPTION**86952           The *batch* utility shall read commands from standard input and schedule them for execution in a  
86953           batch queue. It shall be the equivalent of the command:

86954           at -q b -m now

86955           where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to  
86956           the batch queue with no time constraints and shall be run by the system using algorithms, based  
86957           on unspecified factors, that may vary with each invocation of *batch*.86958 XSI        Users shall be permitted to use *batch* if their name appears in the file **at.allow** which is located in  
86959           an implementation-defined directory. If that file does not exist, the file **at.deny**, which is located  
86960           in an implementation-defined directory, shall be checked to determine whether the user shall be  
86961           denied access to *batch*. If neither file exists, only a process with appropriate privileges shall be  
86962           allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The  
86963           **at.allow** and **at.deny** files shall consist of one user name per line.86964 **OPTIONS**

86965           None.

86966 **OPERANDS**

86967           None.

86968 **STDIN**86969           The standard input shall be a text file consisting of commands acceptable to the shell command  
86970           language described in [Chapter 2](#) (on page 2472).86971 **INPUT FILES**86972 XSI        The text files **at.allow** and **at.deny**, which are located in an implementation-defined directory,  
86973           shall contain zero or more user names, one per line, of users who are, respectively, authorized or  
86974           denied access to the *at* and *batch* utilities.86975 **ENVIRONMENT VARIABLES**86976           The following environment variables shall affect the execution of *batch*:86977           **LANG**        Provide a default value for the internationalization variables that are unset or null.  
86978                        (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
86979                        variables used to determine the values of locale categories.)86980           **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
86981                        internationalization variables.86982           **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
86983                        characters (for example, single-byte as opposed to multi-byte characters in  
86984                        arguments and input files).86985           **LC\_MESSAGES**86986                        Determine the locale that should be used to affect the format and contents of  
86987                        diagnostic messages written to standard error and informative messages written to  
86988                        standard output.86989           **LC\_TIME**     Determine the format and contents for date and time strings written by *batch*.

|       |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------|-----|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 86990 | XSI | <b>NLSPATH</b>                | Determine the location of messages objects and message catalogs.                                                                                                                                                                                                                                                                                                                                                 |
| 86991 |     | <b>SHELL</b>                  | Determine the name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; any of the preceding accompanied by a warning diagnostic about which was chosen. |
| 86992 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 86993 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 86994 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 86995 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 86996 |     | <b>TZ</b>                     | Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <b>-t</b> <i>time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it overrides <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.                 |
| 86997 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 86998 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 86999 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87000 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87001 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87002 |     |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                         |
| 87003 |     | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87004 |     |                               | When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.                                                                                                                                                                                                                                               |
| 87005 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87006 |     | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87007 |     |                               | The following shall be written to standard error when a job has been successfully submitted:                                                                                                                                                                                                                                                                                                                     |
| 87008 |     |                               | "job %s at %s\n", <i>at_job_id</i> , < <i>date</i> >                                                                                                                                                                                                                                                                                                                                                             |
| 87009 |     |                               | where <i>date</i> shall be equivalent in format to the output of:                                                                                                                                                                                                                                                                                                                                                |
| 87010 |     |                               | date +"%a %b %e %T %Y"                                                                                                                                                                                                                                                                                                                                                                                           |
| 87011 |     |                               | The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the <i>TZ</i> variable).                                                                                                                                                                                                                                                                           |
| 87012 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87013 |     |                               | Neither this, nor warning messages concerning the selection of the command interpreter, are considered a diagnostic that changes the exit status.                                                                                                                                                                                                                                                                |
| 87014 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87015 |     |                               | Diagnostic messages, if any, shall be written to standard error.                                                                                                                                                                                                                                                                                                                                                 |
| 87016 |     | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87017 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 87018 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87019 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 87020 |     | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87021 |     |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                     |
| 87022 |     |                               | 0 Successful completion.                                                                                                                                                                                                                                                                                                                                                                                         |
| 87023 |     |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                                                                            |
| 87024 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 87025 |     |                               | The job shall not be scheduled.                                                                                                                                                                                                                                                                                                                                                                                  |



87026 **APPLICATION USAGE**

87027 It may be useful to redirect standard output within the specified commands.

87028 **EXAMPLES**

87029 1. This sequence can be used at a terminal:

```
87030     batch
87031     sort < file >outfile
87032     EOT
```

87033 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
87034 command procedure (the sequence of output redirection specifications is significant):

```
87035     batch <<!
87036     diff file1 file2 2>&1 >outfile | mailx -s "outfile update" mygroup
87037     !
```

87038 Note that this always sends mail when there has been an attempt to update **outfile** and  
87039 the body of the message will be empty unless an error occurred.

87040 3. The following shows how to capture both standard error and standard output:

```
87041     batch <<EOF
87042     {
87043         run-batch-processing |
87044         mailx -s "batch processing output" mygroup
87045     } 2>&1 | mailx -E -s "errors during batch processing" mygroup
87046     EOF
```

87047 **RATIONALE**87048 Early proposals described *batch* in a manner totally separated from *at*, even though the historical  
87049 model treated it almost as a synonym for *at -qb*. A number of features were added to list and  
87050 control batch work separately from those in *at*. Upon further reflection, it was decided that the  
87051 benefit of this did not merit the change to the historical interface.87052 The **-m** option was included on the equivalent *at* command because it is historical practice to  
87053 mail results to the submitter, even if all job-produced output is redirected. As explained in the  
87054 RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling  
87055 delays), despite some historical systems where *at now* would have been considered an error.87056 **FUTURE DIRECTIONS**

87057 None.

87058 **SEE ALSO**87059 [\*at\*](#)87060 [XBD Chapter 8](#) (on page 167)87061 **CHANGE HISTORY**

87062 First released in Issue 2.

87063 **Issue 6**

87064 This utility is marked as part of the User Portability Utilities option.

87065 The NAME is changed to align with the IEEE P1003.2b draft standard.

87066 The normative text is reworded to avoid use of the term “must” for application requirements.

87067 **Issue 7**

87068 The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability  
87069 Utilities is now an option for interactive utilities.

87070 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
87071 by the *batch* utility.

87072 **Issue 8**

87073 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

87074 Austin Group Defect 1368 is applied, changing the EXAMPLES section.

87075 **NAME**

87076 bc — arbitrary-precision arithmetic language

87077 **SYNOPSIS**87078 bc [-l] [*file...*]87079 **DESCRIPTION**

87080 The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files  
 87081 given, then read from the standard input. If the standard input and standard output to *bc* are  
 87082 attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing  
 87083 behavioral constraints described in the following sections.

87084 **OPTIONS**87085 The *bc* utility shall conform to XBD [Section 12.2](#) (on page 215).

87086 The following option shall be supported:

87087 **-l** (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the  
 87088 default zero; see the EXTENDED DESCRIPTION section.

87089 **OPERANDS**

87090 The following operand shall be supported:

87091 *file* A pathname of a text file containing *bc* program statements. After all *files* have  
 87092 been read, *bc* shall read the standard input.

87093 **STDIN**

87094 See the INPUT FILES section.

87095 **INPUT FILES**

87096 Input files shall be text files containing a sequence of comments, statements, and function  
 87097 definitions that shall be executed as they are read.

87098 **ENVIRONMENT VARIABLES**87099 The following environment variables shall affect the execution of *bc*:

87100 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 87101 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 87102 variables used to determine the values of locale categories.)

87103 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 87104 internationalization variables.

87105 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 87106 characters (for example, single-byte as opposed to multi-byte characters in  
 87107 arguments and input files).

87108 **LC\_MESSAGES**

87109 Determine the locale that should be used to affect the format and contents of  
 87110 diagnostic messages written to standard error.

87111 **XSI** **NLSPATH** Determine the location of messages objects and message catalogs.87112 **ASYNCHRONOUS EVENTS**

87113 Default.

87114 **STDOUT**

87115 The output of the *bc* utility shall be controlled by the program read, and consist of zero or more  
 87116 lines containing the value of all executed expressions without assignments. The radix and  
 87117 precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the  
 87118 EXTENDED DESCRIPTION section.

87119 **STDERR**

87120 The standard error shall be used only for diagnostic messages.

87121 **OUTPUT FILES**

87122 None.

87123 **EXTENDED DESCRIPTION**87124 **Grammar**

87125 The grammar in this section and the lexical conventions in the following section shall together  
 87126 describe the syntax for *bc* programs. The general conventions for this style of grammar are  
 87127 described in [Section 1.3](#) (on page 2461). A valid program can be represented as the non-terminal  
 87128 symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax  
 87129 description.

```

87130 %token      EOF NEWLINE STRING LETTER NUMBER
87131 %token      MUL_OP
87132 /*         '*' , '/' , '%'                               */
87133 %token      ASSIGN_OP
87134 /*         '=' , '+=' , '-=' , '*=' , '/=' , '%=' , '^=' */
87135 %token      REL_OP
87136 /*         '==' , '<=' , '>=' , '!=' , '<' , '>'           */
87137 %token      INCR_DECR
87138 /*         '++' , '--'                                   */
87139 %token      Define      Break      Quit      Length
87140 /*         'define' , 'break' , 'quit' , 'length'       */
87141 %token      Return      For        If        While      Sqrt
87142 /*         'return' , 'for' , 'if' , 'while' , 'sqrt'   */
87143 %token      Scale      Ibase      Obase      Auto
87144 /*         'scale' , 'ibase' , 'obase' , 'auto'        */
87145 %start      program
87146 %%
87147 program      : EOF
87148              | input_item program
87149              ;
87150 input_item   : semicolon_list NEWLINE
87151              | function
87152              ;
87153 semicolon_list : /* empty */
87154                | statement
87155                | semicolon_list ';' statement
87156                | semicolon_list ';'
87157                ;
87158 statement_list : /* empty */
87159                | statement
87160                | statement_list NEWLINE

```

```

87161         | statement_list NEWLINE statement
87162         | statement_list ';'
87163         | statement_list ';' statement
87164         ;
87165     statement      : expression
87166         | STRING
87167         | Break
87168         | Quit
87169         | Return
87170         | Return '(' return_expression ')'
87171         | For '(' expression ';'
87172           relational_expression ';'
87173           expression ')' statement
87174         | If '(' relational_expression ')' statement
87175         | While '(' relational_expression ')' statement
87176         | '{' statement_list '}'
87177         ;
87178     function        : Define LETTER '(' opt_define_list ')'
87179         | '{' NEWLINE opt_auto_define_list
87180         | statement_list '}'
87181         ;
87182     opt_define_list : /* empty */
87183         | define_list
87184         ;
87185     opt_auto_define_list : /* empty */
87186         | Auto define_list NEWLINE
87187         | Auto define_list ';'
87188         ;
87189     define_list     : LETTER
87190         | LETTER '[' ']'
87191         | define_list ',' LETTER
87192         | define_list ',' LETTER '[' ']'
87193         ;
87194     opt_argument_list : /* empty */
87195         | argument_list
87196         ;
87197     argument_list   : expression
87198         | expression ',' argument_list
87199         | LETTER '[' ']'
87200         | LETTER '[' ']' ',' argument_list
87201         ;
87202     relational_expression : expression
87203         | expression REL_OP expression
87204         ;
87205     return_expression : /* empty */
87206         | expression
87207         ;

```

```

87208     expression      : named_expression
87209                       | NUMBER
87210                       | '(' expression ')'
87211                       | LETTER '(' opt_argument_list ')'
87212                       | '-' expression
87213                       | expression '+' expression
87214                       | expression '-' expression
87215                       | expression MUL_OP expression
87216                       | expression '^' expression
87217                       | INCR_DECR named_expression
87218                       | named_expression INCR_DECR
87219                       | named_expression ASSIGN_OP expression
87220                       | Length '(' expression ')'
87221                       | Sqrt '(' expression ')'
87222                       | Scale '(' expression ')'
87223                       ;
87224     named_expression  : LETTER
87225                       | LETTER '[' expression ']'
87226                       | Scale
87227                       | Ibase
87228                       | Obase
87229                       ;

```

### 87230 Lexical Conventions in bc

87231 The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as  
 87232 follows:

- 87233 1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a  
 87234 given point.
- 87235 2. A comment shall consist of any characters beginning with the two adjacent characters  
 87236 `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`.  
 87237 Comments shall have no effect except to delimit lexical tokens.
- 87238 3. The `<newline>` shall be recognized as the token **NEWLINE**.
- 87239 4. The token **STRING** shall represent a string constant; it shall consist of any characters  
 87240 beginning with the double-quote character (`"`) and terminated by another occurrence  
 87241 of the double-quote character. The value of the string is the sequence of all characters  
 87242 between, but not including, the two double-quote characters. All characters shall be taken  
 87243 literally from the input, and there is no way to specify a string containing a double-quote  
 87244 character. The length of the value of each string shall be limited to `{BC_STRING_MAX}`  
 87245 bytes.
- 87246 5. A `<blank>` shall have no effect except as an ordinary character if it appears within a  
 87247 **STRING** token, or to delimit a lexical token other than **STRING**.
- 87248 6. The combination of a `<backslash>` character immediately followed by a `<newline>` shall  
 87249 have no effect other than to delimit lexical tokens with the following exceptions:
  - 87250 • It shall be interpreted as the character sequence `"\<newline>"` in **STRING** tokens.

- 87251                   • It shall be ignored as part of a multi-line **NUMBER** token.
- 87252           7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the  
87253           following grammar:
- 87254           **NUMBER** : integer  
87255                   | '.' integer  
87256                   | integer '.'  
87257                   | integer '.' integer  
87258                   ;  
  
87259           integer : digit  
87260                   | integer digit  
87261                   ;  
  
87262           digit : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7  
87263                   | 8 | 9 | A | B | C | D | E | F  
87264                   ;  
  
87265           8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by  
87266           the value of the internal register **ibase** (described below). Each of the **digit** characters  
87267           shall have the value from 0 to 15 in the order listed here, and the <period> character shall  
87268           represent the radix point. The behavior is undefined if digits greater than or equal to the  
87269           value of **ibase** appear in the token. However, note the exception for single-digit values  
87270           being assigned to **ibase** and **obase** themselves, in [Operations in bc](#) (on page 2656).  
  
87271           9. The following keywords shall be recognized as tokens:
- 87272           **auto**       **ibase**       **length**      **return**      **while**  
87273           **break**      **if**        **obase**       **scale**  
87274           **define**     **for**       **quit**       **sqrt**  
  
87275           10. Any of the following characters occurring anywhere except within a keyword shall be  
87276           recognized as the token **LETTER**:
- 87277           a b c d e f g h i j k l m n o p q r s t u v w x y z  
  
87278           11. The following single-character and two-character sequences shall be recognized as the  
87279           token **ASSIGN\_OP**:
- 87280           =   +=   -=   \*=   /=   %=   ^=  
  
87281           12. If an '=' character, as the beginning of a token, is followed by a '-' character with no  
87282           intervening delimiter, the behavior is undefined.  
  
87283           13. The following single-characters shall be recognized as the token **MUL\_OP**:
- 87284           \*   /   %  
87285           14. The following single-character and two-character sequences shall be recognized as the  
87286           token **REL\_OP**:
- 87287           ==   <=   >=   !=   <   >  
  
87288           15. The following two-character sequences shall be recognized as the token **INCR\_DECR**:
- 87289           ++   --  
  
87290           16. The following single characters shall be recognized as tokens whose names are the  
87291           character:

87292 <newline> ( ) , + - ; [ ] ^ { }

87293 17. The token **EOF** is returned when the end of input is reached.

### 87294 Operations in bc

87295 There are three kinds of identifiers: ordinary identifiers, array identifiers, and function  
87296 identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed  
87297 by square brackets ("[]"). An array subscript is required except in an argument or auto list.  
87298 Arrays are singly dimensioned and can contain up to {BC\_DIM\_MAX} elements. Indexing shall  
87299 begin at zero so an array is indexed from 0 to {BC\_DIM\_MAX}-1. Subscripts shall be truncated  
87300 to integers. The application shall ensure that function identifiers are followed by parentheses,  
87301 possibly enclosing arguments. The three types of identifiers do not conflict.

87302 The following table summarizes the rules for precedence and associativity of all operators.  
87303 Operators on the same line shall have the same precedence; rows are in order of decreasing  
87304 precedence.

87305 **Table 3-3** Operators in *bc*

| Operator                  | Associativity |
|---------------------------|---------------|
| ++, --                    | N/A           |
| unary -                   | N/A           |
| ^                         | Right to left |
| *, /, %                   | Left to right |
| +, binary -               | Left to right |
| =, +=, -=, *=, /=, %=, ^= | Right to left |
| ==, <=, >=, !=, <, >      | None          |

87314 Each expression or named expression has a *scale*, which is the number of decimal digits that  
87315 shall be maintained as the fractional portion of the expression.

87316 *Named expressions* are places where values are stored. Named expressions shall be valid on the  
87317 left side of an assignment. The value of a named expression shall be the value stored in the place  
87318 named. Simple identifiers and array elements are named expressions; they have an initial value  
87319 of zero and an initial scale of zero.

87320 The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an  
87321 expression consisting of the name of one of these registers shall be zero; values assigned to any  
87322 of these registers are truncated to integers. The **scale** register shall contain a global value used in  
87323 computing the scale of expressions (as described below). The value of the register **scale** is  
87324 limited to  $0 \leq \text{scale} \leq \{\text{BC\_SCALE\_MAX}\}$  and shall have a default value of zero. The **ibase** and  
87325 **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be  
87326 limited to:

87327  $2 \leq \text{ibase} \leq 16$

87328 The value of **obase** shall be limited to:

87329  $2 \leq \text{obase} \leq \{\text{BC\_BASE\_MAX}\}$

87330 When either **ibase** or **obase** is assigned a single **digit** value from the list in [Lexical Conventions](#)  
87331 [in bc](#) (on page 2654), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to  
87332 base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when  
87333 digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall  
87334 have initial values of 10.



87335 Internal computations shall be conducted as if in decimal, regardless of the input and output  
87336 bases, to the specified number of decimal digits. When an exact result is not achieved (for  
87337 example, **scale**=0; 3.2/1), the result shall be truncated.

87338 For all values of **obase** specified by this volume of POSIX.1-2024, *bc* shall output numeric values  
87339 by performing each of the following steps in order:

- 87340 1. If the value is less than zero, a <hyphen-minus> ('-') character shall be output.
- 87341 2. One of the following is output, depending on the numerical value:
  - 87342 • If the absolute value of the numerical value is greater than or equal to one, the  
87343 integer portion of the value shall be output as a series of digits appropriate to **obase**  
87344 (as described below), most significant digit first. The most significant non-zero digit  
87345 shall be output next, followed by each successively less significant digit.
  - 87346 • If the absolute value of the numerical value is less than one but greater than zero  
87347 and the scale of the numerical value is greater than zero, it is unspecified whether  
87348 the character 0 is output.
  - 87349 • If the numerical value is zero, the character 0 shall be output.
- 87350 3. If the scale of the value is greater than zero and the numeric value is not zero, a <period>  
87351 character shall be output, followed by a series of digits appropriate to **obase** (as described  
87352 below) representing the most significant portion of the fractional part of the value. If *s*  
87353 represents the scale of the value being output, the number of digits output shall be *s* if  
87354 **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s*  
87355 if **obase** is less than 10. For **obase** values other than 10, this should be the number of  
87356 digits needed to represent a precision of 10<sup>*s*</sup>.

87357 For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

87358 0 1 2 3 4 5 6 7 8 9 A B C D E F

87359 which represent the values zero to 15, inclusive, respectively.

87360 For bases greater than 16, each digit shall be written as a separate multi-digit decimal number.  
87361 Each digit except the most significant fractional digit shall be preceded by a single <space>. For  
87362 bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1 000, three-  
87363 digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would be  
87364 written as:

87365 Δ01Δ15Δ24

87366 and in base 125, as:

87367 Δ008Δ024

87368 Very large numbers shall be split across lines with 70 characters per line in the POSIX locale;  
87369 other locales may split at different character boundaries. Lines that are continued shall end with  
87370 a <backslash>.

87371 A function call shall consist of a function name followed by parentheses containing a  
87372 <comma>-separated list of expressions, which are the function arguments. A whole array  
87373 passed as an argument shall be specified by the array name followed by empty square brackets.  
87374 All function arguments shall be passed by value. As a result, changes made to the formal  
87375 parameters shall have no effect on the actual arguments. If the function terminates by executing  
87376 a **return** statement, the value of the function shall be the value of the expression in the  
87377 parentheses of the **return** statement or shall be zero if no expression is provided or if there is no  
87378 **return** statement.

87379 The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be  
 87380 truncated in the least significant decimal place. The scale of the result shall be the scale of the  
 87381 expression or the value of **scale**, whichever is larger.

87382 The result of **length**(*expression*) shall be the total number of significant decimal digits in the  
 87383 expression. The scale of the result shall be zero.

87384 The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be  
 87385 zero.

87386 A numeric constant shall be an expression. The scale shall be the number of digits that follow the  
 87387 radix point in the input representing the constant, or zero if no radix point appears.

87388 The sequence ( *expression* ) shall be an expression with the same value and scale as *expression*.  
 87389 The parentheses can be used to alter the normal precedence.

87390 The semantics of the unary and binary operators are as follows:

87391 *-expression*

87392 The result shall be the negative of the *expression*. The scale of the result shall be the scale of  
 87393 *expression*.

87394 The unary increment and decrement operators shall not modify the scale of the named  
 87395 expression upon which they operate. The scale of the result shall be the scale of that named  
 87396 expression.

87397 *++named-expression*

87398 The named expression shall be incremented by one. The result shall be the value of the  
 87399 named expression after incrementing.

87400 *--named-expression*

87401 The named expression shall be decremented by one. The result shall be the value of the  
 87402 named expression after decrementing.

87403 *named-expression++*

87404 The named expression shall be incremented by one. The result shall be the value of the  
 87405 named expression before incrementing.

87406 *named-expression--*

87407 The named expression shall be decremented by one. The result shall be the value of the  
 87408 named expression before decrementing.

87409 The exponentiation operator, <circumflex> ('^'), shall bind right to left.

87410 *expression^expression*

87411 The result shall be the first *expression* raised to the power of the second *expression*. If the  
 87412 second expression is not an integer, the behavior is undefined. If *a* is the scale of the left  
 87413 expression and *b* is the absolute value of the right expression, the scale of the result shall be:

87414 if  $b \geq 0$   $\min(a * b, \max(\text{scale}, a))$  if  $b < 0$   $\text{scale}$

87415 The multiplicative operators ('\*', '/', '%') shall bind left to right.

87416 *expression\*expression*

87417 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two  
 87418 expressions, then the scale of the result shall be:

87419  $\min(a+b, \max(\text{scale}, a, b))$

87420 *expression/expression*  
 87421 The result shall be the quotient of the two expressions. The scale of the result shall be the  
 87422 value of **scale**.

87423 *expression%expression*  
 87424 For expressions *a* and *b*, *a%b* shall be evaluated equivalent to the steps:

87425 1. Compute *a/b* to current scale.

87426 2. Use the result to compute:

87427  $a - (a / b) * b$

87428 to scale:

87429  $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

87430 The scale of the result shall be:

87431  $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

87432 When **scale** is zero, the '%' operator is the mathematical remainder operator.

87433 The additive operators ('+', '-') shall bind left to right.

87434 *expression+expression*

87435 The result shall be the sum of the two expressions. The scale of the result shall be the  
 87436 maximum of the scales of the expressions.

87437 *expression-expression*

87438 The result shall be the difference of the two expressions. The scale of the result shall be the  
 87439 maximum of the scales of the expressions.

87440 The assignment operators ('=', "+=", "-=", "\*=", "/=", "%=", "^=") shall bind right to left.

87441 *named-expression=expression*

87442 This expression shall result in assigning the value of the expression on the right to the  
 87443 named expression on the left. The scale of both the named expression and the result shall be  
 87444 the scale of *expression*.

87445 The compound assignment forms:

87446 *named-expression <operator>= expression*

87447 shall be equivalent to:

87448 *named-expression=named-expression <operator> expression*

87449 except that the *named-expression* shall be evaluated only once.

87450 Unlike all other operators, the relational operators ('<', '>', "<=", ">=", "==", "!=") shall be  
 87451 only valid as the object of an **if**, **while**, or inside a **for** statement.

87452 *expression1<expression2*

87453 The relation shall be true if the value of *expression1* is strictly less than the value of  
 87454 *expression2*.

87455 *expression1>expression2*

87456 The relation shall be true if the value of *expression1* is strictly greater than the value of  
 87457 *expression2*.

87458 *expression1*<=*expression2*  
 87459 The relation shall be true if the value of *expression1* is less than or equal to the value of  
 87460 *expression2*.

87461 *expression1*>=*expression2*  
 87462 The relation shall be true if the value of *expression1* is greater than or equal to the value of  
 87463 *expression2*.

87464 *expression1*==*expression2*  
 87465 The relation shall be true if the values of *expression1* and *expression2* are equal.

87466 *expression1*!=*expression2*  
 87467 The relation shall be true if the values of *expression1* and *expression2* are unequal.

87468 There are only two storage classes in *bc*: global and automatic (local). Only identifiers that are  
 87469 local to a function need be declared with the **auto** command. The arguments to a function shall  
 87470 be local to the function. All other identifiers are assumed to be global and available to all  
 87471 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto  
 87472 shall be allocated on entry to the function and released on returning from the function. They  
 87473 therefore do not retain values between function calls. Auto arrays shall be specified by the array  
 87474 name followed by empty square brackets. On entry to a function, the old values of the names  
 87475 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the  
 87476 function returns, reference to these names shall refer only to the new values.

87477 References to any of these names from other functions that are called from this function also  
 87478 refer to the new value until one of those functions uses the same name for a local variable.

87479 When a statement is an expression, unless the main operator is an assignment, execution of the  
 87480 statement shall write the value of the expression followed by a <newline>.

87481 When a statement is a string, execution of the statement shall write the value of the string.

87482 Statements separated by <semicolon> or <newline> characters shall be executed sequentially. In  
 87483 an interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical  
 87484 production:

```
87485 input_item : semicolon_list NEWLINE
```

87486 the sequential list of statements making up the **semicolon\_list** shall be executed immediately  
 87487 and any output produced by that execution shall be written without any delay due to buffering.

87488 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

87489 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;  
 87490 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the  
 87491 *relation* is false, execution shall resume after *statement*.

87492 A **for** statement(**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

```
87493 first-expression  

  87494 while (relation) {  

  87495     statement  

  87496     last-expression  

  87497 }
```

87498 The application shall ensure that all three expressions are present.

87499 The **break** statement shall cause termination of a **for** or **while** statement.

87500 The **auto** statement (**auto** *identifier* [*,identifier*] ...) shall cause the values of the identifiers to be

87501 pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers  
 87502 shall be specified by following the array name by empty square brackets. The application shall  
 87503 ensure that the **auto** statement is the first statement in a function definition.

87504 A **define** statement:

```
87505 define LETTER ( opt_define_list ) {
87506     opt_auto_define_list
87507     statement_list
87508 }
```

87509 defines a function named **LETTER**. If a function named **LETTER** was previously defined, the  
 87510 **define** statement shall replace the previous definition. The expression:

```
87511 LETTER ( opt_argument_list )
```

87512 shall invoke the function named **LETTER**. The behavior is undefined if the number of  
 87513 arguments in the invocation does not match the number of parameters in the definition.  
 87514 Functions shall be defined before they are invoked. A function shall be considered to be defined  
 87515 within its own body, so recursive calls are valid. The values of numeric constants within a  
 87516 function shall be interpreted in the base specified by the value of the **ibase** register when the  
 87517 function is invoked.

87518 The **return** statements (**return** and **return(expression)**) shall cause termination of a function,  
 87519 popping of its auto variables, and specification of the result of the function. The first form shall  
 87520 be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the  
 87521 value and scale of the expression returned.

87522 The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement  
 87523 occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

87524 The following functions shall be defined when the **-I** option is specified:

```
87525 s( expression )
87526     Sine of argument in radians.
```

```
87527 c( expression )
87528     Cosine of argument in radians.
```

```
87529 a( expression )
87530     Arctangent of argument.
```

```
87531 l( expression )
87532     Natural logarithm of argument.
```

```
87533 e( expression )
87534     Exponential function of argument.
```

```
87535 j( expression1, expression2 )
87536     Bessel function of expression2 of the first kind of integer order expression1.
```

87537 The scale of the result returned by these functions shall be the value of the **scale** register at the  
 87538 time the function is invoked. The value of the **scale** register after these functions have completed  
 87539 their execution shall be the same value it had upon invocation. The behavior is undefined if any  
 87540 of these functions is invoked with an argument outside the domain of the mathematical  
 87541 function.

87542 **EXIT STATUS**

87543 The following exit values shall be returned:

87544 0 All input files were processed successfully.

87545 *unspecified* An error occurred.87546 **CONSEQUENCES OF ERRORS**87547 If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.87549 In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.87552 **APPLICATION USAGE**87553 Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.87554 For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.87558 The *bc* utility always uses the <period> ( '.' ) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the <period> character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a <comma> as the decimal-point character:87565 

```
define f(a,b) {
87566     ...
87567 }
87568 ...
87569 f(1, 2, 3)
```

87570 Because of such ambiguities, the &lt;period&gt; character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the &lt;period&gt; is also used in output.

87573 **EXAMPLES**87574 In the shell, the following assigns an approximation of the first ten digits of ' $\pi$ ' to the variable *x*:87575 

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

87576 The following *bc* program prints the same approximation of ' $\pi$ ', with a label, to standard output:87578 

```
scale = 10
87579 "pi equals "
87580 104348 / 33215
```

87581 The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the *-I* option is specified):87583 

```
scale = 20
87584 define e(x){
87585     auto a, b, c, i, s
```

```

87586     a = 1
87587     b = 1
87588     s = 1
87589     for (i = 1; 1 == 1; i++){
87590         a = a*x
87591         b = b*i
87592         c = a/b
87593         if (c == 0) {
87594             return(s)
87595         }
87596         s = s+c
87597     }
87598 }

```

87599 The following prints approximate values of the exponential function of the first ten integers:

```

87600 for (i = 1; i <= 10; ++i) {
87601     e(i)
87602 }

```

#### 87603 RATIONALE

87604 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to  
 87605 be part of this volume of POSIX.1-2024 because *bc* was thought to have a more intuitive  
 87606 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be  
 87607 compliant.

87608 The exit status for error conditions has been left unspecified for several reasons:

- 87609 • The *bc* utility is used in both interactive and non-interactive situations. Different exit codes  
 87610 may be appropriate for the two uses.
- 87611 • It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions,  
 87612 and syntax errors are all possibilities.
- 87613 • It is not clear what utility the exit status has.
- 87614 • In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with  
 87615 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*  
 87616 aborted.

87617 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief  
 87618 that *bc file1 file2* is used most often when at least *file1* contains data/function  
 87619 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not  
 87620 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check  
 87621 all its files for accessibility before opening any of them.

87622 There was considerable debate on the appropriateness of the language accepted by *bc*. Several  
 87623 reviewers preferred to see either a pure subset of the C language or some changes to make the  
 87624 language more compatible with C. While the *bc* language has some obvious similarities to C, it  
 87625 has never claimed to be compatible with any version of C. An interpreter for a subset of C might  
 87626 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility  
 87627 is known in historical practice, and it was not within the scope of this volume of POSIX.1-2024 to  
 87628 define such a language and utility. If and when they are defined, it may be appropriate to  
 87629 include them in a future version of this standard. This left the following alternatives:

- 87630 1. Exclude any calculator language from this volume of POSIX.1-2024.
- 87631 The consensus of the standard developers was that a simple programmatic calculator  
87632 language is very useful for both applications and interactive users. The only arguments  
87633 for excluding any calculator were that it would become obsolete if and when a C-  
87634 compatible one emerged, or that the absence would encourage the development of such a  
87635 C-compatible one. These arguments did not sufficiently address the needs of current  
87636 application developers.
- 87637 2. Standardize the historical *dc*, possibly with minor modifications.
- 87638 The consensus of the standard developers was that *dc* is a fundamentally less usable  
87639 language and that that would be far too severe a penalty for avoiding the issue of being  
87640 similar to but incompatible with C.
- 87641 3. Standardize the historical *bc*, possibly with minor modifications.
- 87642 This was the approach taken. Most of the proponents of changing the language would not  
87643 have been satisfied until most or all of the incompatibilities with C were resolved. Since  
87644 most of the changes considered most desirable would break historical applications and  
87645 require significant modification to historical implementations, almost no modifications  
87646 were made. The one significant modification that was made was the replacement of the  
87647 historical *bc* assignment operators "=", and so on, with the more modern "+=", and so  
87648 on. The older versions are considered to be fundamentally flawed because of the lexical  
87649 ambiguity in uses like  $a=-1$ .
- 87650 In order to permit implementations to deal with backwards-compatibility as they see fit,  
87651 the behavior of this one ambiguous construct was made undefined. (At least three  
87652 implementations have been known to support this change already, so the degree of  
87653 change involved should not be great.)
- 87654 The '%' operator is the mathematical remainder operator when **scale** is zero. The behavior of  
87655 this operator for other values of **scale** is from historical implementations of *bc*, and has been  
87656 maintained for the sake of historical applications despite its non-intuitive nature.
- 87657 Historical implementations permit setting **ibase** and **obase** to a broader range of values. This  
87658 includes values less than 2, which were not seen as sufficiently useful to standardize. These  
87659 implementations do not interpret input properly for values of **ibase** that are greater than 16. This  
87660 is because numeric constants are recognized syntactically, rather than lexically, as described in  
87661 this volume of POSIX.1-2024. They are built from lexical tokens of single hexadecimal digits and  
87662 <period> characters. Since <blank> characters between tokens are not visible at the syntactic  
87663 level, it is not possible to recognize the multi-digit "digits" used in the higher bases properly.  
87664 The ability to recognize input in these bases was not considered useful enough to require  
87665 modifying these implementations. Note that the recognition of numeric constants at the  
87666 syntactic level is not a problem with conformance to this volume of POSIX.1-2024, as it does not  
87667 impact the behavior of conforming applications (and correct *bc* programs). Historical  
87668 implementations also accept input with all of the digits '0'-'9' and 'A'-'F' regardless of the  
87669 value of **ibase**; since digits with value greater than or equal to **ibase** are not really appropriate,  
87670 the behavior when they appear is undefined, except for the common case of:
- ```
87671 ibase=8;
87672     /* Process in octal base. */
87673     ...
87674 ibase=A
87675     /* Restore decimal base. */
```
- 87676 In some historical implementations, if the expression to be written is an uninitialized array



87677 element, a leading <space> and/or up to four leading 0 characters may be output before the  
 87678 character zero. This behavior is considered a bug; it is unlikely that any currently conforming  
 87679 application relies on:

```
87680 echo 'b[3]' | bc
```

87681 returning 00000 rather than 0.

87682 Exact calculation of the number of fractional digits to output for a given value in a base other  
 87683 than 10 can be computationally expensive. Historical implementations use a faster  
 87684 approximation, and this is permitted. Note that the requirements apply only to values of **obase**  
 87685 that this volume of POSIX.1-2024 requires implementations to support (in particular, not to 1, 0,  
 87686 or negative bases, if an implementation supports them as an extension).

87687 Historical implementations of *bc* did not allow array parameters to be passed as the last  
 87688 parameter to a function. When *bc* was first standardized in Issue 4, this restriction was allowed.  
 87689 To make *bc* more widely useful, and because there are implementations without this restriction,  
 87690 the allowance for the restriction has been removed.

#### 87691 FUTURE DIRECTIONS

87692 None.

#### 87693 SEE ALSO

87694 [Section 1.3](#) (on page 2461), *awk*

87695 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 87696 CHANGE HISTORY

87697 First released in Issue 4.

#### 87698 Issue 5

87699 The FUTURE DIRECTIONS section is added.

#### 87700 Issue 6

87701 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several  
 87702 interpretations of the ISO POSIX-2: 1993 standard.

87703 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 87704 Issue 7

87705 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

87706 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0066 [584] and XCU/TC2-2008/0067  
 87707 [679] are applied.

#### 87708 Issue 8

87709 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

87710 Austin Group Defect 1230 is applied, changing the EXTENDED DESCRIPTION section to  
 87711 specify that array parameters can be passed as the last parameter to a function.

87712 Austin Group Defect 1570 is applied, removing extra spacing in "==".

87713 **NAME**87714 `bg` — run jobs in the background87715 **SYNOPSIS**87716 UP `bg [job_id...]`87717 **DESCRIPTION**

87718 If job control is enabled (see the description of `set -m`), the shell is interactive, and the current  
 87719 shell execution environment (see [Section 2.13](#), on page 2522) is not a subshell environment, the  
 87720 `bg` utility shall resume suspended jobs from the current shell execution environment by running  
 87721 them as background jobs, as described in [Section 2.11](#) (on page 2518); it may also do so if the  
 87722 shell is non-interactive or the current shell execution environment is a subshell environment. If  
 87723 the job specified by `job_id` is already a running background job, the `bg` utility shall have no effect  
 87724 and shall exit successfully.

87725 **OPTIONS**

87726 None.

87727 **OPERANDS**

87728 The following operand shall be supported:

87729 `job_id` Specify the job to be resumed as a background job. If no `job_id` operand is given,  
 87730 the most recently suspended job shall be used. The format of `job_id` is described in  
 87731 XBD [Section 3.182](#) (on page 57).

87732 **STDIN**

87733 Not used.

87734 **INPUT FILES**

87735 None.

87736 **ENVIRONMENT VARIABLES**87737 The following environment variables shall affect the execution of `bg`:

87738 `LANG` Provide a default value for the internationalization variables that are unset or null.  
 87739 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 87740 variables used to determine the values of locale categories.)

87741 `LC_ALL` If set to a non-empty string value, override the values of all the other  
 87742 internationalization variables.

87743 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
 87744 characters (for example, single-byte as opposed to multi-byte characters in  
 87745 arguments).

87746 `LC_MESSAGES`

87747 Determine the locale that should be used to affect the format and contents of  
 87748 diagnostic messages written to standard error.

87749 XSI `NLSPATH` Determine the location of messages objects and message catalogs.87750 **ASYNCHRONOUS EVENTS**

87751 Default.

87752 **STDOUT**87753 The output of `bg` shall consist of a line in the format:87754 `"[%d] %s\n", <job-number>, <command>`

87755 where the fields are as follows:

87756            <*job-number*> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using  
87757            these utilities, the job can be identified by prefixing the job number with '% '.

87758            <*command*> The associated command that was given to the shell.

#### 87759 **STDERR**

87760            The standard error shall be used only for diagnostic messages.

#### 87761 **OUTPUT FILES**

87762            None.

#### 87763 **EXTENDED DESCRIPTION**

87764            None.

#### 87765 **EXIT STATUS**

87766            The following exit values shall be returned:

87767            0 Successful completion.

87768            >0 An error occurred.

#### 87769 **CONSEQUENCES OF ERRORS**

87770            If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the  
87771            background.

#### 87772 **APPLICATION USAGE**

87773            This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

87774            A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see  
87775            XBD [Chapter 11](#) (on page 199). At that point, *bg* can put the job into the background. This is  
87776            most effective when the job is expecting no terminal input and its output has been redirected to  
87777            non-terminal files. A background job can be forced to stop when it has terminal output by  
87778            issuing the command:

```
87779            stty tostop
```

87780            A background job can be stopped with the command:

```
87781            kill -s stop job ID
```

87782            The *bg* utility does not work as expected when it is operating in its own utility execution  
87783            environment because that environment has no suspended jobs. In the following examples:

```
87784            ... | xargs bg  
87785            (bg)
```

87786            each *bg* operates in a different environment and does not share its parent shell's understanding  
87787            of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

#### 87788 **EXAMPLES**

87789            None.

#### 87790 **RATIONALE**

87791            The extensions to the shell specified in this volume of POSIX.1-2024 have mostly been based on  
87792            features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also  
87793            based on the KornShell. The standard developers examined the characteristics of the C shell  
87794            versions of these utilities and found that differences exist. Despite widespread use of the C shell,  
87795            the KornShell versions were selected for this volume of POSIX.1-2024 to maintain a degree of  
87796            uniformity with the rest of the KornShell features selected (such as the very popular command  
87797            line editing features).

- 87798 The *bg* utility is expected to wrap its output if the output exceeds the number of display  
87799 columns.
- 87800 The *bg* and *fg* utilities are not symmetric as regards the list of process IDs known in the current  
87801 shell execution environment. Whereas *fg* removes a process ID from this list, *bg* has no need to  
87802 add one to this list when it resumes execution of a suspended job in the background, because  
87803 this has already been done by the shell when the suspended background job was created (see  
87804 [Section 2.11](#), on page 2518).
- 87805 **FUTURE DIRECTIONS**  
87806 None.
- 87807 **SEE ALSO**  
87808 [Section 2.9.3.1](#) (on page 2506), *fg*, *kill*, *jobs*, *wait*  
87809 XBD [Section 3.182](#) (on page 57), [Chapter 8](#) (on page 167), [Chapter 11](#) (on page 199)
- 87810 **CHANGE HISTORY**  
87811 First released in Issue 4.
- 87812 **Issue 6**  
87813 This utility is marked as part of the User Portability Utilities option.  
87814 The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory  
87815 in this version. This is a FIPS requirement.
- 87816 **Issue 7**  
87817 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 87818 **Issue 8**  
87819 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
87820 this utility is required to be intrinsic.  
87821 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.  
87822 Austin Group Defect 1254 is applied, updating the DESCRIPTION to account for the addition of  
87823 [Section 2.11](#) (on page 2518) and adding a paragraph to RATIONALE.

87824 **NAME**

87825 c17 — compile standard C programs

87826 **SYNOPSIS**

```
87827 CD c17 [options...] pathname [[pathname] [-I directory]
87828 [-L directory] [-l library] [-R directory]]...
```

87829 **DESCRIPTION**

87830 The *c17* utility is an interface to the standard C compilation system; it shall accept source code  
 87831 written in the C language as defined in section 6 of the ISO C standard. The system conceptually  
 87832 consists of a compilation phase, encompassing Translation Phases 1 through 7 of the ISO C  
 87833 standard, and a linkage phase, for handling Phase 8 of the ISO C standard and extensions  
 87834 described here. The reference to “library components” in Phase 8 shall be taken to refer to  
 87835 components of libraries specified using the `-l` option, libraries specified as *file.a* or *file.so*  
 87836 operands, and the equivalent of a `-l c` option passed to the link editor in the manner specified in  
 87837 the EXTENDED DESCRIPTION. In addition, the compilation phase can be split into a separate  
 87838 preprocessing operation, handling Translation Phases 1 through 4, and a processing operation,  
 87839 handling Phases 5 through 7. Whether a single utility or multiple utilities for handling phases  
 87840 separately is provided by an implementation is left unspecified. The input files referenced by  
 87841 *pathname* operands and `-l` option-arguments shall be compiled and linked to produce an  
 87842 executable file or, if the `-G` option is specified, a shared library file. It is unspecified whether the  
 87843 linking of an executable file occurs entirely within the operation of *c17*; when a *pathname*  
 87844 operand or `-l` option-argument names a shared library, an executable object may be produced  
 87845 that is not fully resolved until the file is executed.

87846 If the `-c` option is specified and the `-o` option is not specified, for all *pathname* operands of the  
 87847 form *file.c* or *file.i*, the files:

87848 `$(basename pathname .c).o`

87849 or

87850 `$(basename pathname .i).o`

87851 respectively shall be created as the result of successful compilation. If the `-c` option is not  
 87852 specified, it is unspecified whether such `.o` files are created or deleted for the *file.c* and *file.i*  
 87853 operands.

87854 If there are no options that prevent link editing (such as `-c` or `-E`), and all input files compile and  
 87855 link without error, the resulting executable file or shared library file shall be written according to  
 87856 the `-o outfile` option, if present. If `-o outfile` is not specified, a resulting executable file shall be  
 87857 written to the file **a.out**; if the file to be written is a shared library file, the behavior is  
 87858 unspecified.

87859 Executable files shall be created as specified in [Section 1.1.1.4](#) (on page 2454), except that the file  
 87860 permission bits shall be set to:

87861 `S_IRWXO | S_IRWXG | S_IRWXU`87862 and the bits specified by the *umask* of the process shall be cleared.87863 **OPTIONS**87864 The *c17* utility shall conform to XBD [Section 12.2](#) (on page 215), except that:

- 87865
- Options can be interspersed with operands.

- 87866 • The order of specifying the **-L**, **-I**, and **-R** options, and the order of specifying **-I** options  
87867 with respect to *pathname* operands is significant.
- 87868 • Conforming applications shall specify each option separately; that is, grouping option  
87869 letters (for example, **-cO**) need not be recognized by all implementations.

87870 The following options shall be supported:

- 87871 **-B mode** If *mode* is **dynamic**, produce a dynamically linked executable file. If the **-B** option  
87872 is present with **-c**, **-E**, or **-G**, the result is unspecified.
- 87873 **-c** Suppress the link-edit phase of the compilation, and do not remove any object files  
87874 that are produced. The application shall ensure that all operands are of the form  
87875 *file.c* or *file.i*.
- 87876 **-D name[=value]**  
87877 Define *name* as if by a C-language **#define** directive. If no *=value* is given, a value of  
87878 1 shall be used. The **-D** option has lower precedence than the **-U** option. That is, if  
87879 *name* is used in both a **-U** and a **-D** option, *name* shall be undefined regardless of  
87880 the order of the options. Additional implementation-defined *names* may be  
87881 provided by the compiler. Implementations shall support at least 2 048 bytes of **-D**  
87882 definitions and 256 *names*.
- 87883 **-E** Copy C-language source files to standard output, executing all preprocessor  
87884 directives; no compilation shall be performed. If any operand is not a text file, the  
87885 effects are unspecified.
- 87886 **-G** Create a shared library or create object files suitable for inclusion in such a shared  
87887 library. Compilations shall be performed in a manner suitable for the creation of  
87888 shared libraries (for example, by producing position-independent code).  
87889 If **-c** is also specified, create object files suitable for inclusion in a shared library.  
87890 If **-c** is not specified, create a shared library. In this case the application shall ensure  
87891 that the file named by the **-o outfile** option-argument includes an element named  
87892 **so** or an implementation-defined element denoting a shared library, where  
87893 elements in the last component of *outfile* are separated by <period> characters, for  
87894 example **libx.so.1**; if no **-o** option is included in the options or the file named by  
87895 the **-o outfile** option does not contain an element named **so** or an implementation-  
87896 defined element denoting a shared library, the result is unspecified. If a *pathname*  
87897 operand or **-I** option-argument names a shared library and that shared library  
87898 defines an object used by the library being created, it shall become a dependency  
87899 of the created shared library.  
87900 If the **-G** option is present with **-B** or **-E**, the result is unspecified.
- 87901 **-g** Produce symbolic information in the object or executable files; the nature of this  
87902 information is unspecified, and may be modified by implementation-defined  
87903 interactions with other options.
- 87904 **-I directory** Change the algorithm for searching for headers whose names are not absolute  
87905 pathnames to look in the directory named by the *directory* pathname before looking  
87906 in the usual places. Thus, headers whose names are enclosed in double-quotes (" ")  
87907 shall be searched for first in the directory of the file with the **#include** line, then in  
87908 directories named in **-I** options, and last in the usual places. For headers whose  
87909 names are enclosed in angle brackets ("**<**" "**>**"), the header shall be searched for only  
87910 in directories named in **-I** options and then in the usual places. Directories named  
87911 in **-I** options shall be searched in the order specified. If the **-I** option is used to

- 87912 specify a directory that is one of the usual places searched by default, the results  
87913 are unspecified. Implementations shall support at least ten instances of this option  
87914 in a single *c17* command invocation.
- 87915 **-L *directory*** Change the algorithm of searching for the libraries named in the **-I** objects to look  
87916 in the directory named by the *directory* pathname before looking in the usual  
87917 places. Directories named in **-L** options shall be searched in the order specified. If  
87918 the **-L** option is used to specify a directory that is one of the usual places searched  
87919 by default, the results are unspecified. Implementations shall support at least ten  
87920 instances of this option in a single *c17* command invocation. If a directory specified  
87921 by a **-L** option contains files with names starting with any of the strings "libc.",  
87922 "libl.", "libpthread.", "libm.", "librt.", "libxnet.", or "liby.",  
87923 the results are unspecified.
- 87924 **-I *library*** Search the library named **liblibrary.a** or **liblibrary.so**. When searching for a library,  
87925 the linker shall look at each directory specified by **-L** options that appear on the  
87926 command line before this **-I** option, in the order given, and then the system default  
87927 libraries. If **liblibrary.a** and **liblibrary.so** both exist in a directory, *c17* shall use  
87928 **liblibrary.so** if either **-B dynamic** or **-G** is specified. Once a library has been found  
87929 (shared or static) in a directory, later directories in the list shall not be considered.  
87930 A library shall be searched when its name is encountered, so the placement of a **-I**  
87931 option is significant. Several standard libraries can be specified in this manner, as  
87932 described in the EXTENDED DESCRIPTION section. Implementations may  
87933 recognize implementation-defined suffixes other than **.a** and **.so** as denoting  
87934 libraries.
- 87935 **-O *optlevel*** Specify the level of code optimization. If the *optlevel* option-argument is the digit  
87936 '0', all special code optimizations shall be disabled. If it is the digit '1', the  
87937 nature of the optimization is unspecified. If the **-O** option is omitted, the nature of  
87938 the system's default optimization is unspecified. It is unspecified whether code  
87939 generated in the presence of the **-O 0** option is the same as that generated when  
87940 **-O** is omitted. Other *optlevel* values may be supported.
- 87941 **-o *outfile*** Name the output file to be produced. If the **-o** option is present with **-E**, or with **-c**  
87942 and more than one input file, the result is unspecified.
- 87943 When creating a single object file (by using **-c** with a single input file), use the  
87944 pathname *outfile*, instead of the default *file.o*, for the object file produced.
- 87945 When creating an executable file, use the pathname *outfile*, instead of the default  
87946 **a.out**, for the executable file produced.
- 87947 When creating a shared library, use the pathname *outfile* as the name of the shared  
87948 library. If no **-o outfile** option is specified when creating a shared library, the result  
87949 is unspecified.
- 87950 **-s** Produce object or executable files, or both, from which symbolic and other  
87951 information not required for proper execution using the *exec* family defined in the  
87952 System Interfaces volume of POSIX.1-2024 has been removed (stripped). If both **-g**  
87953 and **-s** options are present, the action taken is unspecified.
- 87954 **-R *directory*** If the object file format supports it, specify a directory to be searched for shared  
87955 libraries when an executable file or shared library being created by *c17* is  
87956 subsequently executed, or loaded using *dlopen()*. If *directory* contains any <colon>  
87957 or <dollar-sign> characters, the behavior is unspecified. If an implementation  
87958 provides a means for setting a default load time search location or locations, the **-R**

- option shall take precedence.
- The directory named by *directory* shall not be searched by a process performing dynamic loading if either of the following are true:
- The real and effective user IDs of that process are different and the directory has write permission for a user ID outside the set of the effective user ID of that process and any implementation-specific user IDs used for directories containing system libraries.
  - The real and effective group IDs of that process are different and the directory has write permission for group IDs other than the effective group ID of that process.
- Directories named in **-R** options shall be searched in the order specified, before the default system library locations are searched.
- If a directory specified by a **-R** option contains files with names starting with any of the strings "libc.", "libl.", "libpthread.", "libm.", "librt.", "libxnet.", or "liby.", the result is unspecified.
- If the **-R** option is present with **-c** or **-E**, the result is unspecified.
- U name** Remove any initial definition of *name*.
- Multiple instances of the **-D**, **-I**, **-L**, **-I**, **-R**, and **-U** options can be specified.

#### 87977 OPERANDS

- The application shall ensure that at least one *pathname* operand is specified. The following forms for *pathname* operands shall be supported:
- file.c* A C-language source file to be compiled and optionally linked.
- file.i* A text file containing the output of *c17 -E*, to be compiled and optionally linked. The processing already performed by *c17 -E* when the file was produced shall not be repeated when the file is compiled.
- file.a* A library of static object files typically produced by the *ar* utility, and referenced during the link-edit phase. Implementations may recognize implementation-defined suffixes other than **.a** as denoting static object file libraries.
- file.so* A library of shared object files typically produced by the *c17* utility with the **-G** option, and referenced during the link-edit phase. Implementations may recognize implementation-defined suffixes other than **.so** as denoting shared object file libraries.
- file.o* An object file produced by *c17 -c* and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than **.o** as denoting object files.
- The processing of other files is implementation-defined.

#### 87995 STDIN

87996 Not used.

#### 87997 INPUT FILES

- Each input file shall be one of the following:
- A text file containing a C-language source program or the output of *c17 -E*



- 88000           • An object file in the format produced by *c17 -c*
- 88001           • A library of object files in the format produced by archiving zero or more object files using  
88002           *ar*
- 88003           • A shared library in the format produced by *c17 -G*

88004           Implementations may supply additional utilities that produce files in these formats. Additional  
88005           input file formats are implementation-defined.

#### 88006 ENVIRONMENT VARIABLES

88007           The following environment variables shall affect the execution of *c17*:

88008           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
88009                       (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
88010                       variables used to determine the values of locale categories.)

88011           *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
88012                       internationalization variables.

88013           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
88014                       characters (for example, single-byte as opposed to multi-byte characters in  
88015                       arguments and input files).

#### 88016 *LC\_MESSAGES*

88017                       Determine the locale that should be used to affect the format and contents of  
88018                       diagnostic messages written to standard error.

88019 XSI        *NLSPATH*   Determine the location of messages objects and message catalogs.

88020           *TMPDIR*       Provide a pathname that should override the default directory for temporary files,  
88021 XSI                       if any. On XSI-conforming systems, provide a pathname that shall override the  
88022                       default directory for temporary files, if any.

#### 88023 ASYNCHRONOUS EVENTS

88024           Default.

#### 88025 STDOUT

88026           If more than one *pathname* operand ending in *.c* or *.i* (or possibly other unspecified suffixes) is  
88027           given, for each such file:

88028           "*%s*:\n", *<pathname>*

88029           may be written. These messages, if written, shall precede the processing of each input file; they  
88030           shall not be written to the standard output if they are written to the standard error, as described  
88031           in the *STDERR* section.

88032           If the *-E* option is specified, the standard output shall be a text file that represents the results of  
88033           the preprocessing stage of the language; it may contain extra information appropriate for  
88034           subsequent compilation passes and shall contain at least one line with the format:

88035           "# *%d* \"*%s*\" \n", *<line>*, *<pathname>*

88036           for each file processed as a result of a **#include** directive, unless no other output generated from  
88037           that file is present in the output, where *line* is a line number and *pathname* is the pathname used  
88038           to open the file.

88039 **STDERR**

88040 The standard error shall be used only for diagnostic messages, except that if more than one  
 88041 *pathname* operand ending in *.c* or *.i* (or possibly other unspecified suffixes) is given, for each such  
 88042 file:

88043 "%s:\n", <*pathname*>

88044 may be written to allow identification of the diagnostic and warning messages with the  
 88045 appropriate input file. These messages, if written, shall precede the processing of each input file;  
 88046 they shall not be written to the standard error if they are written to the standard output, as  
 88047 described in the STDOUT section.

88048 This utility may produce warning messages about certain conditions that do not warrant  
 88049 returning an error (non-zero) exit value.

88050 **OUTPUT FILES**

88051 Object files or executable files or both are produced in unspecified formats. If the pathname of  
 88052 an object file or executable file to be created by *c17* resolves to an existing directory entry for a  
 88053 file that is not a regular file, it is unspecified whether *c17* shall attempt to create the file or shall  
 88054 issue a diagnostic and exit with a non-zero exit status.

88055 **EXTENDED DESCRIPTION**88056 **Standard Libraries**

88057 The *c17* utility shall recognize the following *-l* options for standard libraries:

88058 **-l c** This option shall make available all interfaces referenced in the System Interfaces  
 88059 volume of POSIX.1-2024, with the possible exception of those interfaces listed as  
 88060 residing in <*aio.h*>, <*arpa/inet.h*>, <*complex.h*>, <*fenv.h*>, <*math.h*>,  
 88061 <*mqueue.h*>, <*netdb.h*>, <*net/if.h*>, <*netinet/in.h*>, <*pthread.h*>, <*sched.h*>,  
 88062 <*semaphore.h*>, <*spawn.h*>, <*sys/socket.h*>, <*threads.h*>, *pthread\_kill()* and  
 88063 *pthread\_sigmask()* in <*signal.h*>, interfaces marked as optional in <*sys/mman.h*>,  
 88064 interfaces marked as ADV (Advisory Information) in <*fcntl.h*>, and interfaces  
 88065 beginning with the prefix *clock\_* or *timer\_* in <*time.h*>. This option shall not be  
 88066 required to be present to cause a search of this library.

88067 **-l l** This option shall make available all interfaces required by the C-language output  
 88068 of *lex* that are not made available through the *-l c* option.

88069 **-l pthread** This option shall make available all interfaces referenced in <*pthread.h*> and  
 88070 <*threads.h*>, and also *pthread\_kill()* and *pthread\_sigmask()* referenced in  
 88071 <*signal.h*>. An implementation may search this library in the absence of this  
 88072 option.

88073 **-l m** This option shall make available all interfaces referenced in <*math.h*>,  
 88074 <*complex.h*>, and <*fenv.h*>. An implementation may search this library in the  
 88075 absence of this option.

88076 **-l rt** This option shall make available all interfaces referenced in <*aio.h*>, <*mqueue.h*>,  
 88077 <*sched.h*>, <*semaphore.h*>, and <*spawn.h*>, interfaces marked as optional in  
 88078 <*sys/mman.h*>, interfaces marked as ADV (Advisory Information) in <*fcntl.h*>,  
 88079 and interfaces beginning with the prefix *clock\_* and *timer\_* in <*time.h*>. An  
 88080 implementation may search this library in the absence of this option.

88081 **-l xnet** This option shall make available all interfaces referenced in <*arpa/inet.h*>,  
 88082 <*netdb.h*>, <*net/if.h*>, <*netinet/in.h*>, and <*sys/socket.h*>. An implementation  
 88083 may search this library in the absence of this option.

88084 **-l y** This option shall make available all interfaces required by the C-language output  
88085 of *yacc* that are not made available through the **-l c** option.

88086 In the absence of options that inhibit invocation of the link editor, such as **-c** or **-E**, the *c17* utility  
88087 shall cause the equivalent of a **-l c** option to be passed to the link editor after the last *pathname*  
88088 operand or **-l** option, causing it to be searched after all other object files and libraries are loaded.

88089 The libraries **c**, **l**, **m**, **pthread**, **rt**, **xnet**, and **y** shall be found as shared libraries when specified as  
88090 the option-argument to the **-l** option and may also be found as static libraries but, except for the  
88091 shared library version of the **c** library, need not exist as regular files. The implementation may  
88092 accept as **-l** option-arguments names of additional implementation-defined libraries that do not  
88093 exist as regular files.

### 88094 External Symbols

88095 The C compiler and link editor shall support the significance of external symbols up to a length  
88096 of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-  
88097 defined maximum symbol length is unspecified.

88098 The compiler and link editor shall support a minimum of 4095 external identifiers in one  
88099 translation unit. A diagnostic message shall be written to the standard output if the  
88100 implementation-defined limit is exceeded; other actions are unspecified.

### 88101 Header Search

88102 If a file with the same name as one of the standard headers defined in XBD [Chapter 14](#) (on page  
88103 221), not provided as part of the implementation, is placed in any of the usual places that are  
88104 searched by default for headers, the results are unspecified.

### 88105 Programming Environments

88106 All implementations shall support one of the following programming environments as a default.  
88107 Implementations may support more than one of the following programming environments.  
88108 Applications can use the `_POSIX_Vn_ILP*` and `_POSIX_Vn_LP*` constants in `<unistd.h>`, and  
88109 corresponding `sysconf()` and `getconf` calls, to determine which programming environments are  
88110 supported.

88111 **Table 3-4** Programming Environments: Type Sizes

Programming Environment <i>getconf</i> Name	Bits in int	Bits in long	Bits in all pointer types	Bits in off_t
_POSIX_V8_ILP32_OFF32	32	32	32	32
_POSIX_V8_ILP32_OFFBIG	32	32	32	≥64
_POSIX_V8_LP64_OFF64	32	64	64	64
_POSIX_V8_LP64_OFFBIG	≥32	≥64	≥64	≥64

88118 All implementations shall support one or more environments where the widths of the following  
88119 types are no greater than the width of type **long**:

88120	<b>blksize_t</b>	<b>ptrdiff_t</b>	<b>tcflag_t</b>
88121	<b>cc_t</b>	<b>size_t</b>	<b>wchar_t</b>
88122	<b>mode_t</b>	<b>speed_t</b>	<b>wint_t</b>
88123	<b>nfds_t</b>	<b>ssize_t</b>	
88124	<b>pid_t</b>	<b>suseconds_t</b>	

88125 The executable files created when these environments are selected shall be in a proper format for  
 88126 execution by the *exec* family of functions. Each environment may be one of the ones in Table 3-4  
 88127 (on page 2675), or it may be another environment. The names for the environments that meet  
 88128 this requirement shall be output by a *getconf* command using the  
 88129 POSIX\_V8\_WIDTH\_RESTRICTED\_ENVS argument, as a <newline>-separated list of names  
 88130 suitable for use with the *getconf* -v option. If more than one environment meets the requirement,  
 88131 the names of all such environments shall be output on separate lines. Any of these names can  
 88132 then be used in a subsequent *getconf* command to obtain the flags specific to that environment  
 88133 with the following suffixes added as appropriate:

88134 `_CFLAGS` To get the C compiler flags.

88135 `_LDFLAGS` To get the linker/loader flags.

88136 `_LIBS` To get the libraries.

88137 This requirement may be removed in a future version.

88138 When this utility processes a file containing a function called *main()*, it shall be defined with a  
 88139 return type equivalent to **int**. Using return from the initial call to *main()* shall be equivalent  
 88140 (other than with respect to language scope issues) to calling *exit()* with the returned value.  
 88141 Reaching the end of the initial call to *main()* shall be equivalent to calling *exit(0)*. The  
 88142 implementation shall not declare a prototype for this function.

88143 Implementations provide configuration strings for C compiler flags, linker/loader flags, and  
 88144 libraries for each supported environment. When an application needs to use a specific  
 88145 programming environment rather than the implementation default programming environment  
 88146 while compiling, the application shall first verify that the implementation supports the desired  
 88147 environment. If the desired programming environment is supported, the application shall then  
 88148 invoke *c17* with the appropriate C compiler flags as the first options for the compile, the  
 88149 appropriate linker/loader flags after any other options except `-I` but before any operands or `-I`  
 88150 options, and the appropriate libraries at the end of the operands and `-I` options.

88151 Conforming applications shall not attempt to link together object files compiled for different  
 88152 programming models. Applications shall also be aware that binary data placed in shared  
 88153 memory or in files might not be recognized by applications built for other programming models.

88154 **Table 3-5** Programming Environments: *c17* Arguments

88155 88156	<b>Programming Environment</b> <i>getconf</i> Name	<b>Use</b>	<b><i>c17</i> Arguments</b> <i>getconf</i> Name
88157 88158 88159	<code>_POSIX_V8_ILP32_OFF32</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V8_ILP32_OFF32_CFLAGS</code> <code>POSIX_V8_ILP32_OFF32_LDFLAGS</code> <code>POSIX_V8_ILP32_OFF32_LIBS</code>
88160 88161 88162	<code>_POSIX_V8_ILP32_OFFBIG</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V8_ILP32_OFFBIG_CFLAGS</code> <code>POSIX_V8_ILP32_OFFBIG_LDFLAGS</code> <code>POSIX_V8_ILP32_OFFBIG_LIBS</code>
88163 88164 88165	<code>_POSIX_V8_LP64_OFF64</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V8_LP64_OFF64_CFLAGS</code> <code>POSIX_V8_LP64_OFF64_LDFLAGS</code> <code>POSIX_V8_LP64_OFF64_LIBS</code>
88166 88167 88168	<code>_POSIX_V8_LPBIG_OFFBIG</code>	C Compiler Flags Linker/Loader Flags Libraries	<code>POSIX_V8_LPBIG_OFFBIG_CFLAGS</code> <code>POSIX_V8_LPBIG_OFFBIG_LDFLAGS</code> <code>POSIX_V8_LPBIG_OFFBIG_LIBS</code>

88169 In addition to the type size programming environments above, all implementations also support  
 88170 a multi-threaded programming environment that is orthogonal to all of the programming  
 88171 environments listed above. The *getconf* utility can be used to get flags for the threaded  
 88172 programming environment, as indicated in Table 3-6.

88173 **Table 3-6** Threaded Programming Environment: *c17* Arguments

88174 88175	Programming Environment <i>getconf</i> Name	Use	<i>c17</i> Arguments <i>getconf</i> Name
88176 88177	_POSIX_THREADS	C Compiler Flags Linker/Loader Flags	POSIX_V8_THREADS_CFLAGS POSIX_V8_THREADS_LDFLAGS

88178 These programming environment flags may be used in conjunction with any of the type size  
 88179 programming environments supported by the implementation.

#### 88180 EXIT STATUS

88181 The following exit values shall be returned:

- 88182 0 Successful compilation or link edit.
- 88183 >0 An error occurred.

#### 88184 CONSEQUENCES OF ERRORS

88185 When *c17* encounters a compilation error that causes an object file not to be created, it shall write  
 88186 a diagnostic to standard error and continue to compile other source code operands, but it shall  
 88187 not perform the link phase and it shall return a non-zero exit status. If the link edit is  
 88188 unsuccessful, a diagnostic message shall be written to standard error and *c17* exits with a non-  
 88189 zero status. A conforming application shall rely on the exit status of *c17*, rather than on the  
 88190 existence or mode of the executable file.

#### 88191 APPLICATION USAGE

88192 Since the *c17* utility usually creates files in the current directory during the compilation process,  
 88193 it is typically necessary to run the *c17* utility in a directory in which a file can be created.

88194 On systems providing POSIX Conformance (see XBD Chapter 2, on page 15), *c17* is required  
 88195 only with the C-Language Development option; XSI-conformant systems always provide *c17*.

88196 Since this standard requires that conforming applications define either `_POSIX_C_SOURCE` or  
 88197 `_XOPEN_SOURCE` before inclusion of any header (see XSH Section 2.2.1, on page 496), if *c17* is  
 88198 used to compile source code that includes a header without defining one of these feature test  
 88199 macros in the required manner, the behavior of *c17* itself and the results of using any files it  
 88200 generates are undefined. When *c17* is used this way, implementations are encouraged to make  
 88201 visible in headers from the ISO C standard only the symbols that are allowed by that standard,  
 88202 and otherwise to behave the same as if `_POSIX_C_SOURCE` or `_XOPEN_SOURCE` had been  
 88203 defined, but portable applications cannot rely on this.

88204 Some historical implementations have created `.o` files when `-c` is not specified and more than  
 88205 one source file is given. Since this area is left unspecified, the application cannot rely on `.o` files  
 88206 being created, but it also must be prepared for any related `.o` files that already exist being deleted  
 88207 at the completion of the link edit.

88208 There is the possible implication that if a user supplies versions of the standard functions (before  
 88209 they would be encountered by an implicit `-I c` or explicit `-I m`), that those versions would be  
 88210 used in place of the standard versions. There are various reasons this might not be true  
 88211 (functions defined as macros, manipulations for clean name space, and so on), so the existence of  
 88212 files named in the same manner as the standard libraries within the `-L` directories is explicitly

88213 stated to produce unspecified behavior.

88214 All of the functions specified in the System Interfaces volume of POSIX.1-2024 may be made  
88215 visible by implementations when the Standard C Library is searched. Conforming applications  
88216 must explicitly request searching the other standard libraries when functions made visible by  
88217 those libraries are used.

88218 In the ISO C standard the mapping from physical source characters to the C source character set  
88219 is implementation-defined. Implementations may strip white-space characters before the  
88220 terminating <newline> of a (physical) line as part of this mapping and, as a consequence of this,  
88221 one or more white-space characters (and no other characters) between a <backslash> character  
88222 and the <newline> character that terminates the line produces implementation-defined results.  
88223 Portable applications should not use such constructs.

88224 Some *c17* compilers not conforming to POSIX.1-2024 do not support trigraphs by default.

88225 Implementations may support multiple programming environments with some of them  
88226 conforming to this standard and some not conforming. The `_POSIX_Vn_ILP*` and  
88227 `_POSIX_Vn_LP*` constants in `<unistd.h>`, and corresponding `sysconf()` and `getconf` calls, only  
88228 indicate whether each programming environment is supported; they do not indicate anything  
88229 about conformance of the environments that are supported. For example, an implementation  
88230 may support the ILP32\_OFF32 environment for legacy reasons with a 32-bit `time_t`, whereas in a  
88231 conforming environment `time_t` is required to have a width of at least 64 bits. Application  
88232 writers should consult an implementation's POSIX Conformance Document for information  
88233 about the conformance of each supported programming environment.

## 88234 EXAMPLES

88235 1. The following usage example compiles `foo.c` and creates the executable file `foo`:

```
88236 c17 -o foo foo.c
```

88237 The following usage example compiles `foo.c` and creates the object file `foo.o`:

```
88238 c17 -c foo.c
```

88239 The following usage example compiles `foo.c` and creates the executable file `a.out`:

```
88240 c17 foo.c
```

88241 The following usage example compiles `foo.c`, links it with `bar.o`, and creates the  
88242 executable file `a.out`. It may also create and leave `foo.o`:

```
88243 c17 foo.c bar.o
```

88244 The following usage example preprocesses `foo.c` and `bar.c` to create `foo.i` and `bar.i`,  
88245 compiles `foo.i` and `bar.i` to create `foo.o` and `bar.o`, then links `foo.o` and `bar.o` to create the  
88246 executable file `foobar`. Each *c17* execution would ideally be invoked from a separate rule  
88247 in a makefile (see *make*, on page 3130) with suitable dependencies so that each is only  
88248 executed when it needs to be:

```
88249 c17 -E foo.c > foo.i
88250 c17 -E bar.c > bar.i
88251 c17 -c foo.i
88252 c17 -c bar.i
88253 c17 -o foobar foo.o bar.o
```

88254 2. The following example shows how an application using threads interfaces can test for  
88255 support of and use a programming environment supporting 32-bit `int`, `long`, and all  
88256 pointer types, and an `off_t` type using at least 64 bits:

```

88257 offbig_env=$(getconf _POSIX_V8_ILP32_OFFBIG)
88258 if [ $offbig_env != "-1" ] && [ $offbig_env != "undefined" ]
88259 then
88260     c17 $(getconf POSIX_V8_ILP32_OFFBIG_CFLAGS) \
88261         $(getconf POSIX_V8_THREADS_CFLAGS) -D_XOPEN_SOURCE=800 \
88262         $(getconf POSIX_V8_ILP32_OFFBIG_LDFLAGS) \
88263         $(getconf POSIX_V8_THREADS_LDFLAGS) foo.c -o foo \
88264         $(getconf POSIX_V8_ILP32_OFFBIG_LIBS) \
88265         -l pthread
88266 else
88267     echo ILP32_OFFBIG programming environment not supported
88268     exit 1
88269 fi

```

3. The following examples clarify the use and interactions of `-L` and `-I` options.

Consider the case in which module `a.c` calls function `f()` in library `libQ.a`, and module `b.c` calls function `g()` in library `libp.a`. Assume that both libraries reside in `/a/b/c`. The command line to compile and link in the desired way is:

```
c17 -L /a/b/c main.o a.c -l Q b.c -l p
```

In this case the `-L` option need only precede the first `-I` option, since both `libQ.a` and `libp.a` reside in the same directory.

Multiple `-L` options can be used when library name collisions occur. Building on the previous example, suppose that the user wants to use a new `libp.a`, in `/a/a/a`, but still wants `f()` from `/a/b/c/libQ.a`:

```
c17 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

In this example, the linker searches the `-L` options in the order specified, and finds `/a/a/a/libp.a` before `/a/b/c/libp.a` when resolving references for `b.c`. The order of the `-I` options is still important, however.

4. The following example shows how an application can use a programming environment where the widths of the following types:

**blksize\_t, cc\_t, mode\_t, nfsd\_t, pid\_t, ptrdiff\_t, size\_t, speed\_t, ssize\_t, suseconds\_t, tflag\_t, wchar\_t, wint\_t**

are no greater than the width of type **long**:

```

88288 # First choose one of the listed environments ...
88289
88290 # ... if there are no additional constraints, the first one will do:
88291 CENV=$(getconf POSIX_V8_WIDTH_RESTRICTED_ENVS | head -n 1)
88292
88293 # ... or, if an environment that supports large files is preferred,
88294 # look for names that contain "OFF64" or "OFFBIG". (This chooses
88295 # the last one in the list if none match.)
88296 for CENV in $(getconf POSIX_V8_WIDTH_RESTRICTED_ENVS)
88297 do
88298     case $CENV in
88299         *OFF64*|*OFFBIG*) break ;;
88300     esac
88301 done

```

```

88301      # The chosen environment name can now be used like this:
88302      c17 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=202405L \
88303      $(getconf ${CENV}_LDFLAGS) foo.c -o foo \
88304      $(getconf ${CENV}_LIBS)

```

88305 5. The following example shows how to create a shared library that does not depend on any  
88306 other shared library:

```

88307      c17 -G -c foo.c bar.c
88308      c17 -G -o foobar.so foo.o bar.o

```

88309 6. The following example shows how to create a dynamic executable that loads application  
88310 specific shared libraries by searching a specified list of directories when it is executed:

```

88311      c17 -G -c foo.c
88312      c17 -G -o /path/to/dir1/foo.so foo.o
88313      c17 -G -c bar.c
88314      c17 -G -o /path/to/dir2/bar.so bar.o
88315      c17 -B dynamic -L /path/to/dir1 -L /path/to/dir2 -R /path/to/dir1 \
88316      -R /path/to/dir2 -o foobar foobar.c -l foo -l bar

```

## 88317 RATIONALE

88318 The *c17* utility is based on the *c89* utility originally introduced in the ISO POSIX-2:1993  
88319 standard.

88320 Some of the changes from *c89* include the ability to intersperse options and operands (which  
88321 many *c89* implementations allowed despite it not being specified), the description of *-l* as an  
88322 option instead of an operand, and the modification to the contents of the Standard Libraries  
88323 section to account for new headers and options; for example, **<spawn.h>** added to the  
88324 description of *-l rt*.

88325 POSIX.1-2024 specifies that the *c17* utility must be able to use regular files for *\*.o* files and for  
88326 **a.out** files. Implementations are free to overwrite existing files of other types when attempting to  
88327 create object files and executable files, but are not required to do so. If something other than a  
88328 regular file is specified and using it fails for any reason, *c17* is required to issue a diagnostic  
88329 message and exit with a non-zero exit status. But for some file types, the problem may not be  
88330 noticed for a long time. For example, if a FIFO named **a.out** exists in the current directory, *c17*  
88331 may attempt to open **a.out** and will hang in the *open()* call until another process opens the FIFO  
88332 for reading. Then *c17* may write most of the **a.out** to the FIFO and fail when it tries to seek back  
88333 close to the start of the file to insert a timestamp (FIFOs are not seekable files). The *c17* utility is  
88334 also allowed to issue a diagnostic immediately if it encounters an **a.out** or *\*.o* file that is not a  
88335 regular file. For portable use, applications should ensure that any **a.out**, *-o* option-argument, or  
88336 *\*.o* files corresponding to any *\*.c* files do not conflict with names already in use that are not  
88337 regular files or symbolic links that point to regular files.

88338 Commands of the form *c17 -c -o . . .* are frequently used to directly place the *.o* file into an  
88339 alternative directory without a need to separately rename the output file. This helps to support  
88340 concurrent compilations and out of tree builds.

88341 Some implementations allow *-c -o directory* to produce *directory/file.o* even when there is more  
88342 than one input file; however, portable applications using *-c* with *-o* must compile only one file  
88343 at a time and must specify the final destination filename rather than a directory.

88344 The shared library version of the *c* library is required to exist as a regular file because the  
88345 dynamic linker needs to be able to load at least one library at execution time. Other standard  
88346 shared libraries need not exist in their own right if the interfaces the standard requires them to



88347 provide exist in the `c` library; all that is required is that they are “found” when specified as `-l`  
88348 option-arguments. Static versions of the standard libraries need not exist as regular files, even if  
88349 they are found as static libraries when specified as `-l` option-arguments.

88350 On many systems, multi-threaded applications run in a programming environment that is  
88351 distinct from that used by single-threaded applications. This multi-threaded programming  
88352 environment (in addition to needing to specify `-l pthread` at link time) may require additional  
88353 flags to be set when headers are processed at compile time (`-D_REENTRANT` being common).  
88354 This programming environment is orthogonal to the type size programming environments  
88355 discussed above and listed in Table 3-4 (on page 2675). This version of the standard adds *getconf*  
88356 utility calls to provide the C compiler flags and linker/loader flags needed to support multi-  
88357 threaded applications. Note that on a system where single-threaded applications are a special  
88358 case of a multi-threaded application, both of these *getconf* calls may return NULL strings; on  
88359 other implementations both of these strings may be non-NULL strings.

88360 The C standardization committee invented trigraphs (e.g., “??!” to represent ‘|’) to address  
88361 character portability problems in development environments based on national variants of the  
88362 7-bit ISO/IEC 646:1991 standard character set. However, these environments were already  
88363 obsolete by the time the first ISO C standard was published, and in practice trigraphs have not  
88364 been used for their intended purpose, and usually are intended to have their original meaning in  
88365 K&R C. For example, in practice a C-language source string like “What??!” is usually intended  
88366 to end in two <question-mark> characters and an <exclamation-mark>, not in ‘|’.

88367 When the `-E` option is used, execution of some `#pragma` preprocessor directives may simply  
88368 result in a copy of the directive being included in the output as part of the allowed extra  
88369 information used by subsequent compilation passes (see `STDOUT`).

88370 This standard requires that, when `-E` is used, lines beginning with ‘#’ are output identifying  
88371 the files processed as a result of `#include` directives in order that `c17 -E` can be used to generate  
88372 makefile dependencies. See *make*. Usually such lines are output each time the origin of the  
88373 subsequent lines in the output changes, and the line number after the ‘#’ is the line number in  
88374 the named file of the line from which the next line in the output was derived.

88375 When the `-R` option is not included when an executable file or shared library is being created,  
88376 some implementations use the environment variables `LD_RUN_PATH` and `LD_LIBRARY_PATH`  
88377 to determine the directories to be searched for shared libraries.

88378 Some implementations permit place-holders preceded by a <dollar-sign> character (‘\$’), such  
88379 as `$ORIGIN`, in the `-R` directory option-argument to be evaluated at load time. Some  
88380 implementations accept a colon separated list of directories for the path to search for shared  
88381 libraries, with the same effect as specifying the `-R` option multiple times. However, these  
88382 features are not universal.

88383 The name of a shared library usually contains an element named `so`. Other implementation-  
88384 defined elements are allowed for backwards compatibility with historical systems, and so that  
88385 tools can be developed on conforming systems to create libraries for multiple environments. For  
88386 example, Microsoft systems use the filename extension `.dll` (and do not allow following text) to  
88387 denote a shared library. The standard allows additional characters to be used in the name of a  
88388 library following an `so` element to permit shared library versioning information to be at the end  
88389 of the library filename rather than requiring that any such strings appear before the final  
88390 element of the library name.

88391 The decision to standardize on `so` as a required element in a shared library name was  
88392 intentional, as the alternative would have been standardizing things such as a new make macro  
88393 `$(SHLIB_EXT)` that would otherwise be needed to write a portable makefile that can compile  
88394 shared libraries despite not having a standardized element name.

88395 If a combination of direct and indirect dependencies of a shared library would require different  
88396 versions of another shared library, options that are not specified by the standard (such as **-B**  
88397 **direct**) will probably need to be used when linking that shared library, so that at runtime the  
88398 intended versions are found.

#### 88399 **FUTURE DIRECTIONS**

88400 If this utility is directed to display a pathname that contains any bytes that have the encoded  
88401 value of a <newline> character when <newline> is a terminator or separator in the output  
88402 format being used, implementations are encouraged to treat this as an error. A future version of  
88403 this standard may require implementations to treat this as an error.

88404 If this utility is directed to create a new directory entry that contains any bytes that have the  
88405 encoded value of a <newline> character, implementations are encouraged to treat this as an  
88406 error. A future version of this standard may require implementations to treat this as an error.

88407 Unlike all of the other non-OB-shaded utilities in this standard, a utility by this name probably  
88408 will not appear in the next version of this standard. This utility's name is tied to the current  
88409 revision of the ISO C standard at the time this standard is approved. Since the ISO C standard  
88410 and this standard are maintained by different organizations on different schedules, we cannot  
88411 predict what the compiler will be named in the next version of the standard.

#### 88412 **SEE ALSO**

88413 [Section 1.1.1.4](#) (on page 2454), *ar*, *getconf*, *make*, *nm*, *strip*, *umask*

88414 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215), [Chapter 14](#) (on page 221)

88415 XSH *exec*, *sysconf()*

#### 88416 **CHANGE HISTORY**

88417 First released in Issue 8. Included for alignment with the ISO/IEC 9899:2018 standard.

88418 **NAME**

88419 cal — print a calendar

88420 **SYNOPSIS**88421 XSI cal **[[*month*] *year*]**88422 **DESCRIPTION**

88423 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from  
 88424 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,  
 88425 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on  
 88426 September 14, 1752.

88427 If no operands are given, *cal* shall produce a one-month calendar for the current month in the  
 88428 current year. If only the *year* operand is given, *cal* shall produce a calendar for all twelve months  
 88429 in the given calendar year. If both *month* and *year* operands are given, *cal* shall produce a one-  
 88430 month calendar for the given month in the given year.

88431 **OPTIONS**

88432 None.

88433 **OPERANDS**

88434 The following operands shall be supported:

88435 *month* Specify the month to be displayed, represented as a decimal integer from 1  
 88436 (January) to 12 (December).

88437 *year* Specify the year for which the calendar is displayed, represented as a decimal  
 88438 integer from 1 to 9999.

88439 **STDIN**

88440 Not used.

88441 **INPUT FILES**

88442 None.

88443 **ENVIRONMENT VARIABLES**88444 The following environment variables shall affect the execution of *cal*:

88445 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 88446 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 88447 variables used to determine the values of locale categories.)

88448 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 88449 internationalization variables.

88450 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 88451 characters (for example, single-byte as opposed to multi-byte characters in  
 88452 arguments).

88453 *LC\_MESSAGES*

88454 Determine the locale that should be used to affect the format and contents of  
 88455 diagnostic messages written to standard error, and informative messages written  
 88456 to standard output.

88457 *LC\_TIME* Determine the format and contents of the calendar.

88458 *NLSPATH* Determine the location of messages objects and message catalogs.

- 88459 `TZ` Determine the timezone used to calculate the value of the current month.
- 88460 **ASYNCHRONOUS EVENTS**
- 88461 Default.
- 88462 **STDOUT**
- 88463 The standard output shall be used to display the calendar, in an unspecified format.
- 88464 **STDERR**
- 88465 The standard error shall be used only for diagnostic messages.
- 88466 **OUTPUT FILES**
- 88467 None.
- 88468 **EXTENDED DESCRIPTION**
- 88469 None.
- 88470 **EXIT STATUS**
- 88471 The following exit values shall be returned:
- 88472 `0` Successful completion.
- 88473 `>0` An error occurred.
- 88474 **CONSEQUENCES OF ERRORS**
- 88475 Default.
- 88476 **APPLICATION USAGE**
- 88477 Note that:
- 88478 `cal 83`
- 88479 refers to A.D. 83, not 1983.
- 88480 **EXAMPLES**
- 88481 None.
- 88482 **RATIONALE**
- 88483 Earlier versions of this standard incorrectly required that the command:
- 88484 `cal 2000`
- 88485 write a one-month calendar for the current calendar month (no matter what the current year is)
- 88486 in the year 2000 to standard output. This did not match historic practice in any known version of
- 88487 the `cal` utility. The description has been updated to match historic practice. When only the *year*
- 88488 operand is given, `cal` writes a twelve-month calendar for the specified year.
- 88489 **FUTURE DIRECTIONS**
- 88490 A future version of this standard may support locale-specific recognition of the date of adoption
- 88491 of the Gregorian calendar.
- 88492 **SEE ALSO**
- 88493 XBD [Chapter 8](#) (on page 167)
- 88494 **CHANGE HISTORY**
- 88495 First released in Issue 2.
- 88496 **Issue 6**
- 88497 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of
- 88498 the Gregorian calendar.

88499 **Issue 7**

88500 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88501 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0074 [56] and XCU/TC1-2008/0075  
88502 [56] are applied.

88503 **Issue 8**

88504 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

88505 **NAME**

88506 cat — concatenate and print files

88507 **SYNOPSIS**88508 cat [-u] [*file*...]88509 **DESCRIPTION**88510 The *cat* utility shall read files in sequence and shall write their contents to the standard output in  
88511 the same sequence.88512 **OPTIONS**88513 The *cat* utility shall conform to XBD [Section 12.2](#) (on page 215).

88514 The following option shall be supported:

88515 **-u** Write bytes from the input file to the standard output without delay as each is  
88516 read.88517 **OPERANDS**

88518 The following operand shall be supported:

88519 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
88520 shall be used. If a *file* is '-', the *cat* utility shall read from the standard input at  
88521 that point in the sequence. The *cat* utility shall not close and reopen standard input  
88522 when it is referenced in this way, but shall accept multiple occurrences of '-' as a  
88523 *file* operand.88524 **STDIN**88525 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
88526 See the INPUT FILES section.88527 **INPUT FILES**

88528 The input files can be any file type.

88529 **ENVIRONMENT VARIABLES**88530 The following environment variables shall affect the execution of *cat*:88531 **LANG** Provide a default value for the internationalization variables that are unset or null.  
88532 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
88533 variables used to determine the values of locale categories.)88534 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
88535 internationalization variables.88536 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
88537 characters (for example, single-byte as opposed to multi-byte characters in  
88538 arguments).88539 **LC\_MESSAGES**88540 Determine the locale that should be used to affect the format and contents of  
88541 diagnostic messages written to standard error.88542 **XSI** **NLSPATH** Determine the location of messages objects and message catalogs.88543 **ASYNCHRONOUS EVENTS**

88544 Default.

**88545 STDOUT**

88546 The standard output shall contain the sequence of bytes read from the input files. Nothing else  
88547 shall be written to the standard output. If the standard output is a regular file, and is the same  
88548 file as any of the input file operands, the implementation may treat this as an error.

**88549 STDERR**

88550 The standard error shall be used only for diagnostic messages.

**88551 OUTPUT FILES**

88552 None.

**88553 EXTENDED DESCRIPTION**

88554 None.

**88555 EXIT STATUS**

88556 The following exit values shall be returned:

88557 0 All input files were output successfully.

88558 >0 An error occurred.

**88559 CONSEQUENCES OF ERRORS**

88560 Default.

**88561 APPLICATION USAGE**

88562 The `-u` option has value in prototyping non-blocking reads from FIFOs. The intent is to support  
88563 the following sequence:

```
88564 mkfifo foo  
88565 cat -u foo > /dev/tty13 &  
88566 cat -u > foo
```

88567 It is unspecified whether standard output is or is not buffered in the default case. This is  
88568 sometimes of interest when standard output is associated with a terminal, since buffering may  
88569 delay the output. The presence of the `-u` option guarantees that unbuffered I/O is available. It is  
88570 implementation-defined whether the `cat` utility buffers output if the `-u` option is not specified.  
88571 Traditionally, the `-u` option is implemented using the equivalent of the `setvbuf()` function  
88572 defined in the System Interfaces volume of POSIX.1-2024.

**88573 EXAMPLES**

88574 The following command:

```
88575 cat myfile
```

88576 writes the contents of the file **myfile** to standard output.

88577 The following command:

```
88578 cat doc1 doc2 > doc.all
```

88579 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

88580 Because of the shell language mechanism used to perform output redirection, a command such  
88581 as this:

```
88582 cat doc doc.end > doc
```

88583 causes the original data in **doc** to be lost before `cat` even begins execution. This is true whether  
88584 the `cat` command fails with an error or silently succeeds (the specification allows both  
88585 behaviors). In order to append the contents of **doc.end** without losing the original contents of  
88586 **doc**, this command should be used instead:

88587 `cat doc.end >> doc`

88588 The command:

88589 `cat start - middle - end > file`

88590 when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a  
88591 single invocation of *cat*. Note, however, that if standard input is a regular file, this would be  
88592 equivalent to the command:

88593 `cat start - middle /dev/null end > file`

88594 because the entire contents of the file would be consumed by *cat* the first time '-' was used as a  
88595 *file* operand and an end-of-file condition would be detected immediately when '-' was  
88596 referenced the second time.

### 88597 RATIONALE

88598 Historical versions of the *cat* utility include the `-e`, `-t`, and `-v`, options which permit the ends of  
88599 lines, `<tab>` characters, and invisible characters, respectively, to be rendered visible in the  
88600 output. The standard developers omitted these options because they provide too fine a degree of  
88601 control over what is made visible, and similar output can be obtained using a command such as:

88602 `sed -n l pathname`

88603 The latter also has the advantage that its output is unambiguous, whereas the output of  
88604 historical *cat -etv* is not.

88605 The `-s` option was omitted because it corresponds to different functions in BSD and System  
88606 V-based systems. The BSD `-s` option to squeeze blank lines can be accomplished by the shell  
88607 script shown in the following example:

```
88608 sed -n '  
88609 # Write non-empty lines.  
88610 ./ {  
88611     p  
88612     d  
88613 }  
88614 # Write a single empty line, then look for more empty lines.  
88615 /^$/ p  
88616 # Get next line, discard the held <newline> (empty line),  
88617 # and look for more empty lines.  
88618 :Empty  
88619 /^$/ {  
88620     N  
88621     s/././  
88622     b Empty  
88623 }  
88624 # Write the non-empty line before going back to search  
88625 # for the first in a set of empty lines.  
88626     p  
88627 '
```

88628 The System V `-s` option to silence error messages can be accomplished by redirecting the  
88629 standard error. Note that the BSD documentation for *cat* uses the term ``blank line'' to mean the  
88630 same as the POSIX ``empty line'': a line consisting only of a `<newline>`.

88631 The BSD `-n` option was omitted because similar functionality can be obtained from the `-n`  
88632 option of the *pr* utility.



88633 **FUTURE DIRECTIONS**

88634 None.

88635 **SEE ALSO**88636 *more*88637 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)88638 XSH *setvbuf()*88639 **CHANGE HISTORY**

88640 First released in Issue 2.

88641 **Issue 7**

88642 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88643 SD5-XCU-ERN-174 is applied, changing the RATIONALE.

88644 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0073 [876] is applied.

88645 **Issue 8**88646 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

88647 **NAME**

88648 cd — change the working directory

88649 **SYNOPSIS**88650 cd [-L] [*directory*]88651 cd -P [-e] [*directory*]88652 **DESCRIPTION**

88653 The *cd* utility shall change the working directory of the current shell execution environment (see  
88654 [Section 2.13](#), on page 2522) by executing the following steps in sequence. (In the following steps,  
88655 the symbol **curpath** represents an intermediate value used to simplify the description of the  
88656 algorithm used by *cd*. There is no requirement that **curpath** be made visible to the application.)

- 88657 1. If no *directory* operand is given and the *HOME* environment variable is empty or  
88658 undefined, the default behavior is implementation-defined and no further steps shall be  
88659 taken.
- 88660 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty  
88661 value, the *cd* utility shall behave as if the directory named in the *HOME* environment  
88662 variable was specified as the *directory* operand.
- 88663 3. If the *directory* operand begins with a <slash> character, set **curpath** to the operand and  
88664 proceed to step 7.
- 88665 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
- 88666 5. Starting with the first pathname in the <colon>-separated pathnames of *CDPATH* (see the  
88667 ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the  
88668 concatenation of that pathname, a <slash> character if that pathname did not end with a  
88669 <slash> character, and the *directory* operand names a directory. If the pathname is null,  
88670 test if the concatenation of dot, a <slash> character, and the operand names a directory. In  
88671 either case, if the resulting string names an existing directory, set **curpath** to that string  
88672 and proceed to step 7. Otherwise, repeat this step with the next pathname in *CDPATH*  
88673 until all pathnames have been tested.
- 88674 6. Set **curpath** to the *directory* operand.
- 88675 7. If the *-P* option is in effect, proceed to step 10. If **curpath** does not begin with a <slash>  
88676 character, set **curpath** to the string formed by the concatenation of the value of *PWD*, a  
88677 <slash> character if the value of *PWD* did not end with a <slash> character, and **curpath**.
- 88678 8. The **curpath** value shall then be converted to canonical form as follows, considering each  
88679 component from beginning to end, in sequence:
  - 88680 a. Dot components and any <slash> characters that separate them from the next  
88681 component shall be deleted.
  - 88682 b. For each dot-dot component, if there is a preceding component and it is neither  
88683 root nor dot-dot, then:
    - 88684 i. If the preceding component does not refer (in the context of pathname  
88685 resolution with symbolic links followed) to a directory, then the *cd* utility  
88686 shall display an appropriate error message and no further steps shall be  
88687 taken.
    - 88688 ii. The preceding component, all <slash> characters separating the preceding  
88689 component from dot-dot, dot-dot, and all <slash> characters separating dot-  
88690 dot from the following component (if any) shall be deleted.

88691 c. An implementation may further simplify **curpath** by removing any trailing  
 88692 <slash> characters that are not also leading <slash> characters, replacing multiple  
 88693 non-leading consecutive <slash> characters with a single <slash>, and replacing  
 88694 three or more leading <slash> characters with a single <slash>. If, as a result of  
 88695 this canonicalization, the **curpath** variable is null, no further steps shall be taken.

88696 9. If **curpath** is longer than {PATH\_MAX} bytes (including the terminating null) and the  
 88697 *directory* operand was not longer than {PATH\_MAX} bytes (including the terminating  
 88698 null), then **curpath** shall be converted from an absolute pathname to an equivalent  
 88699 relative pathname if possible. This conversion shall always be considered possible if the  
 88700 value of *PWD*, with a trailing <slash> added if it does not already have one, is an initial  
 88701 substring of **curpath**. Whether or not it is considered possible under other circumstances  
 88702 is unspecified. Implementations may also apply this conversion if **curpath** is not longer  
 88703 than {PATH\_MAX} bytes or the *directory* operand was longer than {PATH\_MAX} bytes.

88704 10. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with  
 88705 **curpath** as the *path* argument. If these actions fail for any reason, the *cd* utility shall  
 88706 display an appropriate error message and the remainder of this step shall not be  
 88707 executed. If the **-P** option is not in effect, the *PWD* environment variable shall be set to  
 88708 the value that **curpath** had on entry to step 9 (i.e., before conversion to a relative  
 88709 pathname).

88710 If the **-P** option is in effect, the *PWD* environment variable shall be set to the string that  
 88711 would be output by *pwd -P*. If there is insufficient permission on the new directory, or on  
 88712 any parent of that directory, to determine the current working directory, the value of the  
 88713 *PWD* environment variable is unspecified. If both the **-e** and the **-P** options are in effect  
 88714 and *cd* is unable to determine the pathname of the current working directory, *cd* shall  
 88715 complete successfully but return a non-zero exit status.

88716 If, during the execution of the above steps, the *PWD* environment variable is set, the *OLDPWD*  
 88717 shell variable shall also be set to the value of the old working directory (that is the current  
 88718 working directory immediately prior to the call to *cd*). It is unspecified whether, when setting  
 88719 *OLDPWD*, the shell also causes it to be exported if it was not already.

## 88720 OPTIONS

88721 The *cd* utility shall conform to XBD [Section 12.2](#) (on page 215).

88722 The following options shall be supported by the implementation:

88723 **-e** If the **-P** option is in effect, the current working directory is successfully changed,  
 88724 and the correct value of the *PWD* environment variable cannot be determined, exit  
 88725 with exit status 1.

88726 **-L** Handle the operand dot-dot logically; symbolic link components shall not be  
 88727 resolved before dot-dot components are processed (see steps 8. and 9. in the  
 88728 DESCRIPTION).

88729 **-P** Handle the operand dot-dot physically; symbolic link components shall be  
 88730 resolved before dot-dot components are processed (see step 7. in the  
 88731 DESCRIPTION).

88732 If both **-L** and **-P** options are specified, the last of these options shall be used and all others  
 88733 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the  
 88734 DESCRIPTION.

88735 **OPERANDS**

88736 The following operands shall be supported:

88737 *directory* An absolute or relative pathname of the directory that shall become the new  
 88738 working directory. The interpretation of a relative pathname by *cd* depends on the  
 88739 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an  
 88740 empty string, *cd* shall write a diagnostic message to standard error and exit with  
 88741 non-zero status. If *directory* consists of a single '-' (<hyphen-minus>) character,  
 88742 the *cd* utility shall behave as if *directory* contained the value of the *OLDPWD*  
 88743 environment variable, except that after it sets the value of *PWD* it shall write the  
 88744 new value to standard output. The behavior is unspecified if *OLDPWD* does not  
 88745 start with a <slash> character.

88746 **STDIN**

88747 Not used.

88748 **INPUT FILES**

88749 None.

88750 **ENVIRONMENT VARIABLES**88751 The following environment variables shall affect the execution of *cd*:

88752 *CDPATH* A <colon>-separated list of pathnames that refer to directories. The *cd* utility shall  
 88753 use this list in its attempt to change the directory, as described in the  
 88754 DESCRIPTION. An empty string in place of a directory pathname represents the  
 88755 current directory. If *CDPATH* is not set, it shall be treated as if it were an empty  
 88756 string.

88757 *HOME* The name of the directory, used when no *directory* operand is specified.

88758 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 88759 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 88760 variables used to determine the values of locale categories.)

88761 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 88762 internationalization variables.

88763 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 88764 characters (for example, single-byte as opposed to multi-byte characters in  
 88765 arguments).

88766 *LC\_MESSAGES*

88767 Determine the locale that should be used to affect the format and contents of  
 88768 diagnostic messages written to standard error.

88769 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

88770 *OLDPWD* A pathname of the previous working directory, used when the operand is '-'. If  
 88771 an application sets or unsets the value of *OLDPWD*, the behavior of *cd* with a '-'  
 88772 operand is unspecified.

88773 *PWD* This variable shall be set as specified in the DESCRIPTION. If an application sets  
 88774 or unsets the value of *PWD*, the behavior of *cd* is unspecified.

88775 **ASYNCHRONOUS EVENTS**

88776 Default.

88777 **STDOUT**

88778 If a non-empty directory name from *CDPATH* is used, or if the operand '-' is used, and the  
 88779 absolute pathname of the new working directory can be determined, that pathname shall be  
 88780 written to the standard output as follows:

88781 "%s\n", <new directory>

88782 If an absolute pathname of the new current working directory cannot be determined, it is  
 88783 unspecified whether nothing is written to the standard output or the value of **curpath** used in  
 88784 step 10, followed by a <newline>, is written to the standard output.

88785 If a non-empty directory name from *CDPATH* is not used, and the directory argument is not  
 88786 '-', there shall be no output.

88787 **STDERR**

88788 The standard error shall be used only for diagnostic messages.

88789 **OUTPUT FILES**

88790 None.

88791 **EXTENDED DESCRIPTION**

88792 None.

88793 **EXIT STATUS**

88794 The following exit values shall be returned:

88795 0 The current working directory was successfully changed and the value of the *PWD*  
 88796 environment variable was set correctly.

88797 0 The current working directory was successfully changed, the **-e** option is not in effect, the  
 88798 **-P** option is in effect, and the correct value of the *PWD* environment variable could not be  
 88799 determined.

88800 >0 Either the **-e** option or the **-P** option is not in effect, and an error occurred.

88801 1 The current working directory was successfully changed, both the **-e** and the **-P** options are  
 88802 in effect, and the correct value of the *PWD* environment variable could not be determined.

88803 >1 Both the **-e** and the **-P** options are in effect, and an error occurred.

88804 **CONSEQUENCES OF ERRORS**

88805 The working directory shall remain unchanged.

88806 **APPLICATION USAGE**

88807 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

88808 Since *cd* affects the current shell execution environment, it is always provided as a shell regular  
 88809 built-in. If it is called in a subshell or separate utility execution environment, such as one of the  
 88810 following:

```
88811 (cd /tmp)
88812 nohup cd
88813 find . -exec cd {} \;
```

88814 it does not affect the working directory of the caller's environment.

88815 The user must have execute (search) permission in *directory* in order to change to it.

88816 Since *cd* treats the operand '-' as a special case, applications should not pass arbitrary values as  
 88817 the operand. For example, instead of:

```
88818 CDPATH= cd -P -- "$dir"
```

88819 applications should use the following:

```
88820 case $dir in
88821  (/*) cd -P "$dir";;
88822  ("") echo >&2 directory is an empty string; exit 1;;
88823  (*) CDPATH= cd -P "$dir";;
88824 esac
```

88825 If an absolute pathname of the new current working directory cannot be determined, and a non-  
88826 empty directory name from *CDPATH* is used, *cd* may write a pathname to standard output that  
88827 is not an absolute pathname.

#### 88828 EXAMPLES

88829 The following template can be used to perform processing in the directory specified by *location*  
88830 and end up in the current working directory in use before the first *cd* command was issued:

```
88831 cd location
88832 if [ $? -ne 0 ]
88833 then
88834     print error message
88835     exit 1
88836 fi
88837 ... do whatever is desired as long as the OLDPWD environment variable
88838     is not modified
88839 cd -
```

#### 88840 RATIONALE

88841 The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of  
88842 the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

88843 A common extension when *HOME* is undefined is to get the login directory from the user  
88844 database for the invoking user. This does not occur on System V implementations.

88845 Some historical shells, such as the KornShell, took special actions when the directory name  
88846 contained a dot-dot component, selecting the logical parent of the directory, rather than the  
88847 actual parent directory; that is, it moved up one level toward the '/' in the pathname,  
88848 remembering what the user typed, rather than performing the equivalent of:

```
88849 chdir("../");
```

88850 In such a shell, the following commands would not necessarily produce equivalent output for all  
88851 directories:

```
88852 cd .. && ls      ls ..
```

88853 This behavior is now the default. It is not consistent with the definition of dot-dot in most  
88854 historical practice; that is, while this behavior has been optionally available in the KornShell,  
88855 other shells have historically not supported this functionality. The logical pathname is stored in  
88856 the *PWD* environment variable when the *cd* utility completes and this value is used to construct  
88857 the next directory name if *cd* is invoked with the *-L* option.

88858 When the *-P* option is in effect, the correct value of the *PWD* environment variable cannot be  
88859 determined on some systems, but still results in a zero exit status. The value of *PWD* doesn't  
88860 matter to some shell scripts and in those cases this is not a problem. In other cases, especially  
88861 with multiple calls to *cd*, the values of *PWD* and *OLDPWD* are important but the standard  
88862 provided no easy way to know that this was the case. The *-e* option has been added, even  
88863 though this was not historic practice, to give script writers a reliable way to know when the  
88864 value of *PWD* is not reliable.

88865 **FUTURE DIRECTIONS**

88866 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 88867 value of a <newline> character when <newline> is a terminator or separator in the output  
 88868 format being used, implementations are encouraged to treat this as an error. A future version of  
 88869 this standard may require implementations to treat this as an error.

88870 **SEE ALSO**

88871 [Section 2.13](#) (on page 2522), *pwd*  
 88872 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)  
 88873 XSH *chdir()*

88874 **CHANGE HISTORY**

88875 First released in Issue 2.

88876 **Issue 6**

88877 The following new requirements on POSIX implementations derive from alignment with the  
 88878 Single UNIX Specification:

- 88879 • The *cd* – operand, *PWD*, and *OLDPWD* are added.

88880 The *-L* and *-P* options are added to align with the IEEE P1003.2b draft standard. This also  
 88881 includes the introduction of a new description to include the effect of these options.

88882 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to  
 88883 make it clear that the *-L* and *-P* options are mutually-exclusive.

88884 **Issue 7**

88885 Austin Group Interpretation 1003.1-2001 #037 is applied.

88886 Austin Group Interpretation 1003.1-2001 #199 is applied, clarifying how the *cd* utility handles  
 88887 concatenation of two pathnames when the first pathname ends in a <slash> character.

88888 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88889 Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of “the operand”.

88890 Changes to the *pwd* utility and *PWD* environment variable have been made to match the  
 88891 changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.

88892 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0076 [230], XCU/TC1-2008/0077  
 88893 [240], XCU/TC1-2008/0078 [240], and XCU/TC1-2008/0079 [123] are applied.

88894 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0074 [584] is applied.

88895 **Issue 8**

88896 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
 88897 directed to display a pathname that contains any bytes that have the encoded value of a  
 88898 <newline> character when <newline> is a terminator or separator in the output format being  
 88899 used.

88900 Austin Group Defect 253 is applied, adding the *-e* option.

88901 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
 88902 this utility is required to be intrinsic.

88903 Austin Group Defect 1045 is applied, clarifying the behavior when the *directory* operand is ' - '.

88904 Austin Group Defect 1047 is applied, requiring *cd* to treat an empty *directory* operand as an error

88905 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

88906  
88907

Austin Group Defect 1527 is applied, clarifying the behavior when an absolute pathname of the new current working directory cannot be determined.

88908  
88909

Austin Group Defect 1601 is applied, clarifying that when setting *OLDPWD*, the shell need not cause it to be exported if it was not already.



88910 **NAME**88911 cflow — generate a C-language flowgraph (**DEVELOPMENT**)88912 **SYNOPSIS**

```
88913 XSI cflow [-r] [-d num] [-D name[=def]]... [-i incl] [-I dir]...
88914 [-U dir]... file...
```

88915 **DESCRIPTION**

88916 The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex*, or *yacc*  
 88917 source files, and attempt to build a graph, written to standard output, charting the external  
 88918 references.

88919 **OPTIONS**

88920 The *cflow* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the **-D**,  
 88921 **-I**, and **-U** options (which are identical to their interpretation by *c17*) is significant.

88922 The following options shall be supported:

- 88923 **-d num** Indicate the depth at which the flowgraph is cut off. The application shall ensure  
 88924 that the argument *num* is a decimal integer. By default this is a very large number  
 88925 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive  
 88926 integer shall be ignored.
- 88927 **-i incl** Increase the number of included symbols. The *incl* option-argument is one of the  
 88928 following characters:
- 88929 *x* Include external and static data symbols. The default shall be to include only  
 88930 functions in the flowgraph.
- 88931 *\_* (Underscore) Include names that begin with an <underscore>. The default  
 88932 shall be to exclude these functions (and data if **-i x** is used).
- 88933 **-r** Reverse the caller: callee relationship, producing an inverted listing showing the  
 88934 callers of each function. The listing shall also be sorted in lexicographical order by  
 88935 callee.

88936 **OPERANDS**

88937 The following operand is supported:

- 88938 *file* The pathname of a file for which a graph is to be generated. Filenames suffixed by  
 88939 **.I** shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c17* input, and **.i** as the  
 88940 output of *c17* **-E**. Such files shall be processed as appropriate, determined by their  
 88941 suffix.
- 88942 Files suffixed by **.s** (conventionally assembler source) may have more limited  
 88943 information extracted from them.

88944 **STDIN**

88945 Not used.

88946 **INPUT FILES**

88947 The input files shall be object files or assembler, C-language, *lex*, or *yacc* source files.

88948 **ENVIRONMENT VARIABLES**

88949 The following environment variables shall affect the execution of *cflow*:

- 88950 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 88951 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 88952 variables used to determine the values of locale categories.)

- 88953 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
88954 internationalization variables.
- 88955 *LC\_COLLATE*  
88956 Determine the locale for the ordering of the output when the *-r* option is used.
- 88957 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
88958 characters (for example, single-byte as opposed to multi-byte characters in  
88959 arguments and input files).
- 88960 *LC\_MESSAGES*  
88961 Determine the locale that should be used to affect the format and contents of  
88962 diagnostic messages written to standard error.
- 88963 *NLSPATH* Determine the location of messages objects and message catalogs.
- 88964 **ASYNCHRONOUS EVENTS**  
88965 Default.
- 88966 **STDOUT**  
88967 The flowgraph written to standard output shall be formatted as follows:  
88968 "%d %s:%s\n", <reference number>, <global>, <definition>
- 88969 Each line of output begins with a reference (that is, line) number, followed by indentation of at  
88970 least one column position per level. This is followed by the name of the global, a <colon>, and  
88971 its definition. Normally globals are only functions not defined as an external or beginning with  
88972 an <underscore>; see the **OPTIONS** section for the *-i* inclusion option. For information extracted  
88973 from C-language source, the definition consists of an abstract type declaration (for example, **char**  
88974 \*) and, delimited by angle brackets, the name of the source file and the line number where the  
88975 definition was found. Definitions extracted from object files indicate the filename and location  
88976 counter under which the symbol appeared (for example, *text*).
- 88977 Once a definition of a name has been written, subsequent references to that name contain only  
88978 the reference number of the line where the definition can be found. For undefined references,  
88979 only "<>" shall be written.
- 88980 **STDERR**  
88981 The standard error shall be used only for diagnostic messages.
- 88982 **OUTPUT FILES**  
88983 None.
- 88984 **EXTENDED DESCRIPTION**  
88985 None.
- 88986 **EXIT STATUS**  
88987 The following exit values shall be returned:  
88988 0 Successful completion.  
88989 >0 An error occurred.
- 88990 **CONSEQUENCES OF ERRORS**  
88991 Default.

**88992 APPLICATION USAGE**

88993 Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can  
88994 confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

**88995 EXAMPLES**

88996 Given the following in **file.c**:

```
88997     int i;  
88998     int f ();  
88999     int g ();  
89000     int h ();  
89001     int  
89002     main(void)  
89003     {  
89004         f ();  
89005         g ();  
89006         f ();  
89007     }  
89008     int  
89009     f ()  
89010     {  
89011         i = h ();  
89012     }
```

89013 The command:

```
89014 cflow -i x file.c
```

89015 produces the output:

```
89016 1 main: int(), <file.c 6>  
89017 2   f: int(), <file.c 13>  
89018 3     h: <>  
89019 4     i: int, <file.c 1>  
89020 5     g: <>
```

**89021 RATIONALE**

89022 None.

**89023 FUTURE DIRECTIONS**

89024 None.

**89025 SEE ALSO**

89026 [c17](#), [lex](#), [yacc](#)

89027 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**89028 CHANGE HISTORY**

89029 First released in Issue 2.

**89030 Issue 6**

89031 The normative text is reworded to avoid use of the term “must” for application requirements.

**89032 Issue 7**

89033 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89034 **Issue 8**

89035

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89036

Austin Group Defect 1195 is applied, changing `main()` to `main(void)`.

89037 **NAME**

89038 chgrp — change the file group ownership

89039 **SYNOPSIS**89040 chgrp [-h] *group file...*89041 chgrp -R [-H|-L|-P] *group file...*89042 **DESCRIPTION**89043 The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID  
89044 specified by the *group* operand.89045 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
89046 directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to  
89047 the *chown()* function defined in the System Interfaces volume of POSIX.1-2024, called with the  
89048 following arguments:

- 89049
- The *file* operand shall be used as the *path* argument.
  - The user ID of the file shall be used as the *owner* argument.
  - The specified group ID shall be used as the *group* argument.

89052 Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-  
89053 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-  
89054 group-ID bits of other file types may be cleared.89055 **OPTIONS**89056 The *chgrp* utility shall conform to XBD [Section 12.2](#) (on page 215).

89057 The following options shall be supported by the implementation:

89058 **-h** For each *file* operand that names a file of type symbolic link, *chgrp* shall attempt to  
89059 set the group ID of the symbolic link instead of the file referenced by the symbolic  
89060 link.89061 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
89062 is specified on the command line, *chgrp* shall change the group of the directory  
89063 referenced by the symbolic link and all files in the file hierarchy below it.89064 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
89065 is specified on the command line or encountered during the traversal of a file  
89066 hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic  
89067 link and all files in the file hierarchy below it.89068 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
89069 or encountered during the traversal of a file hierarchy, *chgrp* shall change the group  
89070 ID of the symbolic link. The *chgrp* utility shall not follow the symbolic link to any  
89071 other part of the file hierarchy.89072 **-R** Recursively change file group IDs. For each *file* operand that names a directory,  
89073 *chgrp* shall change the group of the directory and all files in the file hierarchy  
89074 below it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these  
89075 options will be used as the default.89076 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be  
89077 considered an error. The last option specified shall determine the behavior of the utility.

89078 **OPERANDS**

89079 The following operands shall be supported:

89080 *group* A group name from the group database or a numeric group ID. Either specifies a  
 89081 group ID to be given to each file named by one of the *file* operands. If a numeric  
 89082 *group* operand exists in the group database as a group name, the group ID number  
 89083 associated with that group name is used as the group ID.

89084 *file* A pathname of a file whose group ID is to be modified.

89085 **STDIN**

89086 Not used.

89087 **INPUT FILES**

89088 None.

89089 **ENVIRONMENT VARIABLES**89090 The following environment variables shall affect the execution of *chgrp*:

89091 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 89092 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 89093 variables used to determine the values of locale categories.)

89094 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 89095 internationalization variables.

89096 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 89097 characters (for example, single-byte as opposed to multi-byte characters in  
 89098 arguments).

89099 *LC\_MESSAGES*

89100 Determine the locale that should be used to affect the format and contents of  
 89101 diagnostic messages written to standard error.

89102 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

89103 **ASYNCHRONOUS EVENTS**

89104 Default.

89105 **STDOUT**

89106 Not used.

89107 **STDERR**

89108 The standard error shall be used only for diagnostic messages.

89109 **OUTPUT FILES**

89110 None.

89111 **EXTENDED DESCRIPTION**

89112 None.

89113 **EXIT STATUS**

89114 The following exit values shall be returned:

89115 0 The utility executed successfully and all requested changes were made.

89116 &gt;0 An error occurred.

**89117 CONSEQUENCES OF ERRORS**

89118 Default.

**89119 APPLICATION USAGE**

89120 Only the owner of a file or the user with appropriate privileges may change the owner or group  
89121 of a file.

89122 Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the  
89123 *group* specified is not the effective group ID or one of the supplementary group IDs of the calling  
89124 process.

**89125 EXAMPLES**

89126 None.

**89127 RATIONALE**

89128 The System V and BSD versions use different exit status codes. Some implementations used the  
89129 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
89130 can overflow the range of valid exit status values. The standard developers chose to mask these  
89131 by specifying only 0 and >0 as exit values.

89132 The functionality of *chgrp* is described substantially through references to *chown()*. In this way,  
89133 there is no duplication of effort required for describing the interactions of permissions, multiple  
89134 groups, and so on.

**89135 FUTURE DIRECTIONS**

89136 None.

**89137 SEE ALSO**

89138 *chmod*, *chown*

89139 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

89140 XSH *chown()*

**89141 CHANGE HISTORY**

89142 First released in Issue 2.

**89143 Issue 6**

89144 New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
89145 options affect the processing of symbolic links.

89146 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
89147 section to “Default.”.

89148 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to  
89149 make it clear that **-h** and **-R** are optional.

**89150 Issue 7**

89151 SD5-XCU-ERN-8 is applied, removing the **-R** from the first line of the SYNOPSIS.

89152 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89153 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0080 [237,341] is applied.

**89154 Issue 8**

89155 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89156 **NAME**

89157 chmod — change the file modes

89158 **SYNOPSIS**89159 chmod [-R] *mode file...*89160 **DESCRIPTION**89161 The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*  
89162 operand in the way specified by the *mode* operand.89163 It is implementation-defined whether and how the *chmod* utility affects any alternate or  
89164 additional file access control mechanism (see XBD [Section 4.7](#), on page 97) being used for the  
89165 specified file.89166 Only a process whose effective user ID matches the user ID of the file, or a process with  
89167 appropriate privileges, shall be permitted to change the file mode bits of a file.89168 Upon successfully changing the file mode bits of a file, the *chmod* utility shall mark for update  
89169 the last file status change timestamp of the file.89170 **OPTIONS**89171 The *chmod* utility shall conform to XBD [Section 12.2](#) (on page 215).

89172 The following option shall be supported:

89173 **-R** Recursively change file mode bits. For each *file* operand that names a directory,  
89174 *chmod* shall change the file mode bits of the directory and all files in the file  
89175 hierarchy below it.89176 **OPERANDS**

89177 The following operands shall be supported:

89178 *mode* Represents the change to be made to the file mode bits of each file named by one of  
89179 the *file* operands; see the EXTENDED DESCRIPTION section.89180 *file* A pathname of a file whose file mode bits shall be modified.89181 **STDIN**

89182 Not used.

89183 **INPUT FILES**

89184 None.

89185 **ENVIRONMENT VARIABLES**89186 The following environment variables shall affect the execution of *chmod*:89187 *LANG* Provide a default value for the internationalization variables that are unset or null.  
89188 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
89189 variables used to determine the values of locale categories.)89190 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
89191 internationalization variables.89192 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
89193 characters (for example, single-byte as opposed to multi-byte characters in  
89194 arguments).89195 *LC\_MESSAGES*89196 Determine the locale that should be used to affect the format and contents of  
89197 diagnostic messages written to standard error.



- 89198 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 89199 **ASYNCHRONOUS EVENTS**
- 89200 Default.
- 89201 **STDOUT**
- 89202 Not used.
- 89203 **STDERR**
- 89204 The standard error shall be used only for diagnostic messages.
- 89205 **OUTPUT FILES**
- 89206 None.
- 89207 **EXTENDED DESCRIPTION**
- 89208 The *mode* operand shall be either a *symbolic\_mode* expression or a non-negative octal integer. The
- 89209 *symbolic\_mode* form is described by the grammar later in this section.
- 89210 Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*.
- 89211 The operations shall be performed on each *file* in the order in which the **clauses** are specified.
- 89212 The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits,
- 89213 respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.
- 89214 The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode
- 89215 bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who**
- 89216 contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.
- 89217 The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a
- 89218 directory or if the current (unmodified) file mode bits have at least one of the execute bits
- 89219 (S\_IXUSR, S\_IXGRP, or S\_IXOTH) set. It shall be ignored if the file is not a directory and none of
- 89220 the execute bits are set in the current file mode bits.
- 89221 The **permcop** symbols **u**, **g**, and **o** shall represent the current permissions associated with the
- 89222 user, group, and other parts of the file mode bits, respectively. For the remainder of this section,
- 89223 **perm** refers to the non-terminals **perm** and **permcop** in the grammar.
- 89224 If multiple **actionlists** are grouped with a single **wholist** in the grammar, each **actionlist** shall be
- 89225 applied in the order specified with that **wholist**. The *op* symbols shall represent the operation
- 89226 performed, as follows:
- 89227 + If **perm** is not specified, the '+' operation shall not change the file mode bits.
- 89228 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
- 89229 other permissions, except for those with corresponding bits in the file mode creation mask
- 89230 of the invoking process, shall be set.
- 89231 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.
- 89232 – If **perm** is not specified, the '-' operation shall not change the file mode bits.
- 89233 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
- 89234 other permissions, except for those with corresponding bits in the file mode creation mask
- 89235 of the invoking process, shall be cleared.
- 89236 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be
- 89237 cleared.
- 89238 = Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the
- 89239 file mode bits specified in this volume of POSIX.1-2024.

89240 If **perm** is not specified, the '=' operation shall make no further modifications to the file  
89241 mode bits.

89242 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and  
89243 other permissions, except for those with corresponding bits in the file mode creation mask  
89244 of the invoking process, shall be set.

89245 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

89246 When using the symbolic mode form on a regular file, it is implementation-defined whether or  
89247 not:

- 89248 • Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all  
89249 execute bits are currently clear and none are being set are ignored.
- 89250 • Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-  
89251 on-execution bits.
- 89252 • Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all  
89253 execute bits are currently clear are ignored. However, if the command *ls -l file* writes an *s*  
89254 in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is  
89255 set, the commands *chmod u-s file* or *chmod g-s file*, respectively, shall not be ignored.

89256 When using the symbolic mode form on other file types, it is implementation-defined whether  
89257 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are  
89258 honored.

89259 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols  
89260 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be  
89261 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**  
89262 symbol **s**.

89263 XSI The **perm** symbol **t** shall specify the S\_ISVTX bit. When used with a file of type directory, it can  
89264 be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who**  
89265 symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these  
89266 combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other  
89267 than directory is unspecified.

89268 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

89269 For each bit set in the octal number, the corresponding file permission bit shown in the following  
89270 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in  
89271 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-  
89272 execution, bits shown in the following table shall be set; if these bits are not set in the octal  
89273 number, they are cleared. For other file types, it is implementation-defined whether or not  
89274 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are  
89275 honored.

Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit
4000	S_ISUID	0400	S_IRUSR	0040	S_IRGRP	0004	S_IROTH
2000	S_ISGID	0200	S_IWUSR	0020	S_IWGRP	0002	S_IWOTH
1000	S_ISVTX	0100	S_IXUSR	0010	S_IXGRP	0001	S_IXOTH

89280 When bits are set in the octal number other than those listed in the table above, the behavior is  
89281 unspecified.

89282 **Grammar for chmod**

89283 The grammar and lexical conventions in this section describe the syntax for the *symbolic\_mode*  
 89284 operand. The general conventions for this style of grammar are described in [Section 1.3](#) (on page  
 89285 2461). A valid *symbolic\_mode* can be represented as the non-terminal symbol *symbolic\_mode* in  
 89286 the grammar. This formal syntax shall take precedence over the preceding text syntax  
 89287 description.

89288 The lexical processing is based entirely on single characters. Implementations need not allow  
 89289 <blank> characters within the single argument being processed.

```
89290 %start    symbolic_mode
89291 %%

89292 symbolic_mode    : clause
89293                  | symbolic_mode ',' clause
89294                  ;

89295 clause           : actionlist
89296                  | wholist actionlist
89297                  ;

89298 wholist          : who
89299                  | wholist who
89300                  ;

89301 who              : 'u' | 'g' | 'o' | 'a'
89302                  ;

89303 actionlist       : action
89304                  | actionlist action
89305                  ;

89306 action           : op
89307                  | op permlist
89308                  | op permcopy
89309                  ;

89310 permcopy         : 'u' | 'g' | 'o'
89311                  ;

89312 op               : '+' | '-' | '='
89313                  ;

89314 permlist         : perm
89315                  | perm permlist
89316                  ;

89317 XSI perm         : 'r' | 'w' | 'x' | 'X' | 's' | 't'
89318                  ;
```

89319 **EXIT STATUS**

89320 The following exit values shall be returned:

89321 0 The utility executed successfully and all requested changes were made.

89322 >0 An error occurred.

89323 **CONSEQUENCES OF ERRORS**

89324 Default.

89325 **APPLICATION USAGE**

89326 Some implementations of the *chmod* utility change the mode of a directory before the files in the  
 89327 directory when performing a recursive (**-R** option) change; others change the directory mode  
 89328 after the files in the directory. If an application tries to remove read or search permission for a  
 89329 file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying  
 89330 to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users  
 89331 should not try to make a hierarchy inaccessible to themselves.

89332 Some implementations of *chmod* never used the *umask* of the process when changing modes;  
 89333 systems conformant with this volume of POSIX.1-2024 do so when **who** is not specified. Note  
 89334 the difference between:

89335 `chmod a-w file`

89336 which removes all write permissions, and:

89337 `chmod -- -w file`

89338 which removes write permissions that would be allowed if **file** was created with the same  
 89339 *umask*.

89340 Conforming applications should never assume that they know how the set-user-ID and set-  
 89341 group-ID bits on directories are interpreted.

89342 **EXAMPLES**

89343  
89344  
89345  
89346  
89347  
89348  
89349  
89350  
89351  
89352

Mode	Results
<i>a+=</i>	Equivalent to <i>a+,a=</i> ; clears all file mode bits.
<i>go+-w</i>	Equivalent to <i>go+,go-w</i> ; clears group and other write bits.
<i>g=o-w</i>	Equivalent to <i>g=o,g-w</i> ; sets group bit to match other bits and then clears group write bit.
<i>g-r+w</i>	Equivalent to <i>g-r,g+w</i> ; clears group read bit and sets group write bit.
<i>uo=g</i>	Sets owner bits to match group bits and sets other bits to match group bits.

89353 **RATIONALE**

89354 The functionality of *chmod* is described substantially through references to concepts defined in  
 89355 the System Interfaces volume of POSIX.1-2024. In this way, there is less duplication of effort  
 89356 required for describing the interactions of permissions. However, the behavior of this utility is  
 89357 not described in terms of the *chmod()* function from the System Interfaces volume of  
 89358 POSIX.1-2024 because that specification requires certain side-effects upon alternate file access  
 89359 control mechanisms that might not be appropriate, depending on the implementation.

89360 Implementations that support mandatory file and record locking as specified by the 1984  
 89361 /usr/group standard historically used the combination of set-group-ID bit set and group  
 89362 execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the  
 89363 symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory  
 89364 locking mode is not changed without explicit indication that that was what the user intended.  
 89365 Therefore, the details on how the implementation treats these conditions must be defined in the  
 89366 documentation. This volume of POSIX.1-2024 does not require mandatory locking (nor does the  
 89367 System Interfaces volume of POSIX.1-2024), but does allow it as an extension. However, this  
 89368 volume of POSIX.1-2024 does require that the *ls* and *chmod* utilities work consistently in this

89369 area. If `ls -l file` indicates that the set-group-ID bit is set, `chmod g-s file` must clear it (assuming  
89370 appropriate privileges exist to change modes).

89371 The System V and BSD versions use different exit status codes. Some implementations used the  
89372 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
89373 can overflow the range of valid exit status values. This problem is avoided here by specifying  
89374 only 0 and >0 as exit values.

89375 The System Interfaces volume of POSIX.1-2024 indicates that implementation-defined  
89376 restrictions may cause the S\_ISUID and S\_ISGID bits to be ignored. This volume of POSIX.1-2024  
89377 allows the `chmod` utility to choose to modify these bits before calling `chmod()` (or some function  
89378 providing equivalent capabilities) for non-regular files. Among other things, this allows  
89379 implementations that use the set-user-ID and set-group-ID bits on directories to enable extended  
89380 features to handle these extensions in an intelligent manner.

89381 The **X perm** symbol was adopted from BSD-based systems because it provides commonly  
89382 desired functionality when doing recursive (`-R` option) modifications. Similar functionality is  
89383 not provided by the `find` utility. Historical BSD versions of `chmod`, however, only supported **X**  
89384 with `op+`; it has been extended in this volume of POSIX.1-2024 because it is also useful with `op=`.  
89385 (It has also been added for `op-` even though it duplicates `x`, in this case, because it is intuitive  
89386 and easier to explain.)

89387 The grammar was extended with the `permcopy` non-terminal to allow historical-practice forms of  
89388 symbolic modes like `o=u -g` (that is, set the “other” permissions to the permissions of “owner”  
89389 minus the permissions of “group”).

#### 89390 FUTURE DIRECTIONS

89391 None.

#### 89392 SEE ALSO

89393 [ls](#), [umask](#)

89394 [XBD Section 4.7](#) (on page 97), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

89395 XSH `chmod()`

#### 89396 CHANGE HISTORY

89397 First released in Issue 2.

#### 89398 Issue 6

89399 The following new requirements on POSIX implementations derive from alignment with the  
89400 Single UNIX Specification:

- 89401 • Octal modes have been kept and made mandatory despite being marked obsolescent in the  
89402 ISO POSIX-2: 1993 standard.

89403 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
89404 section to “Default.”.

89405 The Open Group Base Resolution bwg2001-010 is applied, adding the description of the  
89406 S\_ISVTX bit and the **t perm** symbol as part of the XSI option.

89407 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/16 is applied, changing the XSI shaded  
89408 text in the EXTENDED DESCRIPTION from:

89409 “The **perm** symbol **t** shall specify the S\_ISVTX bit and shall apply to directories only. The  
89410 effect when using it with any other file type is unspecified. It can be used with the **who**  
89411 symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**  
89412 or **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.”

89413 to:

89414 ``The **perm** symbol **t** shall specify the S\_ISVTX bit. When used with a file of type directory,

89415 it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to

89416 specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning

89417 of these combinations is unspecified. The effect when using the **perm** symbol **t** with any

89418 file type other than directory is unspecified.”

89419 This change is to permit historical behavior.

89420 **Issue 7**

89421 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89422 Austin Group Interpretation 1003.1-2001 #130 is applied, adding text to the DESCRIPTION

89423 about about marking for update the last file status change timestamp of the file.

89424 **Issue 8**

89425 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89426 **NAME**

89427 chown — change the file ownership

89428 **SYNOPSIS**

89429 chown [-h] owner[:group] file...

89430 chown -R [-H|-L|-P] owner[:group] file...

89431 **DESCRIPTION**89432 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID  
89433 specified by the *owner* operand.89434 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the  
89435 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to  
89436 the *chown()* function defined in the System Interfaces volume of POSIX.1-2024, called with the  
89437 following arguments:

- 89438 1. The *file* operand shall be used as the *path* argument.
- 89439 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*  
89440 argument.
- 89441 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used  
89442 as the *group* argument; otherwise, the group ownership shall not be changed.

89443 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-group-  
89444 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-  
89445 group-ID bits of other file types may be cleared.89446 **OPTIONS**89447 The *chown* utility shall conform to XBD [Section 12.2](#) (on page 215).

89448 The following options shall be supported by the implementation:

- 89449 **-h** For each file operand that names a file of type symbolic link, *chown* shall attempt to  
89450 set the user ID of the symbolic link. If a group ID was specified, for each file  
89451 operand that names a file of type symbolic link, *chown* shall attempt to set the  
89452 group ID of the symbolic link.
- 89453 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory  
89454 is specified on the command line, *chown* shall change the user ID (and group ID, if  
89455 specified) of the directory referenced by the symbolic link and all files in the file  
89456 hierarchy below it.
- 89457 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory  
89458 is specified on the command line or encountered during the traversal of a file  
89459 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the  
89460 directory referenced by the symbolic link and all files in the file hierarchy below it.
- 89461 **-P** If the **-R** option is specified and a symbolic link is specified on the command line  
89462 or encountered during the traversal of a file hierarchy, *chown* shall change the  
89463 owner ID (and group ID, if specified) of the symbolic link. The *chown* utility shall  
89464 not follow the symbolic link to any other part of the file hierarchy.
- 89465 **-R** Recursively change file user and group IDs. For each *file* operand that names a  
89466 directory, *chown* shall change the user ID (and group ID, if specified) of the  
89467 directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is  
89468 specified, it is unspecified which of these options will be used as the default.

89469 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be

89470 considered an error. The last option specified shall determine the behavior of the utility.

#### 89471 OPERANDS

89472 The following operands shall be supported:

89473 *owner[:group]* A user ID and optional group ID to be assigned to *file*. The *owner* portion of this  
 89474 operand shall be a user name from the user database or a numeric user ID. Either  
 89475 specifies a user ID which shall be given to each file named by one of the *file*  
 89476 operands. If a numeric *owner* operand exists in the user database as a user name,  
 89477 the user ID number associated with that user name shall be used as the user ID.  
 89478 Similarly, if the *group* portion of this operand is present, it shall be a group name  
 89479 from the group database or a numeric group ID. Either specifies a group ID which  
 89480 shall be given to each file. If a numeric group operand exists in the group database  
 89481 as a group name, the group ID number associated with that group name shall be  
 89482 used as the group ID.

89483 *file* A pathname of a file whose user ID is to be modified.

#### 89484 STDIN

89485 Not used.

#### 89486 INPUT FILES

89487 None.

#### 89488 ENVIRONMENT VARIABLES

89489 The following environment variables shall affect the execution of *chown*:

89490 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 89491 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 89492 variables used to determine the values of locale categories.)

89493 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 89494 internationalization variables.

89495 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 89496 characters (for example, single-byte as opposed to multi-byte characters in  
 89497 arguments).

89498 *LC\_MESSAGES*

89499 Determine the locale that should be used to affect the format and contents of  
 89500 diagnostic messages written to standard error.

89501 *XSI* *NLSPATH* Determine the location of messages objects and message catalogs.

#### 89502 ASYNCHRONOUS EVENTS

89503 Default.

#### 89504 STDOUT

89505 Not used.

#### 89506 STDERR

89507 The standard error shall be used only for diagnostic messages.

#### 89508 OUTPUT FILES

89509 None.



89510 **EXTENDED DESCRIPTION**

89511 None.

89512 **EXIT STATUS**

89513 The following exit values shall be returned:

89514 0 The utility executed successfully and all requested changes were made.

89515 &gt;0 An error occurred.

89516 **CONSEQUENCES OF ERRORS**

89517 Default.

89518 **APPLICATION USAGE**89519 Only the owner of a file or the user with appropriate privileges may change the owner or group  
89520 of a file.89521 Some implementations restrict the use of *chown* to a user with appropriate privileges.89522 **EXAMPLES**

89523 None.

89524 **RATIONALE**89525 The System V and BSD versions use different exit status codes. Some implementations used the  
89526 exit status as a count of the number of errors that occurred; this practice is unworkable since it  
89527 can overflow the range of valid exit status values. These are masked by specifying only 0 and >0  
89528 as exit values.89529 The functionality of *chown* is described substantially through references to functions in the  
89530 System Interfaces volume of POSIX.1-2024. In this way, there is no duplication of effort required  
89531 for describing the interactions of permissions, multiple groups, and so on.89532 The 4.3 BSD method of specifying both owner and group was included in this volume of  
89533 POSIX.1-2024 because:89534 • There are cases where the desired end condition could not be achieved using the *chgrp* and  
89535 *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the  
89536 desired group and the desired owner is not a member of the current group, the *chown()*  
89537 function could fail unless both owner and group are changed at the same time.)89538 • Even if they could be changed independently, in cases where both are being changed, there  
89539 is a 100% performance penalty caused by being forced to invoke both utilities.89540 The BSD syntax *user[.group]* was changed to *user[:group]* in this volume of POSIX.1-2024 because  
89541 the <period> is a valid character in login names (as specified by the Base Definitions volume of  
89542 POSIX.1-2024, login names consist of characters in the portable filename character set). The  
89543 <colon> character was chosen as the replacement for the <period> character because it would  
89544 never be allowed as a character in a user name or group name on historical implementations.89545 The **-R** option is considered by some observers as an undesirable departure from the historical  
89546 UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there  
89547 seemed to be no good reason to require other tools to have to duplicate that functionality.  
89548 However, the **-R** option was deemed an important user convenience, is far more efficient than  
89549 forking a separate process for each element of the directory hierarchy, and is in widespread  
89550 historical use.

89551 **FUTURE DIRECTIONS**

89552 None.

89553 **SEE ALSO**89554 *chgrp, chmod*

89555 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

89556 XSH *chown()*89557 **CHANGE HISTORY**

89558 First released in Issue 2.

89559 **Issue 6**89560 New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
89561 options affect the processing of symbolic links.

89562 The normative text is reworded to avoid use of the term “must” for application requirements.

89563 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS  
89564 section to “Default.”.89565 The “otherwise, ...” text in item 3. of the DESCRIPTION is changed to “otherwise, the group  
89566 ownership shall not be changed”.89567 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to  
89568 make it clear that **-h** and **-R** are optional.89569 **Issue 7**89570 SD5-XCU-ERN-9 is applied, removing the **-R** from the first line of the SYNOPSIS.

89571 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89572 The description of the **-h** and **-P** options is revised.89573 **Issue 8**89574 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89575 **NAME**

89576 cksum — write file checksums and sizes

89577 **SYNOPSIS**89578 cksum [*file*...]89579 **DESCRIPTION**

89580 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)  
 89581 for each input file, and also write to standard output the number of octets in each file. The CRC  
 89582 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996  
 89583 standard (Ethernet).

89584 The encoding for the CRC checksum is defined by the generating polynomial:

$$89585 G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

89586 Mathematically, the CRC value corresponding to a given file shall be defined by the following  
 89587 procedure:

- 89588 1. The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial  
 89589  $M(x)$  of degree  $n-1$ . These *n* bits are the bits from the file, with the most significant bit  
 89590 being the most significant bit of the first octet of the file and the last bit being the least  
 89591 significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral  
 89592 number of octets, followed by one or more octets representing the length of the file as a  
 89593 binary value, least significant octet first. The smallest number of octets capable of  
 89594 representing this integer shall be used.
- 89595 2.  $M(x)$  is multiplied by  $x^{32}$  (that is, shifted left 32 bits) and divided by  $G(x)$  using mod 2  
 89596 division, producing a remainder  $R(x)$  of degree  $\leq 31$ .
- 89597 3. The coefficients of  $R(x)$  are considered to be a 32-bit sequence.
- 89598 4. The bit sequence is complemented and the result is the CRC.

89599 **OPTIONS**

89600 None.

89601 **OPERANDS**

89602 The following operand shall be supported:

89603 *file* A pathname of a file to be checked. If no *file* operands are specified, the standard  
 89604 input shall be used.

89605 **STDIN**

89606 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
 89607 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
 89608 the standard input shall not be used. See the INPUT FILES section.

89609 **INPUT FILES**

89610 The input files can be any file type.

89611 **ENVIRONMENT VARIABLES**89612 The following environment variables shall affect the execution of *cksum*:

89613 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 89614 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 89615 variables used to determine the values of locale categories.)

89616 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 89617 internationalization variables.

89618 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 89619 characters (for example, single-byte as opposed to multi-byte characters in  
 89620 arguments).

89621 *LC\_MESSAGES*  
 89622 Determine the locale that should be used to affect the format and contents of  
 89623 diagnostic messages written to standard error.

89624 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

#### 89625 **ASYNCHRONOUS EVENTS**

89626 Default.

#### 89627 **STDOUT**

89628 For each file processed successfully, the *cksum* utility shall write in the following format:

89629 "%u %d %s\n", <checksum>, <# of octets>, <pathname>

89630 If no *file* operand was specified, the pathname and its leading <space> shall be omitted.

#### 89631 **STDERR**

89632 The standard error shall be used only for diagnostic messages.

#### 89633 **OUTPUT FILES**

89634 None.

#### 89635 **EXTENDED DESCRIPTION**

89636 None.

#### 89637 **EXIT STATUS**

89638 The following exit values shall be returned:

89639 0 All files were processed successfully.

89640 >0 An error occurred.

#### 89641 **CONSEQUENCES OF ERRORS**

89642 Default.

#### 89643 **APPLICATION USAGE**

89644 The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of  
 89645 the same, such as to ensure that files transmitted over noisy media arrive intact. However, this  
 89646 comparison cannot be considered cryptographically secure. This utility should be avoided  
 89647 whenever non-trivial requirements (including safety and security) have to be fulfilled.

89648 Although input files to *cksum* can be any type, the results need not be what would be expected  
 89649 on character special device files or on file types not described by the System Interfaces volume of  
 89650 POSIX.1-2024. Since this volume of POSIX.1-2024 does not specify the block size used when  
 89651 doing input, checksums of character special files need not process all of the data in those files.

89652 The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted  
 89653 between two systems and undergoes any data transformation (such as changing little-endian  
 89654 byte ordering to big-endian), identical CRC values cannot be expected. Implementations  
 89655 performing such transformations may extend *cksum* to handle such situations.

#### 89656 **EXAMPLES**

89657 None.

89658 **RATIONALE**

89659 The *cksum* utility is included in this standard for reasons of portability but is not suitable for uses  
 89660 where non-trivial requirements (including safety and security) have to be fulfilled.  
 89661 Implementations are encouraged to provide utilities that implement hash and integrity  
 89662 checksum algorithms of higher security and to keep up to date with developments in this area.

89663 The following C-language program can be used as a model to describe the algorithm. It assumes  
 89664 that a **char** is one octet. It also assumes that the entire file is available for one pass through the  
 89665 function. This was done for simplicity in demonstrating the algorithm, rather than as an  
 89666 implementation model.

```

89667 static unsigned long crctab[] = {
89668     0x00000000,
89669     0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
89670     0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
89671     0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbdbd,
89672     0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
89673     0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
89674     0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
89675     0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
89676     0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
89677     0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
89678     0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
89679     0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
89680     0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
89681     0xe5ffe643, 0xe8bcc9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
89682     0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
89683     0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcd8bb16,
89684     0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
89685     0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93d8db, 0x6f52c06c,
89686     0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
89687     0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
89688     0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbfa1b04b,
89689     0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
89690     0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
89691     0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
89692     0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
89693     0xc27dede8, 0xcf3ecb31, 0xcbf6d686, 0xd5b88683, 0xd1799b34,
89694     0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcdfd59, 0x608edb80,
89695     0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
89696     0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
89697     0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
89698     0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
89699     0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
89700     0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
89701     0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
89702     0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
89703     0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
89704     0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
89705     0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9fff77d71,
89706     0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
89707     0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
89708     0x4e8ee645, 0x4a4ffb7f2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
  
```

```

89709     0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
89710     0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,
89711     0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
89712     0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
89713     0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
89714     0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
89715     0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xee2ed18,
89716     0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
89717     0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
89718     0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
89719     0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
89720     };

89721     unsigned long memcrc(const unsigned char *b, size_t n)
89722     {
89723     /* Input arguments:
89724     *   const unsigned char*  b == byte sequence to checksum
89725     *   size_t                n == length of sequence
89726     */

89727         register size_t i;
89728         register unsigned c, s = 0;

89729         for (i = n; i > 0; --i) {
89730             c = *b++;
89731             s = (s << 8) ^ crctab[(s >> 24) ^ c];
89732         }

89733         /* Extend with the length of the string. */
89734         while (n != 0) {
89735             c = n & 0377;
89736             n >>= 8;
89737             s = (s << 8) ^ crctab[(s >> 24) ^ c];
89738         }

89739         return ~s;
89740     }

```

89741 The historical practice of writing the number of “blocks” has been changed to writing the  
89742 number of octets, since the latter is not only more useful, but also since historical  
89743 implementations have not been consistent in defining what a “block” meant.

89744 The algorithm used was selected to increase the operational robustness of *cksum*. Neither the  
89745 System V nor BSD *sum* algorithm was selected. Since each of these was different and each was  
89746 the default behavior on those systems, no realistic compromise was available if either were  
89747 selected—some set of historical applications would break. Therefore, the name was changed to  
89748 *cksum*. Although the historical *sum* commands will probably continue to be provided for many  
89749 years, programs designed for portability across systems should use the new name.

89750 The algorithm selected is based on that used by the ISO/IEC 8802-3: 1996 standard (Ethernet) for  
89751 the frame check sequence field. The algorithm used does not match the technical definition of a  
89752 *checksum*; the term is used for historical reasons. The length of the file is included in the CRC  
89753 calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also  
89754 because it guards against inadvertent collisions between files that begin with different series of  
89755 zero octets. The chance that two different files produce identical CRCs is much greater when  
89756 their lengths are not considered. Keeping the length and the checksum of the file itself separate

89757 would yield a slightly more robust algorithm, but historical usage has always been that a single  
 89758 number (the checksum as printed) represents the signature of the file. It was decided that  
 89759 historical usage was the more important consideration.

89760 Early proposals contained modifications to the Ethernet algorithm that involved extracting table  
 89761 values whenever an intermediate result became zero. This was demonstrated to be less robust  
 89762 than the current method and mathematically difficult to describe or justify.

89763 The calculation used is identical to that given in pseudo-code in the referenced Sarwate article.  
 89764 The pseudo-code rendition is:

```
89765 X <- 0; Y <- 0;
89766 for i <- m -1 step -1 until 0 do
89767   begin
89768     T <- X(1) ^ A[i];
89769     X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
89770     comment: f[T] and f'[T] denote the T-th words in the
89771             table f and f' ;
89772     X <- X ^ f[T]; Y <- Y ^ f'[T];
89773   end
```

89774 The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]**  
 89775 represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables  
 89776 **f** and **f'** are a single table containing 32-bit values.

89777 The referenced Sarwate article also discusses generating the table.

#### 89778 **FUTURE DIRECTIONS**

89779 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 89780 value of a <newline> character when <newline> is a terminator or separator in the output  
 89781 format being used, implementations are encouraged to treat this as an error. A future version of  
 89782 this standard may require implementations to treat this as an error.

#### 89783 **SEE ALSO**

89784 XBD [Chapter 8](#) (on page 167)

#### 89785 **CHANGE HISTORY**

89786 First released in Issue 4.

#### 89787 **Issue 7**

89788 Austin Group Interpretation 1003.1-2001 #092 is applied.

89789 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89790 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0081 [446] is applied.

#### 89791 **Issue 8**

89792 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
 89793 directed to display a pathname that contains any bytes that have the encoded value of a  
 89794 <newline> character when <newline> is a terminator or separator in the output format being  
 89795 used.

89796 Austin Group Defect 1041 is applied, changing the APPLICATION USAGE and RATIONALE  
 89797 sections.

89798 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89799 **NAME**

89800 cmp — compare two files

89801 **SYNOPSIS**89802 cmp [-l|-s] *file1 file2*89803 **DESCRIPTION**

89804 The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the  
 89805 same. Under default options, if they differ, it shall write to standard output the byte and line  
 89806 number at which the first difference occurred. Bytes and lines shall be numbered beginning with  
 89807 1.

89808 **OPTIONS**89809 The *cmp* utility shall conform to XBD [Section 12.2](#) (on page 215).

89810 The following options shall be supported:

89811 **-l** (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for  
 89812 each difference.

89813 **-s** Write nothing to standard output or standard error when files differ; indicate  
 89814 differing files through exit status only. It is unspecified whether a diagnostic  
 89815 message is written to standard error when an error is encountered; if a message is  
 89816 not written, the error is indicated through exit status only.

89817 **OPERANDS**

89818 The following operands shall be supported:

89819 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall  
 89820 be used.

89821 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input  
 89822 shall be used.

89823 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or  
 89824 character special file, the results are undefined.

89825 **STDIN**

89826 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the  
 89827 INPUT FILES section.

89828 **INPUT FILES**

89829 The input files can be any file type.

89830 **ENVIRONMENT VARIABLES**89831 The following environment variables shall affect the execution of *cmp*:

89832 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 89833 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 89834 variables used to determine the values of locale categories.)

89835 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 89836 internationalization variables.

89837 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 89838 characters (for example, single-byte as opposed to multi-byte characters in  
 89839 arguments).

89840 **LC\_MESSAGES**

89841 Determine the locale that should be used to affect the format and contents of  
 89842 diagnostic messages written to standard error and informative messages written to



89843 standard output.

89844 XSI **NLSPATH** Determine the location of messages objects and message catalogs.

### 89845 ASYNCHRONOUS EVENTS

89846 Default.

### 89847 STDOUT

89848 In the POSIX locale, results of the comparison shall be written to standard output. When no  
89849 options are used, the format shall be:

89850 "%s %s differ: char %d, line %d\n", *file1*, *file2*,  
89851 <byte number>, <line number>

89852 When the `-l` option is used, the format shall be:

89853 "%d %o %o\n", <byte number>, <differing byte>,  
89854 <differing byte>

89855 for each byte that differs. The first <differing byte> number is from *file1* while the second is from  
89856 *file2*. In both cases, <byte number> shall be relative to the beginning of the file, beginning with 1.

89857 No output shall be written to standard output when the `-s` option is used.

### 89858 STDERR

89859 The standard error shall be used only for diagnostic messages. If the `-l` option is used and *file1*  
89860 and *file2* differ in length, or if the `-s` option is not used and *file1* and *file2* are identical for the  
89861 entire length of the shorter file, in the POSIX locale the following diagnostic message shall be  
89862 written:

89863 "cmp: EOF on %s%s\n", <name of shorter file>, <additional info>

89864 The <additional info> field shall either be null or a string that starts with a <blank> and contains  
89865 no <newline> characters. Some implementations report on the number of lines in this case.

89866 If the `-s` option is used and an error occurs, it is unspecified whether a diagnostic message is  
89867 written to standard error.

### 89868 OUTPUT FILES

89869 None.

### 89870 EXTENDED DESCRIPTION

89871 None.

### 89872 EXIT STATUS

89873 The following exit values shall be returned:

89874 0 The files are identical.

89875 1 The files are different; this includes the case where one file is identical to the first part of the  
89876 other.

89877 >1 An error occurred.

### 89878 CONSEQUENCES OF ERRORS

89879 Default.

89880 **APPLICATION USAGE**

89881 Although input files to *cmp* can be any type, the results might not be what would be expected on  
 89882 character special device files or on file types not described by the System Interfaces volume of  
 89883 POSIX.1-2024. Since this volume of POSIX.1-2024 does not specify the block size used when  
 89884 doing input, comparisons of character special files need not compare all of the data in those files.

89885 For files which are not text files, line numbers simply reflect the presence of a <newline>,  
 89886 without any implication that the file is organized into lines.

89887 Since the behavior of `-s` differs between implementations as to whether error messages are  
 89888 written, the only way to ensure consistent behavior of *cmp* when `-s` is used is to redirect  
 89889 standard error to `/dev/null`.

89890 If error messages are wanted, instead of using `-s` standard output should be redirected to  
 89891 `/dev/null`, and anything written to standard error should be discarded if the exit status is 1. For  
 89892 example:

```
89893 silent_cmp() {
89894     # compare files with no output except error messages
89895     message=$(cmp "$@" 2>&1 >/dev/null)
89896     status=$?
89897     case $status in
89898         (0|1) ;;
89899         (*) printf '%s\n' "$message" ;;
89900     esac
89901     return $status
89902 }
```

89903 **EXAMPLES**

89904 None.

89905 **RATIONALE**

89906 The global language in [Section 1.4](#) (on page 2462) indicates that using two mutually-exclusive  
 89907 options together produces unspecified results. Some System V implementations consider the  
 89908 option usage:

```
89909 cmp -l -s ...
```

89910 to be an error. They also treat:

```
89911 cmp -s -l ...
```

89912 as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

89913 The word **char** in the standard output format comes from historical usage, even though it is  
 89914 actually a byte number. When *cmp* is supported in other locales, implementations are  
 89915 encouraged to use the word *byte* or its equivalent in another language. Users should not  
 89916 interpret this difference to indicate that the functionality of the utility changed between locales.

89917 Some implementations report on the number of lines in the identical-but-shorter file case. This is  
 89918 allowed by the inclusion of the <additional info> fields in the output format. The restriction on  
 89919 having a leading <blank> and no <newline> characters is to make parsing for the filename  
 89920 easier. It is recognized that some filenames containing white-space characters make parsing  
 89921 difficult anyway, but the restriction does aid programs used on systems where the names are  
 89922 predominantly well behaved.

**89923 FUTURE DIRECTIONS**

89924 If this utility is directed to display a pathname that contains any bytes that have the encoded  
89925 value of a <newline> character when <newline> is a terminator or separator in the output  
89926 format being used, implementations are encouraged to treat this as an error. A future version of  
89927 this standard may require implementations to treat this as an error.

89928 Future versions of this standard may require that diagnostic messages are written to standard  
89929 error when the `-s` option is specified.

**89930 SEE ALSO**

89931 *comm*, *diff*

89932 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

**89933 CHANGE HISTORY**

89934 First released in Issue 2.

**89935 Issue 7**

89936 SD5-XCU-ERN-96 is applied, updating the STDERR section.

89937 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89938 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0075 [478] is applied.

**89939 Issue 8**

89940 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
89941 directed to display a pathname that contains any bytes that have the encoded value of a  
89942 <newline> character when <newline> is a terminator or separator in the output format being  
89943 used.

89944 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

89945 **NAME**

89946 comm — select or reject lines common to two files

89947 **SYNOPSIS**89948 comm [-123] *file1 file2*89949 **DESCRIPTION**89950 The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating  
89951 sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and  
89952 lines in both files.89953 If the lines in both files are not ordered according to the collating sequence of the current locale,  
89954 the results are unspecified.89955 If the collating sequence of the current locale does not have a total ordering of all characters (see  
89956 XBD [Section 7.3.2](#), on page 139) and any lines from the input files collate equally but are not  
89957 identical, *comm* shall treat them as different lines and shall expect them to be ordered according  
89958 to a further byte-by-byte comparison using the collating sequence for the POSIX locale; if they  
89959 are not ordered in this way, the output of *comm* can identify such lines as being both unique to  
89960 *file1* and unique to *file2* instead of being in both files.89961 **OPTIONS**89962 The *comm* utility shall conform to XBD [Section 12.2](#) (on page 215).

89963 The following options shall be supported:

- 89964 -1 Suppress the output column of lines unique to *file1*.
- 89965 -2 Suppress the output column of lines unique to *file2*.
- 89966 -3 Suppress the output column of lines duplicated in *file1* and *file2*.

89967 **OPERANDS**

89968 The following operands shall be supported:

89969 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall  
89970 be used.89971 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input  
89972 shall be used.89973 If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or  
89974 character special file, the results are undefined.89975 **STDIN**89976 The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input.  
89977 See the INPUT FILES section.89978 **INPUT FILES**

89979 The input files shall be text files.

89980 **ENVIRONMENT VARIABLES**89981 The following environment variables shall affect the execution of *comm*:89982 *LANG* Provide a default value for the internationalization variables that are unset or null.  
89983 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
89984 variables used to determine the values of locale categories.)89985 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
89986 internationalization variables.

- 89987 **LC\_COLLATE**
- 89988 Determine the locale for the collating sequence *comm* expects to have been used
- 89989 when the input files were sorted.
- 89990 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
- 89991 characters (for example, single-byte as opposed to multi-byte characters in
- 89992 arguments and input files).
- 89993 **LC\_MESSAGES**
- 89994 Determine the locale that should be used to affect the format and contents of
- 89995 diagnostic messages written to standard error.
- 89996 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 89997 **ASYNCHRONOUS EVENTS**
- 89998 Default.
- 89999 **STDOUT**
- 90000 The *comm* utility shall produce output depending on the options selected. If the **-1**, **-2**, and **-3**
- 90001 options are all selected, *comm* shall write nothing to standard output.
- 90002 If the **-1** option is not selected, lines contained only in *file1* shall be written using the format:
- 90003 "%s\n", <line in file1>
- 90004 If the **-2** option is not selected, lines contained only in *file2* are written using the format:
- 90005 "%s%s\n", <lead>, <line in file2>
- 90006 where the string <lead> is as follows:
- 90007 <tab> The **-1** option is not selected.
- 90008 null string The **-1** option is selected.
- 90009 If the **-3** option is not selected, lines contained in both files shall be written using the format:
- 90010 "%s%s\n", <lead>, <line in both>
- 90011 where the string <lead> is as follows:
- 90012 <tab><tab> Neither the **-1** nor the **-2** option is selected.
- 90013 <tab> Exactly one of the **-1** and **-2** options is selected.
- 90014 null string Both the **-1** and **-2** options are selected.
- 90015 If the input files were ordered according to the collating sequence of the current locale, the lines
- 90016 written shall be in the collating sequence of the current locale. If the input files contained any
- 90017 lines that collated equally but were not identical and within each file those lines were ordered
- 90018 according to a further byte-by-byte comparison using the collating sequence for the POSIX
- 90019 locale, then lines written that collate equally but are not identical shall be ordered according to a
- 90020 further byte-by-byte comparison using the collating sequence for the POSIX locale.
- 90021 **STDERR**
- 90022 The standard error shall be used only for diagnostic messages.
- 90023 **OUTPUT FILES**
- 90024 None.

90025 **EXTENDED DESCRIPTION**

90026 None.

90027 **EXIT STATUS**

90028 The following exit values shall be returned:

90029 0 All input files were successfully output as specified.

90030 &gt;0 An error occurred.

90031 **CONSEQUENCES OF ERRORS**

90032 Default.

90033 **APPLICATION USAGE**90034 If the input files are not properly presorted, the output of *comm* might not be useful.

90035 When using *comm* to process pathnames, it is recommended that *LC\_ALL*, or at least *LC\_CTYPE*  
 90036 and *LC\_COLLATE*, are set to *POSIX* or *C* in the environment, since pathnames can contain byte  
 90037 sequences that do not form valid characters in some locales, in which case the utility's behavior  
 90038 would be undefined. In the *POSIX* locale each byte is a valid single-byte character, and therefore  
 90039 this problem is avoided.

90040 If the collating sequence of the current locale does not have a total ordering of all characters,  
 90041 since *comm* treats lines as being the same only if they are identical, some lines can be  
 90042 misleadingly identified as being both unique to *file1* and unique to *file2* if lines that collate  
 90043 equally but are not identical are not ordered in the way that *comm* expects. If the input does not  
 90044 come from utilities (such as *ls* and *sort*) which provide this ordering, the problem can be avoided  
 90045 by pre-sorting the input files using *sort*.

90046 **EXAMPLES**

90047 If a file named *xcu* contains a sorted list of the utilities in this volume of *POSIX.1-2024*, a file  
 90048 named *xpg3* contains a sorted list of the utilities specified in the *X/Open Portability Guide, Issue*  
 90049 *3*, and a file named *svid89* contains a sorted list of the utilities in the *System V Interface*  
 90050 *Definition Third Edition*:

90051 `comm -23 xcu xpg3 | comm -23 - svid89`

90052 would print a list of utilities in this volume of *POSIX.1-2024* not specified by either of the other  
 90053 documents:

90054 `comm -12 xcu xpg3 | comm -12 - svid89`

90055 would print a list of utilities specified by all three documents, and:

90056 `comm -12 xpg3 svid89 | comm -23 - xcu`

90057 would print a list of utilities specified by both *XPG3* and the *SVID*, but not specified in this  
 90058 volume of *POSIX.1-2024*.

90059 **RATIONALE**

90060 None.

90061 **FUTURE DIRECTIONS**

90062 None.

90063 **SEE ALSO**90064 *cmp*, *diff*, *sort*, *uniq*90065 [XBD Section 7.3.2](#) (on page 139), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

90066 **CHANGE HISTORY**

90067 First released in Issue 2.

90068 **Issue 6**

90069 The normative text is reworded to avoid use of the term “must” for application requirements.

90070 **Issue 7**90071 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0076 [963], XCU/TC2-2008/0077  
90072 [663], and XCU/TC2-2008/0078 [963] are applied.90073 **Issue 8**90074 Austin Group Defect 1070 is applied, changing the requirements when any lines from the input  
90075 files collate equally but are not identical.90076 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

90077 **NAME**

90078            command — execute a simple command

90079 **SYNOPSIS**90080            command [-p] *command\_name* [*argument...*]90081            command [-p] [-v|-V] *command\_name*90082 **DESCRIPTION**90083            The *command* utility shall cause the shell to treat the arguments as a simple command, suppressing the shell function lookup that is described in [Section 2.9.1.4](#) (on page 2502), item 1c.90085            If the *command\_name* is the same as the name of one of the special built-in utilities, the special properties in the enumerated list at the beginning of [Section 2.15](#) (on page 2526) shall not occur. In every other respect, if *command\_name* is not the name of a function, the effect of *command* (with no options) shall be the same as omitting *command*, except that *command\_name* does not appear in the command word position in the *command* command, and consequently is not subject to alias substitution (see [Section 2.3.1](#), on page 2477) nor recognized as a reserved word (see [Section 2.4](#), on page 2478).90092            When the *-v* or *-V* option is used, the *command* utility shall provide information concerning how a command name is interpreted by the shell.90094            The *command* utility shall be treated as a declaration utility if the first argument passed to the utility is recognized as a declaration utility. In this case, subsequent words of the form *name=word* shall be expanded in an assignment context. See [Section 2.9.1.1](#) (on page 2500).90097 **OPTIONS**90098            The *command* utility shall conform to XBD [Section 12.2](#) (on page 215).

90099            The following options shall be supported:

90100            **-p**            Perform the command search using a default value for *PATH* that is guaranteed to find all of the standard utilities.90102            **-v**            Write a string to standard output that indicates the pathname or command that will be used by the shell, in the current shell execution environment (see [Section 2.13](#), on page 2522), to invoke *command\_name*, but do not invoke *command\_name*.90105                        • Executable utilities, regular built-in utilities, *command\_names* including a <slash> character, and any implementation-provided functions that are found using the *PATH* variable (as described in [Section 2.9.1.4](#), on page 2502), shall be written as absolute pathnames.90109                        • Shell functions, special built-in utilities, regular built-in utilities not associated with a *PATH* search, and shell reserved words shall be written as just their names.

90112                        • An alias shall be written as a command line that represents its alias definition.

90114                        • Otherwise, no output shall be written and the exit status shall reflect that the name was not found.

90116            **-V**            Write a string to standard output that indicates how the name given in the *command\_name* operand will be interpreted by the shell, in the current shell execution environment (see [Section 2.13](#), on page 2522), but do not invoke *command\_name*. Although the format of this string is unspecified, it shall indicate in which of the following categories *command\_name* falls and shall include the information stated:



- 90122 • Executable utilities, regular built-in utilities, and any implementation-
- 90123 provided functions that are found using the *PATH* variable (as described in
- 90124 [Section 2.9.1.4](#), on page 2502), shall be identified as such and include the
- 90125 absolute pathname in the string.
- 90126 • Other shell functions shall be identified as functions.
- 90127 • Aliases shall be identified as aliases and their definitions included in the
- 90128 string.
- 90129 • Special built-in utilities shall be identified as special built-in utilities.
- 90130 • Regular built-in utilities not associated with a *PATH* search shall be identified
- 90131 as regular built-in utilities. (The term “regular” need not be used.)
- 90132 • Shell reserved words shall be identified as reserved words.

### 90133 OPERANDS

90134 The following operands shall be supported:

90135 *argument* One of the strings treated as an argument to *command\_name*.

90136 *command\_name*

90137 The name of a utility or a special built-in utility.

### 90138 STDIN

90139 Not used.

### 90140 INPUT FILES

90141 None.

### 90142 ENVIRONMENT VARIABLES

90143 The following environment variables shall affect the execution of *command*:

90144 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 90145 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 90146 variables used to determine the values of locale categories.)

90147 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 90148 internationalization variables.

90149 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 90150 characters (for example, single-byte as opposed to multi-byte characters in  
 90151 arguments).

90152 *LC\_MESSAGES*

90153 Determine the locale that should be used to affect the format and contents of  
 90154 diagnostic messages written to standard error and informative messages written to  
 90155 standard output.

90156 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

90157 *PATH* Determine the search path used during the command search described in [Section](#)  
 90158 [2.9.1.4](#) (on page 2502), except as described under the *-p* option.

### 90159 ASYNCHRONOUS EVENTS

90160 Default.

90161 **STDOUT**

90162 When the `-v` option is specified, standard output shall be formatted as:

90163 `"%s\n", <pathname or command>`

90164 When the `-V` option is specified, standard output shall be formatted as:

90165 `"%s\n", <unspecified>`

90166 **STDERR**

90167 The standard error shall be used only for diagnostic messages.

90168 **OUTPUT FILES**

90169 None.

90170 **EXTENDED DESCRIPTION**

90171 None.

90172 **EXIT STATUS**

90173 When the `-v` or `-V` options are specified, the following exit values shall be returned:

90174 0 Successful completion.

90175 >0 The *command\_name* could not be found or an error occurred.

90176 Otherwise, the following exit values shall be returned:

90177 126 The utility specified by *command\_name* was found but could not be invoked.

90178 127 An error occurred in the *command* utility or the utility specified by *command\_name* could not  
90179 be found.

90180 Otherwise, the exit status of *command* shall be that of the simple command specified by the  
90181 arguments to *command*.

90182 **CONSEQUENCES OF ERRORS**

90183 Default.

90184 **APPLICATION USAGE**

90185 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

90186 The order for command search allows functions to override regular built-ins and path searches.  
90187 This utility is necessary to allow functions that have the same name as a utility to call the utility  
90188 (instead of a recursive call to the function).

90189 The system default path is available using *getconf*; however, since *getconf* may need to have the  
90190 *PATH* set up before it can be called itself, the following can be used:

90191 `command -p getconf PATH`

90192 There are some advantages to suppressing the special characteristics of special built-ins on  
90193 occasion. For example:

90194 `command exec > unwritable-file`

90195 does not cause a non-interactive script to abort, so that the output status can be checked by the  
90196 script.

90197 The *command*, *env*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127  
90198 if a utility to be invoked cannot be found, so that applications can distinguish “failure to find a  
90199 utility” from “invoked utility exited with an error indication”. However, the *command* and *nohup*  
90200 utilities also use exit code 127 when an error occurs in those utilities, which means exit code 127  
90201 is not universally a “not found” indicator. The value 127 was chosen because it is not commonly

90202 used for other meanings; most utilities use small values for “normal error conditions” and the  
 90203 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 90204 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 90205 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 90206 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 90207 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 90208 any other reason.

90209 Since the *-v* and *-V* options of *command* produce output in relation to the current shell execution  
 90210 environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell  
 90211 or separate utility execution environment, such as one of the following:

```
90212 (PATH=foo command -v)
90213 nohup command -v
```

90214 it does not necessarily produce correct results. For example, when called with *nohup* or an *exec*  
 90215 function, in a separate utility execution environment, most implementations are not able to  
 90216 identify aliases, functions, or special built-ins.

90217 Two types of regular built-ins could be encountered on a system and these are described  
 90218 separately by *command*. The description of command search in [Section 2.9.1.4](#) (on page 2502)  
 90219 allows for a standard utility to be implemented as a regular built-in as long as it is found in the  
 90220 appropriate place in a *PATH* search. So, for example, *command -v true* might yield */bin/true* or  
 90221 some similar pathname. Other implementation-defined utilities that are not defined by this  
 90222 volume of POSIX.1-2024 might exist only as built-ins and have no pathname associated with  
 90223 them. These produce output identified as (regular) built-ins. Applications encountering these are  
 90224 not able to count on *execing* them, using them with *nohup*, overriding them with a different  
 90225 *PATH*, and so on.

90226 The *command* utility takes on the expansion behavior of the command that it is wrapping.  
 90227 Therefore, in

```
90228 command command export a=~
```

90229 *command* is recognized as a declaration utility, and the command sets the variable *a* to the value  
 90230 of *\$HOME* because it performs tilde-expansion of an assignment context; while

```
90231 command echo a=~
```

90232 outputs the literal string "a=~" because regular expansion can only perform tilde-expansion at  
 90233 the beginning of the word. However, the shell need only perform lexical analysis of the next  
 90234 argument when deciding if *command* should be treated as a declaration utility; therefore, with:

```
90235 var=export; command $var a=~
```

90236 and

```
90237 command -- export a=~
```

90238 it is unspecified whether the word *a=~* is handled in an assignment context or as a regular  
 90239 expansion.

## 90240 EXAMPLES

90241 1. Make a version of *cd* that always prints out the new working directory exactly once:

```
90242 cd() {
90243     command cd "$@" >/dev/null
90244     pwd
90245 }
```

90246 2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

```
90247 IFS='
90248 '
90249 # The preceding value should be <space><tab><newline>.
90250 # Set IFS to its default value.
90251
90252 \unalias -a
90253 # Unset all possible aliases.
90254 # Note that unalias is escaped to prevent an alias
90255 # being used for unalias.
90256
90257 unset -f command
90258 # Ensure command is not a user function.
90259
90260 PATH="$ (command -p getconf PATH) :$PATH"
90261 # Put on a reliable PATH prefix.
90262
90263 # ...
```

90260 At this point, given correct permissions on the directories called by *PATH*, the script has  
 90261 the ability to ensure that any utility it calls is the intended one. It is being very cautious  
 90262 because it assumes that implementation extensions may be present that would allow user  
 90263 functions to exist when it is invoked; this capability is not specified by this volume of  
 90264 POSIX.1-2024, but it is not prohibited as an extension. For example, the *ENV* variable  
 90265 precedes the invocation of the script with a user start-up script. Such a script could define  
 90266 functions to spoof the application.

## 90267 RATIONALE

90268 Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

90269 There is nothing in the description of *command* that implies the command line is parsed any  
 90270 differently from that of any other simple command. For example:

```
90271 command a | b ; c
```

90272 is not parsed in any special way that causes ' | ' or ' ; ' to be treated other than a pipe operator  
 90273 or <semicolon> or that prevents function lookup on **b** or **c**. However, some implementations  
 90274 extend the shell’s assignment syntax, for example to allow an array to be populated with a single  
 90275 assignment, and in order for such an extension to be usable in assignments specified as  
 90276 arguments to *export* and *readonly* these shells have those utility names as separate tokens in their  
 90277 grammar. When *command* is used to execute these utilities it also needs to be a separate token in  
 90278 the grammar so that the same extended assignment syntax can still be recognized in this case.  
 90279 This standard only permits an extension of this nature when the input to the shell would contain  
 90280 a syntax error according to the standard grammar, and therefore it cannot affect how ' | ' and  
 90281 ' ; ' are parsed in the example above. Note that although *command* can be a separate token in the  
 90282 shell’s grammar, it cannot be a reserved word since *command* is a candidate for alias substitution  
 90283 whereas reserved words are not (see [Section 2.3.1](#)).

90284 The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since  
 90285 *command* also goes to the file system to search for utilities, the name *builtin* would not be  
 90286 intuitive.

90287 The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special  
 90288 built-in for the following reasons:

90289 • The removal of exportable functions made the special precedence of a special built-in  
90290 unnecessary.

90291 • A special built-in has special properties (see [Section 2.15](#), on page 2526) that were  
90292 inappropriate for invoking other utilities. For example, two commands such as:

90293 `date > unwritable-file`

90294 `command date > unwritable-file`

90295 would have entirely different results; in a non-interactive script, the former would  
90296 continue to execute the next command, the latter would abort. Introducing this semantic  
90297 difference along with suppressing functions was seen to be non-intuitive.

90298 The `-p` option is present because it is useful to be able to ensure a safe path search that finds all  
90299 the standard utilities. This search might not be identical to the one that occurs through one of the  
90300 `exec` functions (as defined in the System Interfaces volume of POSIX.1-2024) when `PATH` is unset.  
90301 At the very least, this feature is required to allow the script to access the correct version of `getconf`  
90302 so that the value of the default path can be accurately retrieved.

90303 The `command -v` and `-V` options were added to satisfy requirements from users that are  
90304 currently accomplished by three different historical utilities: `type` in the System V shell, `whence` in  
90305 the KornShell, and `which` in the C shell. Since there is no historical agreement on how and what  
90306 to accomplish here, the POSIX `command` utility was enhanced and the historical utilities were left  
90307 unmodified. The C shell `which` merely conducts a path search. The KornShell `whence` is more  
90308 elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases,  
90309 exported aliases, and undefined functions.

90310 The output format of `-V` was left mostly unspecified because human users are its only audience.  
90311 Applications should not be written to care about this information; they can use the output of `-v`  
90312 to differentiate between various types of commands, but the additional information that may be  
90313 emitted by the more verbose `-V` is not needed and should not be arbitrarily constrained in its  
90314 verbosity or localization for application parsing reasons.

#### 90315 **FUTURE DIRECTIONS**

90316 If this utility is directed to display a pathname that contains any bytes that have the encoded  
90317 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
90318 format being used, implementations are encouraged to treat this as an error. A future version of  
90319 this standard may require implementations to treat this as an error.

#### 90320 **SEE ALSO**

90321 [Section 2.9.1.4](#) (on page 2502), [Section 2.9.1.1](#) (on page 2500), [Section 2.13](#) (on page 2522), [Section](#)  
90322 [2.15](#) (on page 2526), `sh`, `type`

90323 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

90324 XSH `exec`

#### 90325 **CHANGE HISTORY**

90326 First released in Issue 4.

#### 90327 **Issue 7**

90328 Austin Group Interpretation 1003.1-2001 #196 is applied, changing the SYNOPSIS to allow `-p` to  
90329 be used with `-v` (or `-V`).

90330 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

90331 The `command` utility is moved from the User Portability Utilities option to the Base. User  
90332 Portability Utilities is now an option for interactive utilities.

90333 The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard  
90334 *getconf\_CS\_PATH* with *getconf PATH*.

90335 **Issue 8**

90336 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
90337 directed to display a pathname that contains any bytes that have the encoded value of a  
90338 <newline> character when <newline> is a terminator or separator in the output format being  
90339 used.

90340 Austin Group Defects 351 and 1393 are applied, requiring *command* to be a declaration utility if  
90341 the first argument passed to the utility is recognized as a declaration utility.

90342 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
90343 this utility is required to be intrinsic.

90344 Austin Group Defect 1117 is applied, changing “implementation-defined” to “implementation-  
90345 provided”.

90346 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

90347 Austin Group Defect 1161 is applied, changing “Utilities” to “Executable utilities” in the  
90348 descriptions of the *-v* and *-V* options.

90349 Austin Group Defect 1431 is applied, changing “item 1b” to “item 1c”.

90350 Austin Group Defect 1586 is applied, adding the *timeout* utility.

90351 Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

90352 Austin Group Defect 1637 is applied, clarifying that (when no options are specified)  
90353 *command\_name* is not subject to alias substitution nor recognized as a reserved word.

90354 **NAME**

90355 compress, uncompress, zcat — compress and decompress data

90356 **SYNOPSIS**

```

90357 XSI compress [-fv] [-b value] [-g | -m algo] [file...]
90358 compress -c [-fv] [-b value] [-g | -m algo] [file]
90359 compress -d [-cfv] [file...]
90360 uncompress [-cfv] [file...]
90361 zcat [file...]

```

90362 **DESCRIPTION**

90363 The *compress* utility, when the **-d** option is not specified, shall apply the compression algorithm  
 90364 identified by the **-g** option or the **-m algo** option to the named files to attempt to reduce their  
 90365 size without loss of information. The *compress* utility with the **-d** option shall apply the  
 90366 appropriate decompression algorithm to the named files to restore the data to their original  
 90367 state.

90368 The *uncompress* utility shall be equivalent to *compress -d*. The *zcat* utility shall be equivalent to  
 90369 *compress -c -d*. If multiple *file* operands are specified, the decompressed data from each input  
 90370 file shall be concatenated to standard output.

90371 When compressing data, unless the **-c** option is specified, after an input file other than standard  
 90372 input has been compressed, the compressed data from the input file shall be stored in a file with  
 90373 the same pathname as the input file but with an added suffix. The added suffix shall be the  
 90374 suffix associated with the algorithm (see the algorithms in Table 3-7, on page 2737). If  
 90375 appending the suffix would make the size of the last component of the output file's pathname  
 90376 exceed {NAME\_MAX} bytes, the command shall fail. If appending the suffix would make the  
 90377 size of the pathname exceed {PATH\_MAX} bytes, the command may fail.

90378 When decompressing data, unless the **-c** option is specified, after an input file other than  
 90379 standard input has been decompressed, the decompressed data from the input file shall be  
 90380 stored in a file with the same pathname as the input file but with the suffix associated with the  
 90381 **OB** algorithm removed. If *file* has no suffix associated with a known compression algorithm or *file*  
 90382 does not exist and does not have a **.Z** suffix, *file* shall be used as the name of the output file, and  
 90383 the default suffix **.Z** shall be appended to *file* to form the input pathname. The behavior is  
 90384 unspecified if the input pathname ends with a suffix other than the suffix associated with the  
 90385 algorithm used to compress the data. When the **-c** option is specified, *file* can have any suffix, or  
 90386 no suffix, and the utility shall use *file* as the input file and examine the file's contents to  
 90387 determine which algorithm to use to decompress the data (it is not an error if *file* does not have a  
 90388 suffix that matches the suffix associated with the compression algorithm).

90389 When compressing or decompressing a file other than standard input and the **-c** option is not  
 90390 specified, if the invoking process has sufficient privilege, the ownership, modes, access time, and  
 90391 modification time of the output file shall match the ownership, modes, access time, and  
 90392 modification time of the input file. After the output file has been successfully created, the input  
 90393 file shall be removed if the invoking process has sufficient privileges. If the invoking process  
 90394 does not have sufficient privileges to remove the input file (for example, if the directory has the  
 90395 S\_ISVTX bit set) the behavior depends on whether the **-f** option is specified: if **-f** is not  
 90396 specified, the output file shall be removed, a diagnostic message shall be written and the utility  
 90397 shall continue processing other files but the final exit status shall be non-zero; if **-f** is specified,  
 90398 the output file shall not be removed and it is unspecified whether the inability to remove the  
 90399 input file is treated as an error. If it is not treated as an error, a warning message may be written

90400 to standard error

90401 If no *file* operands are specified, standard input shall be compressed or decompressed to  
90402 standard output.

90403 OB If an input file that is to be removed after processing has multiple hard links, the *compress* and  
90404 *uncompress* utilities may write a diagnostic message to standard error and do nothing with the  
90405 file; this behavior may depend on whether the *-f* option is specified. If a diagnostic message is  
90406 written, the final exit status shall be non-zero.

#### 90407 OPTIONS

90408 The *compress*, *uncompress*, and *zcat* utilities shall conform to XBD [Section 12.2](#) (on page 215),  
90409 except that Guideline 1 does not apply to *uncompress* since the utility name has ten letters.

90410 The following options shall be supported:

90411 **-b** *value* If the compression algorithm is LZW, *value* specifies the maximum number of bits  
90412 to use in a code. For a conforming application, the *value* argument shall be:

90413  $9 \leq \textit{value} \leq 16$

90414 The implementation may allow values of greater than 16. The default shall be 14,  
90415 15, or 16.

90416 If the compression algorithm is DEFLATE, *value* specifies the compression level.  
90417 For a conforming application, the *value* argument shall be:

90418  $1 \leq \textit{value} \leq 9$

90419 The default shall be 6.

90420 For other algorithms, *value* specifies implementation-defined tuning.

90421 **-c** Write to standard output; the input files shall not be changed, and no output files  
90422 shall be created.

90423 **-d** Decompress files. When invoked with the *-d* option, the *compress* utility shall  
90424 restore previously compressed files to their original state.

90425 **-f** Force compression or decompression of file, even if it does not (for compression)  
90426 actually reduce the size of the file, or if the corresponding output file already  
90427 exists. If the *-f* option is not given and the standard input is a terminal, the user  
90428 shall be prompted as to whether an existing output file should be overwritten. If  
90429 the response is affirmative, the existing file shall be overwritten. If the standard  
90430 input is not a terminal and *-f* is not given, *compress* or *uncompress* shall write a  
90431 diagnostic message to standard error, the existing file shall not be overwritten, and  
90432 the utility shall exit with a status greater than zero. If the *-f* option is specified and  
90433 an input file other than standard input has multiple hard links, it is  
90434 implementation-defined whether the input file is unlinked after the corresponding  
90435 output file is successfully written, or if processing of that file is skipped and a  
90436 diagnostic message is written to standard error.

90437 **-g** Equivalent to *-m gzip*.

90438 **-m** *algo* Use the algorithm defined by *algo* to compress the files. The following algorithms  
90439 shall be supported:



90440 **Table 3-7** Compression algorithms, **-m** option-argument values, and suffixes

Algorithm	<i>algo</i>	Filename Suffix
Adaptive LZW	<b>lzw</b>	<b>.Z</b>
RFC1951 DEFLATE	<b>deflate</b>	<b>.gz</b>
Synonym for DEFLATE	<b>gzip</b>	<b>.gz</b>

90445 Other implementation-defined algorithms may be supported.

90446 If neither of the **-m algo** and **-g** options is specified, **lzw** shall be used as a default  
 90447 *algo* value. Specifying more than one of the mutually exclusive **-g** and **-m algo**  
 90448 options, or multiple **-m algo** options, shall not be considered an error. The last  
 90449 option specified shall determine the behavior of the utility.

90450 On systems not supporting the selected algorithm, the input files shall not be  
 90451 changed and an exit status greater than two shall be returned.

90452 **Note:** The Lempel-Ziv compression algorithm is described in the now-expired US  
 90453 Patent 4464650, which was issued to William Eastman, Abraham Lempel, Jacob  
 90454 Ziv, and Martin Cohn on August 7th, 1984 and assigned to Sperry Corporation.

90455 The Lempel-Ziv-Welch compression algorithm is described in the now-expired  
 90456 US Patent 4558302, which was issued to Terry A. Welch on December 10th, 1985  
 90457 and assigned to Sperry Corporation.

90458 **-v** For *compress*, write the percentage reduction of each file to standard error. For  
 90459 *uncompress*, write messages to standard error concerning the expansion of each file.

## 90460 OPERANDS

90461 The following operand shall be supported:

90462 *file* A pathname of a file to be compressed or decompressed. If a *file* is '-', the utility  
 90463 shall read from standard input at that point in the sequence and write to standard  
 90464 output. If more than one *file* operand is '-', the behavior is unspecified.

## 90465 STDIN

90466 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.

## 90467 INPUT FILES

90468 If *file* operands are specified, the corresponding input files contain the data to be compressed or  
 90469 decompressed.

## 90470 ENVIRONMENT VARIABLES

90471 The following environment variables shall affect the execution of *compress*:

90472 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 90473 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 90474 variables used to determine the values of locale categories.)

90475 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 90476 internationalization variables.

90477 *LC\_COLLATE*

90478 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 90479 character collating elements used in the extended regular expression defined for  
 90480 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

90481 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 90482 characters (for example, single-byte as opposed to multi-byte characters in  
 90483 arguments), the behavior of character classes used in the extended regular  
 90484 expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

90485 *LC\_MESSAGES*  
 90486 Determine the locale used to process affirmative responses, and the locale used to  
 90487 affect the format and contents of diagnostic messages, prompts, and the output  
 90488 from the **-v** option written to standard error.

90489 *NLSPATH* Determine the location of messages objects and message catalogs.

#### 90490 **ASYNCHRONOUS EVENTS**

90491 Default.

#### 90492 **STDOUT**

90493 For the *compress* and *uncompress* utilities, the standard output shall be used if no *file* operands are  
 90494 specified, if a *file* operand is '-', or if the **-c** option is specified. Otherwise, the standard output  
 90495 shall not be used.

90496 The *zcat* utility shall write the decompressed data to the standard output.

#### 90497 **STDERR**

90498 The standard error shall be used only for diagnostic and prompt messages, the optional warning  
 90499 message described in DESCRIPTION, and the output from **-v**.

#### 90500 **OUTPUT FILES**

90501 When decompressing input files other than standard input, the corresponding output files shall  
 90502 contain the decompressed input data. When compressing input files other than standard input,  
 90503 the corresponding output files shall contain the compressed input data. If the selected *algo* is  
 90504 **deflate** or **gzip**, the compressed output shall be in the GZIP format described in RFC 1952. For  
 90505 other algorithms, the compressed output file format is implementation-defined and interchange  
 90506 of such files between implementations (including access via unspecified file sharing  
 90507 mechanisms) is not required by POSIX.1-2024.

#### 90508 **EXTENDED DESCRIPTION**

90509 None.

#### 90510 **EXIT STATUS**

90511 The following exit values shall be returned for *compress*:

90512 0 Successful completion.

90513 1 An error occurred.

90514 2 One or more files were not compressed because they would have increased in size (and the  
 90515 **-f** option was not specified).

90516 >2 An error occurred.

90517 The following exit values shall be returned for *uncompress* and *zcat*:

90518 0 Successful completion.

90519 >0 An error occurred.

#### 90520 **CONSEQUENCES OF ERRORS**

90521 If an error occurs while compressing or decompressing an input file other than standard input,  
 90522 the input file shall remain unmodified.

**90523 APPLICATION USAGE**

90524 The amount of compression obtained depends on the size of the input, the number of bits per  
90525 code, and the distribution of common substrings. Typically, text such as source code or English is  
90526 reduced by 50-60%. Compression is generally much better than that achieved by Huffman  
90527 coding or adaptive Huffman coding (*compact*), and takes less time to compute.

90528 Although *compress* strictly follows the default actions upon receipt of a signal or when an error  
90529 occurs, some unexpected results may occur. In some implementations it is likely that a partially  
90530 compressed file is left in place, alongside its uncompressed input file. Since the general  
90531 operation of *compress* is to delete the uncompressed file only after the *.Z* file has been  
90532 successfully filled, an application should always carefully check the exit status of *compress* before  
90533 arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

90534 In addition to trying *file* and *file.Z* when looking for a file to decompress, some implementations  
90535 of *uncompress* and *zcat* also try suffixes for other known compression algorithms if neither *file* nor  
90536 *file.Z* is found. This version of the standard allows, but does not require this behavior. Portable  
90537 applications should always specify the full pathname (including the suffix) of files to be  
90538 decompressed.

**90539 EXAMPLES**

90540 None.

**90541 RATIONALE**

90542 Earlier versions of this standard limited the number of bits used by conforming applications for  
90543 the *lzw* algorithm to 14 due to address space limitations on 16-bit architectures. Using 15 or 16 is  
90544 a much more common default when using current hardware.

90545 Earlier versions of this standard only supported LZW compression. The standard developers  
90546 noted that existing implementations added other compression utilities, such as *gzip*, and found it  
90547 desirable to support this widespread usage. Some implementations had extended the *compress*  
90548 utility to support such other schemes. The standard developers generalized this practice by the  
90549 addition of the *-m* option, even though this was not previous practice.

90550 The *uncompress -d* option is added to match undocumented existing practice of tested  
90551 implementations.

**90552 FUTURE DIRECTIONS**

90553 If this utility is directed to create a new directory entry that contains any bytes that have the  
90554 encoded value of a <newline> character, implementations are encouraged to treat this as an  
90555 error. A future version of this standard may require implementations to treat this as an error.

90556 When decompressing a file, the requirement to add *.Z* to a *file* operand if the given pathname  
90557 does not include a suffix associated with a known compression algorithm or if *file* does not exist  
90558 and does not already have a *.Z* extension is an obsolescent feature and may be removed in a  
90559 future version.

**90560 SEE ALSO**

90561 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**90562 CHANGE HISTORY**

90563 First released in Issue 4.

**90564 Issue 6**

90565 The normative text is reworded to avoid use of the term “must” for application requirements.

90566 An error case is added for systems not supporting adaptive Lempel-Ziv coding.

90567 **Issue 7**

90568 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

90569 Austin Group Interpretation 1003.1-2001 #125 is applied, revising the ENVIRONMENT  
90570 VARIABLES section.

90571 **Issue 8**

90572 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
90573 filenames containing any bytes that have the encoded value of a <newline> character.

90574 Austin Group Defect 1041 is applied, combining the *compress*, *uncompress* and *zcat* pages into one  
90575 and extensively modifying most sections.

90576 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

90577 **NAME**

90578 cp — copy files

90579 **SYNOPSIS**90580 cp [-PfiP] *source\_file target\_file*90581 cp [-PfiP] *source\_file... target*90582 cp -R [-H|-L|-P] [-fiP] *source\_file... target*90583 **DESCRIPTION**

90584 The first synopsis form is denoted by two operands, neither of which are existing files of type  
 90585 directory. The *cp* utility shall copy the contents of *source\_file* (or, if *source\_file* is a file of type  
 90586 symbolic link, the contents of the file referenced by *source\_file*) to the destination path named by  
 90587 *target\_file*.

90588 The second synopsis form is denoted by two or more operands where the **-R** option is not  
 90589 specified and the first synopsis form is not applicable. It shall be an error if any *source\_file* is a file  
 90590 of type directory, if *target* does not exist, or if *target* does not name a directory. The *cp* utility shall  
 90591 copy the contents of each *source\_file* (or, if *source\_file* is a file of type symbolic link, the contents of  
 90592 the file referenced by *source\_file*) to the destination path named by the concatenation of *target*, a  
 90593 single *<slash>* character if *target* did not end in a *<slash>*, and the last component of *source\_file*.

90594 The third synopsis form is denoted by two or more operands where the **-R** option is specified.  
 90595 The *cp* utility shall copy each file in the file hierarchy rooted in each *source\_file* to a destination  
 90596 path named as follows:

- 90597 • If *target* exists and names an existing directory, the name of the corresponding destination  
 90598 path for each file in the file hierarchy shall be the concatenation of *target*, a single *<slash>*  
 90599 character if *target* did not end in a *<slash>*, and the pathname of the file relative to the  
 90600 directory containing *source\_file*.
- 90601 • If *target* does not exist and two operands are specified, the name of the corresponding  
 90602 destination path for *source\_file* shall be *target*; the name of the corresponding destination  
 90603 path for all other files in the file hierarchy shall be the concatenation of *target*, a *<slash>*  
 90604 character, and the pathname of the file relative to *source\_file*.

90605 It shall be an error if *target* does not exist and more than two operands are specified, or if *target*  
 90606 exists and does not name a directory.

90607 In the following description, the term *dest\_file* refers to the file named by the destination path.  
 90608 The term *source\_file* refers to the file that is being copied, whether specified as an operand or a  
 90609 file in a file hierarchy rooted in a *source\_file* operand. If *source\_file* is a file of type symbolic link:

- 90610 • If the **-R** option was not specified, *cp* shall take actions based on the type and contents of  
 90611 the file referenced by the symbolic link, and not by the symbolic link itself, unless the **-P**  
 90612 option was specified.
- 90613 • If the **-R** option was specified:
  - 90614 — If none of the options **-H**, **-L**, nor **-P** were specified, it is unspecified which of **-H**,  
 90615 **-L**, or **-P** will be used as a default.
  - 90616 — If the **-H** option was specified, *cp* shall take actions based on the type and contents of  
 90617 the file referenced by any symbolic link specified as a *source\_file* operand.
  - 90618 — If the **-L** option was specified, *cp* shall take actions based on the type and contents of  
 90619 the file referenced by any symbolic link specified as a *source\_file* operand or any  
 90620 symbolic links encountered during traversal of a file hierarchy.

90621 — If the **-P** option was specified, *cp* shall copy any symbolic link specified as a  
 90622 *source\_file* operand and any symbolic links encountered during traversal of a file  
 90623 hierarchy, and shall not follow any symbolic links.

90624 For each *source\_file*, the following steps shall be taken:

90625 1. If *source\_file* references the same file as *dest\_file*, *cp* may write a diagnostic message to  
 90626 standard error; it shall do nothing more with *source\_file* and shall go on to any remaining  
 90627 files.

90628 2. If *source\_file* is of type directory, the following steps shall be taken:

90629 a. If the **-R** option was not specified, *cp* shall write a diagnostic message to standard  
 90630 error, do nothing more with *source\_file*, and go on to any remaining files.

90631 b. If *source\_file* was not specified as an operand and *source\_file* is dot or dot-dot, *cp*  
 90632 shall do nothing more with *source\_file* and go on to any remaining files.

90633 c. If *dest\_file* exists and it is a file type not specified by the System Interfaces volume  
 90634 of POSIX.1-2024, the behavior is implementation-defined.

90635 d. If *dest\_file* exists and it is not of type directory, *cp* shall write a diagnostic message  
 90636 to standard error, do nothing more with *source\_file* or any files below *source\_file* in  
 90637 the file hierarchy, and go on to any remaining files.

90638 e. If the directory *dest\_file* does not exist, it shall be created with file permission bits  
 90639 set to the same value as those of *source\_file*, modified by the file creation mask of  
 90640 the user if the **-p** option was not specified, and then bitwise-inclusively OR'ed  
 90641 with S\_IRWXU. If *dest\_file* cannot be created, *cp* shall write a diagnostic message to  
 90642 standard error, do nothing more with *source\_file*, and go on to any remaining files.  
 90643 It is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source\_file*.

90644 f. The files in the directory *source\_file* shall be copied to the directory *dest\_file*, taking  
 90645 the four steps (1 to 4) listed here with the files as *source\_files*.

90646 g. If *dest\_file* was created, its file permission bits shall be changed (if necessary) to be  
 90647 the same as those of *source\_file*, modified by the file creation mask of the user if the  
 90648 **-p** option was not specified.

90649 h. The *cp* utility shall do nothing more with *source\_file* and go on to any remaining  
 90650 files.

90651 3. If *source\_file* is of type regular file, the following steps shall be taken:

90652 a. The behavior is unspecified if *dest\_file* exists and was written by a previous step.  
 90653 Otherwise, if *dest\_file* exists, the following steps shall be taken:

90654 i. If the **-i** option is in effect, the *cp* utility shall write a prompt to the standard  
 90655 error and read a line from the standard input. If the response is not  
 90656 affirmative, *cp* shall do nothing more with *source\_file* and go on to any  
 90657 remaining files.

90658 ii. A file descriptor for *dest\_file* shall be obtained by performing actions  
 90659 equivalent to the *open()* function defined in the System Interfaces volume of  
 90660 POSIX.1-2024 called using *dest\_file* as the *path* argument, and the bitwise-  
 90661 inclusive OR of O\_WRONLY and O\_TRUNC as the *oflag* argument.

90662 iii. If the attempt to obtain a file descriptor fails and the **-f** option is in effect, *cp*  
 90663 shall attempt to remove the file by performing actions equivalent to the  
 90664 *unlink()* function defined in the System Interfaces volume of POSIX.1-2024

90665 called using *dest\_file* as the *path* argument. If this attempt succeeds, *cp* shall  
90666 continue with step 3b.

90667 b. If *dest\_file* does not exist, a file descriptor shall be obtained by performing actions  
90668 equivalent to the *open()* function defined in the System Interfaces volume of  
90669 POSIX.1-2024 called using *dest\_file* as the *path* argument, and the bitwise-inclusive  
90670 OR of *O\_WRONLY* and *O\_CREAT* as the *oflag* argument. The file permission bits  
90671 of *source\_file* shall be the *mode* argument.

90672 c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to  
90673 standard error, do nothing more with *source\_file*, and go on to any remaining files.

90674 d. The contents of *source\_file* shall be written to the file descriptor. Any write errors  
90675 shall cause *cp* to write a diagnostic message to standard error and continue to step  
90676 3e.

90677 e. The file descriptor shall be closed.

90678 f. The *cp* utility shall do nothing more with *source\_file*. If a write error occurred in  
90679 step 3d, it is unspecified if *cp* continues with any remaining files. If no write error  
90680 occurred in step 3d, *cp* shall go on to any remaining files.

90681 4. Otherwise, the **-R** option was specified, and the following steps shall be taken:

90682 a. The *dest\_file* shall be created with the same file type as *source\_file*.

90683 b. If *source\_file* is a file of type FIFO, the file permission bits shall be the same as those  
90684 of *source\_file*, modified by the file creation mask of the user if the **-p** option was not  
90685 specified. Otherwise, the permissions, owner ID, and group ID of *dest\_file* are  
90686 implementation-defined.

90687 If this creation fails for any reason, *cp* shall write a diagnostic message to standard  
90688 error, do nothing more with *source\_file*, and go on to any remaining files.

90689 c. If *source\_file* is a file of type symbolic link, and the options require the symbolic link  
90690 itself to be acted upon, the pathname contained in *dest\_file* shall be the same as the  
90691 pathname contained in *source\_file*.

90692 If this fails for any reason, *cp* shall write a diagnostic message to standard error, do  
90693 nothing more with *source\_file*, and go on to any remaining files.

90694 If the implementation provides additional or alternate access control mechanisms (see XBD  
90695 [Section 4.7](#), on page 97), their effect on copies of files is implementation-defined.

## 90696 OPTIONS

90697 The *cp* utility shall conform to XBD [Section 12.2](#) (on page 215).

90698 The following options shall be supported:

90699 **-f** If a file descriptor for a destination file cannot be obtained, as described in step  
90700 3.a.ii., attempt to unlink the destination file and proceed.

90701 **-H** Take actions based on the type and contents of the file referenced by any symbolic  
90702 link specified as a *source\_file* operand.

90703 **-i** Write a prompt to standard error before copying to any existing non-directory  
90704 destination file. If the response from the standard input is affirmative, the copy  
90705 shall be attempted; otherwise, it shall not.

90706	<b>-L</b>	Take actions based on the type and contents of the file referenced by any symbolic link specified as a <i>source_file</i> operand or any symbolic links encountered during traversal of a file hierarchy.
90707		
90708		
90709	<b>-P</b>	Take actions on any symbolic link specified as a <i>source_file</i> operand or any symbolic link encountered during traversal of a file hierarchy.
90710		
90711	<b>-p</b>	Duplicate the following characteristics of each source file in the corresponding destination file:
90712		
90713		1. The time of last data modification and time of last access. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.
90714		
90715		2. The user ID and group ID. If this duplication fails for any reason, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.
90716		
90717		3. The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.
90718		
90719		
90720		If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but are not duplicated in the destination file, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.
90721		
90722		
90723		
90724		The order in which the preceding characteristics are duplicated is unspecified. The <i>dest_file</i> shall not be deleted if these characteristics cannot be preserved.
90725		
90726	<b>-R</b>	Copy file hierarchies.
90727		Specifying more than one of the mutually-exclusive options <b>-H</b> , <b>-L</b> , and <b>-P</b> shall not be considered an error. The last option specified shall determine the behavior of the utility.
90728		
90729	<b>OPERANDS</b>	
90730		The following operands shall be supported:
90731	<i>source_file</i>	A pathname of a file to be copied. If a <i>source_file</i> operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard input.
90732		
90733	<i>target_file</i>	A pathname of an existing or nonexistent file, used for the output when a single file is copied. If a <i>target_file</i> operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard output.
90734		
90735		
90736	<i>target</i>	A pathname of a directory to contain the copied files.
90737	<b>STDIN</b>	
90738		The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.
90739		
90740	<b>INPUT FILES</b>	
90741		The input files specified as operands may be of any file type.
90742	<b>ENVIRONMENT VARIABLES</b>	
90743		The following environment variables shall affect the execution of <i>cp</i> :
90744	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)
90745		
90746		



90747	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
90748		
90749	<i>LC_COLLATE</i>	
90750		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the <b>yesexpr</b> locale keyword in the <i>LC_MESSAGES</i> category.
90751		
90752		
90753	<i>LC_CTYPE</i>	
90754		Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the <b>yesexpr</b> locale keyword in the <i>LC_MESSAGES</i> category.
90755		
90756		
90757		
90758	<i>LC_MESSAGES</i>	
90759		Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.
90760		
90761		
90762	XSI <i>NLSPATH</i>	Determine the location of messages objects and message catalogs.
90763	<b>ASYNCHRONOUS EVENTS</b>	
90764		Default.
90765	<b>STDOUT</b>	
90766		Not used.
90767	<b>STDERR</b>	
90768		A prompt shall be written to standard error under the conditions specified in the <i>DESCRIPTION</i> section. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.
90769		
90770		
90771	<b>OUTPUT FILES</b>	
90772		The output files may be of any type.
90773	<b>EXTENDED DESCRIPTION</b>	
90774		None.
90775	<b>EXIT STATUS</b>	
90776		The following exit values shall be returned:
90777		0 All requested files (excluding files where a non-affirmative response was given to a request for confirmation) were successfully copied.
90778		
90779		>0 An error occurred.
90780	<b>CONSEQUENCES OF ERRORS</b>	
90781		If <i>cp</i> is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.
90782		
90783		

90784 **APPLICATION USAGE**

90785 The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to  
 90786 prevent users from creating programs that are set-user-ID or set-group-ID to them when copying  
 90787 files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example,  
 90788 if a file is set-user-ID and the copy has a different group ID than the source, a new group of users  
 90789 has execute permission to a set-user-ID program than did previously. In particular, this is a  
 90790 problem for superusers copying users' trees.

90791 **EXAMPLES**

90792 None.

90793 **RATIONALE**

90794 The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally  
 90795 removing files when copying. Although the 4.3 BSD version does not prompt if the standard  
 90796 input is not a terminal, the standard developers decided that use of `-i` is a request for  
 90797 interaction, so when the destination path exists, the utility takes instructions from whatever  
 90798 responds on standard input.

90799 The exact format of the interactive prompts is unspecified. Only the general nature of the  
 90800 contents of prompts are specified because implementations may desire more descriptive  
 90801 prompts than those used on historical implementations. Therefore, an application using the `-i`  
 90802 option relies on the system to provide the most suitable dialog directly with the user, based on  
 90803 the behavior specified.

90804 The `-p` option is historical practice on BSD systems, duplicating the time of last data  
 90805 modification and time of last access. This volume of POSIX.1-2024 extends it to preserve the user  
 90806 and group IDs, as well as the file permissions. This requirement has obvious problems in that  
 90807 the directories are almost certainly modified after being copied. This volume of POSIX.1-2024  
 90808 requires that the modification times be preserved. The statement that the order in which the  
 90809 characteristics are duplicated is unspecified is to permit implementations to provide the  
 90810 maximum amount of security for the user. Implementations should take into account the  
 90811 obvious security issues involved in setting the owner, group, and mode in the wrong order or  
 90812 creating files with an owner, group, or mode different from the final value.

90813 It is unspecified whether `cp` writes diagnostic messages when the user and group IDs cannot be  
 90814 set due to the widespread practice of users using `-p` to duplicate some portion of the file  
 90815 characteristics, indifferent to the duplication of others. Historic implementations only write  
 90816 diagnostic messages on errors other than [EPERM].

90817 Earlier versions of this standard included support for the `-r` option to copy file hierarchies. The  
 90818 `-r` option is historical practice on BSD and BSD-derived systems. This option is no longer  
 90819 specified by POSIX.1-2024 but may be present in some implementations. The `-R` option was  
 90820 added as a close synonym to the `-r` option, selected for consistency with all other options in this  
 90821 volume of POSIX.1-2024 that do recursive directory descent.

90822 The difference between `-R` and the removed `-r` option is in the treatment by `cp` of file types other  
 90823 than regular and directory. It was implementation-defined how the `-` option treated special files  
 90824 to allow both historical implementations and those that chose to support `-r` with the same  
 90825 abilities as `-R` defined by this volume of POSIX.1-2024. The original `-r` flag, for historic reasons,  
 90826 did not handle special files any differently from regular files, but always read the file and copied  
 90827 its contents. This had obvious problems in the presence of special file types; for example,  
 90828 character devices, FIFOs, and sockets.

90829 When a failure occurs during the copying of a file hierarchy, `cp` is required to attempt to copy  
 90830 files that are on the same level in the hierarchy or above the file where the failure occurred. It is  
 90831 unspecified if `cp` shall attempt to copy files below the file where the failure occurred (which

90832 cannot succeed in any case).

90833 Permissions, owners, and groups of created special file types have been deliberately left as  
90834 implementation-defined. This is to allow systems to satisfy special requirements (for example,  
90835 allowing users to create character special devices, but requiring them to be owned by a certain  
90836 group). In general, it is strongly suggested that the permissions, owner, and group be the same  
90837 as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable  
90838 that additional privileges are required to create block, character, or other implementation-  
90839 defined special file types.

90840 Additionally, the `-p` option explicitly requires that all set-user-ID and set-group-ID permissions  
90841 be discarded if any of the owner or group IDs cannot be set. This is to keep users from  
90842 unintentionally giving away special privilege when copying programs.

90843 When creating regular files, historical versions of *cp* use the mode of the source file as modified  
90844 by the file mode creation mask. Other choices would have been to use the mode of the source file  
90845 unmodified by the creation mask or to use the same mode as would be given to a new file  
90846 created by the user (plus the execution bits of the source file) and then modify it by the file mode  
90847 creation mask. In the absence of any strong reason to change historic practice, it was in large part  
90848 retained.

90849 When creating directories, historical versions of *cp* use the mode of the source directory, plus  
90850 read, write, and search bits for the owner, as modified by the file mode creation mask. This is  
90851 done so that *cp* can copy trees where the user has read permission, but the owner does not. A  
90852 side-effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the  
90853 copy is done, historical versions of *cp* set the permissions on the created directory to be the same  
90854 as the source directory, unmodified by the file creation mask.

90855 This behavior has been modified so that *cp* is always able to create the contents of the directory,  
90856 regardless of the file creation mask. After the copy is done, the permissions are set to be the same  
90857 as the source directory, as modified by the file creation mask. This latter change from historical  
90858 behavior is to prevent users from accidentally creating directories with permissions beyond  
90859 those they would normally set and for consistency with the behavior of *cp* in creating files.

90860 It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations  
90861 are strongly encouraged to do so. Historical implementations have detected the attempt in most  
90862 cases.

90863 There are two methods of copying subtrees in this volume of POSIX.1-2024. The other method is  
90864 described as part of the *pax* utility (see *pax*). Both methods are historical practice. The *cp* utility  
90865 provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each  
90866 provides additional functionality to the other; in particular, *pax* maintains the hard-link structure  
90867 of the hierarchy, while *cp* does not. It is the intention of the standard developers that the results  
90868 be similar (using appropriate option combinations in both utilities). The results are not required  
90869 to be identical; there seemed insufficient gain to applications to balance the difficulty of  
90870 implementations having to guarantee that the results would be exactly identical.

90871 The wording allowing *cp* to copy a directory to implementation-defined file types not specified  
90872 by the System Interfaces volume of POSIX.1-2024 is provided so that implementations  
90873 supporting symbolic links are not required to prohibit copying directories to symbolic links.  
90874 Other extensions to the System Interfaces volume of POSIX.1-2024 file types may need to use  
90875 this loophole as well.

**90876 FUTURE DIRECTIONS**

90877 If this utility is directed to create a new directory entry that contains any bytes that have the  
90878 encoded value of a <newline> character, implementations are encouraged to treat this as an  
90879 error. A future version of this standard may require implementations to treat this as an error.

**90880 SEE ALSO**

90881 *mv, find, ln, pax*

90882 XBD Section 4.7 (on page 97), Chapter 8 (on page 167), Section 12.2 (on page 215)

90883 XSH *open()*, *unlink()*

**90884 CHANGE HISTORY**

90885 First released in Issue 2.

**90886 Issue 6**

90887 The **-r** option is marked obsolescent.

90888 The new options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These  
90889 options affect the processing of symbolic links.

90890 IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the **-P** option.

90891 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/18 is applied, correcting an error in the  
90892 SEE ALSO section.

**90893 Issue 7**

90894 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
90895 *LC\_MESSAGES* environment variable.

90896 Austin Group Interpretations 1003.1-2001 #092, #164, #165, and #168 are applied.

90897 SD5-XCU-ERN-31 and SD5-XCU-ERN-42 are applied, updating the DESCRIPTION.

90898 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

90899 SD5-XCU-ERN-102 is applied, clarifying the **-i** option within the OPTIONS section.

90900 The obsolescent **-r** option is removed.

90901 The **-P** option is added to the SYNOPSIS and to the DESCRIPTION with respect to the **-R**  
90902 option.

**90903 Issue 8**

90904 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
90905 filenames containing any bytes that have the encoded value of a <newline> character.

90906 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

90907 Austin Group Defect 1732 is applied, changing the EXIT STATUS section.

90908 **NAME**

90909 crontab — schedule periodic background work

90910 **SYNOPSIS**90911 crontab [*file*]90912 UP crontab [**-e** | **-l** | **-r**]90913 **DESCRIPTION**

90914 UP The *crontab* utility shall create, replace, or edit a user's crontab entry; a crontab entry is a list of  
 90915 commands and the times at which they shall be executed. The new crontab entry can be input by  
 90916 UP specifying *file* or input from standard input if no *file* operand is specified, or by using an editor,  
 90917 if **-e** is specified.

90918 Upon execution of a command from a crontab entry, the implementation shall supply a default  
 90919 environment, defining at least the following environment variables:

90920 *HOME* A pathname of the user's home directory.90921 *LOGNAME* The user's login name.90922 *PATH* A string representing a search path guaranteed to find all of the standard utilities.90923 *SHELL* A pathname of the command interpreter. When *crontab* is invoked as specified by  
 90924 this volume of POSIX.1-2024, the value shall be a pathname for *sh*.

90925 The values of these variables when *crontab* is invoked as specified by this volume of  
 90926 POSIX.1-2024 shall not affect the default values provided when the scheduled command is run.

90927 If standard output and standard error are not redirected by commands executed from the  
 90928 crontab entry, any generated output or errors shall be mailed, via an implementation-defined  
 90929 method, to the user.

90930 XSI Users shall be permitted to use *crontab* if their names appear in the file **cron.allow** which is  
 90931 located in an implementation-defined directory. If that file does not exist, the file **cron.deny**,  
 90932 which is located in an implementation-defined directory, shall be checked to determine whether  
 90933 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate  
 90934 privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage  
 90935 shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line.

90936 **OPTIONS**90937 The *crontab* utility shall conform to XBD [Section 12.2](#) (on page 215).

90938 The following options shall be supported:

90939 UP **-e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if  
 90940 the crontab entry does not exist. When editing is complete, the entry shall be  
 90941 installed as the user's crontab entry.

90942 **-l** (The letter ell.) List the invoking user's crontab entry.90943 **-r** Remove the invoking user's crontab entry.90944 **OPERANDS**

90945 The following operand shall be supported:

90946 *file* The pathname of a file that contains specifications, in the format defined in the  
 90947 INPUT FILES section, for crontab entries.

90948 **STDIN**

90949 See the INPUT FILES section.

90950 **INPUT FILES**

90951 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file  
 90952 consisting of lines of six fields each. The fields shall be separated by <blank> characters. The  
 90953 first five fields shall be integer patterns that specify the following:

- 90954 1. Minute [0,59]
- 90955 2. Hour [0,23]
- 90956 3. Day of the month [1,31]
- 90957 4. Month of the year [1,12]
- 90958 5. Day of the week ([0,6] with 0=Sunday)

90959 Each of these patterns can be either an <asterisk> (meaning all valid values), an element, or a list  
 90960 of elements separated by <comma> characters. An element shall be either a number or two  
 90961 numbers separated by a <hyphen-minus> (meaning an inclusive range). The specification of  
 90962 days can be made by two fields (day of the month and day of the week). If month, day of month,  
 90963 and day of week are all <asterisk> characters, every day shall be matched. If either the month or  
 90964 day of month is specified as an element or list, but the day of week is an <asterisk>, the month  
 90965 and day of month fields shall specify the days that match. If both month and day of month are  
 90966 specified as an <asterisk>, but day of week is an element or list, then only the specified days of  
 90967 the week match. Finally, if either the month or day of month is specified as an element or list,  
 90968 and the day of week is also specified as an element or list, then any day matching either the  
 90969 month and day of month, or the day of week, shall be matched.

90970 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified  
 90971 times. A <percent-sign> character in this field shall be translated to a <newline>. Any character  
 90972 preceded by a <backslash> (including the '%') shall cause that character to be treated literally.  
 90973 Only the first line (up to a '%' or end-of-line) of the command field shall be executed by the  
 90974 command interpreter. The other lines shall be made available to the command as standard input.

90975 Blank lines and those whose first non-<blank> is '#' shall be ignored.

90976 XSI The text files **cron.allow** and **cron.deny**, which are located in an implementation-defined  
 90977 directory, shall contain zero or more user names, one per line, of users who are, respectively,  
 90978 authorized or denied access to the service underlying the *crontab* utility.

90979 **ENVIRONMENT VARIABLES**

90980 The following environment variables shall affect the execution of *crontab*:

90981 **EDITOR** Determine the editor to be invoked when the **-e** option is specified. The default  
 90982 editor shall be *vi*.

90983 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 90984 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 90985 variables used to determine the values of locale categories.)

90986 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 90987 internationalization variables.

90988 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 90989 characters (for example, single-byte as opposed to multi-byte characters in  
 90990 arguments and input files).

90991 *LC\_MESSAGES*  
 90992 Determine the locale that should be used to affect the format and contents of  
 90993 diagnostic messages written to standard error.

90994 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

90995 **ASYNCHRONOUS EVENTS**  
 90996 Default.

90997 **STDOUT**  
 90998 If the `-l` option is specified, the crontab entry shall be written to the standard output.

90999 **STDERR**  
 91000 The standard error shall be used only for diagnostic messages.

91001 **OUTPUT FILES**  
 91002 None.

91003 **EXTENDED DESCRIPTION**  
 91004 None.

91005 **EXIT STATUS**  
 91006 The following exit values shall be returned:  
 91007 0 Successful completion.  
 91008 >0 An error occurred.

91009 **CONSEQUENCES OF ERRORS**  
 91010 UP The user's crontab entry is not submitted, removed, `edited`, or listed.

91011 **APPLICATION USAGE**  
 91012 The format of the crontab entry shown here is guaranteed only for the POSIX locale. Other  
 91013 cultures may be supported with substantially different interfaces, although implementations are  
 91014 encouraged to provide comparable levels of functionality.

91015 The default settings of the *HOME*, *LOGNAME*, *PATH*, and *SHELL* variables that are given to the  
 91016 scheduled job are not affected by the settings of those variables when *crontab* is run; as stated,  
 91017 they are defaults. The text about "invoked as specified by this volume of POSIX.1-2024" means  
 91018 that the implementation may provide extensions that allow these variables to be affected at  
 91019 runtime, but that the user has to take explicit action in order to access the extension, such as give  
 91020 a new option flag or modify the format of the crontab entry.

91021 A typical user error is to type only *crontab*; this causes the system to wait for the new crontab  
 91022 entry on standard input. If end-of-file is typed (generally `<control>-D`), the crontab entry is  
 91023 replaced by an empty file. In this case, the user should type the interrupt character, which  
 91024 prevents the crontab entry from being replaced.

91025 **EXAMPLES**

91026 1. Clean up files named **core** every weekday morning at 3:15 am:  
 91027 `15 3 * * 1-5 find "$HOME" -name core -exec rm -f {} + 2>/dev/null`

91028 2. Mail a birthday greeting:  
 91029 `0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.`

91030 3. As an example of specifying the two types of days:  
 91031 `0 0 1,15 * 1`

91032 would run a command on the first and fifteenth of each month, as well as on every  
91033 Monday. To specify days by only one field, the other field should be set to '\*'; for  
91034 example:

91035 0 0 \* \* 1

91036 would run a command only on Mondays.

#### 91037 RATIONALE

91038 All references to a *cron* daemon and to *cron files* have been omitted. Although historical  
91039 implementations have used this arrangement, there is no reason to limit future implementations.

91040 This description of *crontab* is designed to support only users with normal privileges. The format  
91041 of the input is based on the System V *crontab*; however, there is no requirement here that the  
91042 actual system database used by the *cron* daemon (or a similar mechanism) use this format  
91043 internally. For example, systems derived from BSD are likely to have an additional field  
91044 appended that indicates the user identity to be used when the job is submitted.

91045 The `-e` option was adopted from the SVID as a user convenience, although it does not exist in all  
91046 historical implementations.

#### 91047 FUTURE DIRECTIONS

91048 None.

#### 91049 SEE ALSO

91050 *at*

91051 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

#### 91052 CHANGE HISTORY

91053 First released in Issue 2.

#### 91054 Issue 6

91055 This utility is marked as part of the User Portability Utilities option.

91056 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 91057 Issue 7

91058 The *crontab* utility (except for the `-e` option) is moved from the User Portability Utilities option  
91059 to the Base. User Portability Utilities is now an option for interactive utilities.

91060 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
91061 by the *crontab* utility.

91062 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91063 The first example is changed to remove the unreliable use of `find | xargs`.

91064 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0079 [584] is applied.

#### 91065 Issue 8

91066 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

91067 Austin Group Defect 1141 is applied, changing “core files” to “files named core”.



91068 **NAME**

91069           csplit — split files based on context

91070 **SYNOPSIS**91071           csplit [-ks] [-f *prefix*] [-n *number*] *file arg...*91072 **DESCRIPTION**91073           The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into  
91074           other files as directed by the *arg* operands, and write the sizes of the files.91075 **OPTIONS**91076           The *csplit* utility shall conform to XBD [Section 12.2](#) (on page 215).

91077           The following options shall be supported:

91078           **-f** *prefix*   Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is *xx00 ... xxn*. If  
91079           the *prefix* argument would create a filename exceeding {NAME\_MAX} bytes, an  
91080           error shall result, *csplit* shall exit with a diagnostic message, and no files shall be  
91081           created.91082           **-k**           Leave previously created files intact. By default, *csplit* shall remove created files if  
91083           an error occurs.91084           **-n** *number*   Use *number* decimal digits to form filenames for the file pieces. The default shall be  
91085           2.91086           **-s**           Suppress the output of file size messages.91087 **OPERANDS**

91088           The following operands shall be supported:

91089           *file*           The pathname of a text file to be split. If *file* is '-', the standard input shall be  
91090           used.91091           Each *arg* operand can be one of the following:91092           /*rexp*/*offset*91093           A file shall be created using the content of the lines from the current line up to, but  
91094           not including, the line that results from the evaluation of the regular expression  
91095           with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for  
91096           basic regular expressions described in XBD [Section 9.3](#) (on page 181). The  
91097           application shall use the sequence "\/" to specify a <slash> character within the  
91098           *rexp*. The optional offset shall be a positive or negative integer value representing a  
91099           number of lines. A positive integer value can be preceded by '+'. If the selection  
91100           of lines from an *offset* expression of this type would create a file with zero lines, or  
91101           one with greater than the number of lines left in the input file, the results are  
91102           unspecified. After the section is created, the current line shall be set to the line that  
91103           results from the evaluation of the regular expression with any offset applied. If the  
91104           current line is the first line in the file and a regular expression operation has not yet  
91105           been performed, the pattern match of *rexp* shall be applied from the current line to  
91106           the end of the file. Otherwise, the pattern match of *rexp* shall be applied from the  
91107           line following the current line to the end of the file.91108           %*rexp*%*offset*91109           Equivalent to */rexp/offset*, except that no file shall be created for the selected  
91110           section of the input file. The application shall use the sequence "%%" to specify a  
91111           <percent-sign> character within the *rexp*.

- 91112 *line\_no* Create a file from the current line up to (but not including) the line number *line\_no*.  
 91113 Lines in the file shall be numbered starting at one. The current line becomes  
 91114 *line\_no*.
- 91115 {*num*} Repeat operand. This operand can follow any of the operands described  
 91116 previously. If it follows a *rexp* type operand, that operand shall be applied *num*  
 91117 more times. If it follows a *line\_no* operand, the file shall be split every *line\_no* lines,  
 91118 *num* times, from that point.
- 91119 An error shall be reported if an operand does not reference a line between the current position  
 91120 and the end of the file.
- 91121 **STDIN**
- 91122 See the INPUT FILES section.
- 91123 **INPUT FILES**
- 91124 The input file shall be a text file.
- 91125 **ENVIRONMENT VARIABLES**
- 91126 The following environment variables shall affect the execution of *csplit*:
- 91127 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 91128 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 91129 variables used to determine the values of locale categories.)
- 91130 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 91131 internationalization variables.
- 91132 *LC\_COLLATE*
- 91133 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 91134 character collating elements within regular expressions.
- 91135 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 91136 characters (for example, single-byte as opposed to multi-byte characters in  
 91137 arguments and input files) and the behavior of character classes within regular  
 91138 expressions.
- 91139 *LC\_MESSAGES*
- 91140 Determine the locale that should be used to affect the format and contents of  
 91141 diagnostic messages written to standard error.
- 91142 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 91143 **ASYNCHRONOUS EVENTS**
- 91144 If the *-k* option is specified, created files shall be retained. Otherwise, the default action occurs.
- 91145 **STDOUT**
- 91146 Unless the *-s* option is used, the standard output shall consist of one line per file created, with a  
 91147 format as follows:
- 91148 "%d\n", <*file size in bytes*>
- 91149 **STDERR**
- 91150 The standard error shall be used only for diagnostic messages.
- 91151 **OUTPUT FILES**
- 91152 The output files shall contain portions of the original input file; otherwise, unchanged.

**91153 EXTENDED DESCRIPTION**

91154 None.

**91155 EXIT STATUS**

91156 The following exit values shall be returned:

91157 0 Successful completion.

91158 >0 An error occurred.

**91159 CONSEQUENCES OF ERRORS**

91160 By default, created files shall be removed if an error occurs. When the `-k` option is specified,  
91161 created files shall not be removed if an error occurs.

**91162 APPLICATION USAGE**

91163 None.

**91164 EXAMPLES**

91165 1. This example creates four files, `cobol00` ... `cobol03`:

91166 `csplit -f cobol file '/procedure division/' /par5./ /par16./`

91167 After editing the split files, they can be recombined as follows:

91168 `cat cobol0[0-3] > file`

91169 Note that this example overwrites the original file.

91170 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up  
91171 to 9999 lines; this is because lines in the file are numbered from 1 rather than zero, for  
91172 historical reasons:

91173 `csplit -k file 100 {99}`

91174 3. Assuming that `prog.c` follows the C-language coding convention of ending routines with  
91175 a `'}'` at the beginning of the line, this example creates a file containing each separate C  
91176 routine (up to 21) in `prog.c`:

91177 `csplit -k prog.c '%main(%' '/^}'+1' {20}`

**91178 RATIONALE**

91179 The `-n` option was added to extend the range of filenames that could be handled.

91180 Consideration was given to adding a `-a` flag to use the alphabetic filename generation used by  
91181 the historical `split` utility, but the functionality added by the `-n` option was deemed to make  
91182 alphabetic naming unnecessary.

**91183 FUTURE DIRECTIONS**

91184 If this utility is directed to create a new directory entry that contains any bytes that have the  
91185 encoded value of a `<newline>` character, implementations are encouraged to treat this as an  
91186 error. A future version of this standard may require implementations to treat this as an error.

**91187 SEE ALSO**

91188 *sed*, *split*

91189 XBD Chapter 8 (on page 167), Section 9.3 (on page 181), Section 12.2 (on page 215)

**91190 CHANGE HISTORY**

91191 First released in Issue 2.

- 91192 **Issue 5**  
91193 The FUTURE DIRECTIONS section is added.
- 91194 **Issue 6**  
91195 This utility is marked as part of the User Portability Utilities option.  
91196 The APPLICATION USAGE section is added.  
91197 The description of regular expression operands is changed to align with the IEEE P1003.2b draft  
91198 standard.  
91199 The normative text is reworded to avoid use of the term “must” for application requirements.
- 91200 **Issue 7**  
91201 The *csplit* utility is moved from the User Portability Utilities option to the Base. User Portability  
91202 Utilities is now an option for interactive utilities.  
91203 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.  
91204 The SYNOPSIS and OPERANDS sections are revised to use a single *arg* to split a file into two  
91205 pieces.
- 91206 **Issue 8**  
91207 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
91208 filenames containing any bytes that have the encoded value of a <newline> character.  
91209 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

91210 **NAME**91211 ctags — create a tags file (**DEVELOPMENT**)91212 **SYNOPSIS**91213 CD SD ctags [-a] [-f *tagsfile*] *pathname...*91214 ctags -x *pathname...*91215 **DESCRIPTION**

91216 The *ctags* utility shall write a *tagsfile* or an index of objects from C-language source files specified  
 91217 by the *pathname* operands. The *tagsfile* shall list the locators of C-language objects within the  
 91218 source files. A locator consists of a name, *pathname*, and either a search pattern or a line number  
 91219 that can be used in searching for the object definition. The objects that shall be recognized are  
 91220 specified in the EXTENDED DESCRIPTION section.

91221 **OPTIONS**91222 The *ctags* utility shall conform to XBD [Section 12.2](#) (on page 215).

91223 The following options shall be supported:

91224 **-a** Append to *tagsfile*.91225 **-f *tagsfile*** Write the object locator lists into *tagsfile* instead of the default file named **tags** in the  
91226 current directory.91227 **-x** Produce a list of object names, the line number, and filename in which each is  
91228 defined, as well as the text of that line, and write this to the standard output. A  
91229 *tagsfile* shall not be created when **-x** is specified.91230 **OPERANDS**91231 The following *pathname* operands are supported:91232 ***file.c*** Files with basenames ending with the **.c** suffix shall be treated as C-language  
91233 source code. Such files that are not valid input to *c17* produce unspecified results.91234 ***file.h*** Files with basenames ending with the **.h** suffix shall be treated as C-language  
91235 source code. Such files that are not valid input to *c17* produce unspecified results.

91236 The handling of other files is implementation-defined.

91237 **STDIN**

91238 See the INPUT FILES section.

91239 **INPUT FILES**

91240 The input files shall be text files containing C-language source code.

91241 **ENVIRONMENT VARIABLES**91242 The following environment variables shall affect the execution of *ctags*:91243 **LANG** Provide a default value for the internationalization variables that are unset or null.  
91244 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
91245 variables used to determine the values of locale categories.)91246 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
91247 internationalization variables.91248 **LC\_COLLATE**91249 Determine the order in which output is sorted for the **-x** option. The POSIX locale  
91250 determines the order in which the *tagsfile* is written.

- 91251 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 91252 characters (for example, single-byte as opposed to multi-byte characters in  
 91253 arguments and input files). If the locale is not compatible with the C locale  
 91254 described by the ISO C standard, the results are unspecified.
- 91255 **LC\_MESSAGES**  
 91256 Determine the locale that should be used to affect the format and contents of  
 91257 diagnostic messages written to standard error.
- 91258 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 91259 **ASYNCHRONOUS EVENTS**  
 91260 Default.
- 91261 **STDOUT**  
 91262 The list of object name information produced by the `-x` option shall be written to standard  
 91263 output in the following format:  
 91264 `"%s %d %s %s", <object-name>, <line-number>, <filename>, <text>`  
 91265 where `<text>` is the text of line `<line-number>` of file `<filename>`.
- 91266 **STDERR**  
 91267 The standard error shall be used only for diagnostic messages.
- 91268 **OUTPUT FILES**  
 91269 When the `-x` option is not specified, the format of the output file shall be:  
 91270 `"%s\t%s\t/%s/\n", <identifier>, <filename>, <pattern>`  
 91271 where `<pattern>` is a search pattern that could be used by an editor to find the defining instance  
 91272 of `<identifier>` in `<filename>` (where *defining instance* is indicated by the declarations listed in the  
 91273 EXTENDED DESCRIPTION).  
 91274 An optional `<circumflex>` (`'^'`) can be added as a prefix to `<pattern>`, and an optional `<dollar-`  
 91275 `sign>` can be appended to `<pattern>` to indicate that the pattern is anchored to the beginning  
 91276 (end) of a line of text. Any `<slash>` or `<backslash>` characters in `<pattern>` shall be preceded by a  
 91277 `<backslash>` character. The anchoring `<circumflex>`, `<dollar-sign>`, and escaping `<backslash>`  
 91278 characters shall not be considered part of the search pattern. All other characters in the search  
 91279 pattern shall be considered literal characters.  
 91280 An alternative format is:  
 91281 `"%s\t%s\t?%s?\n", <identifier>, <filename>, <pattern>`  
 91282 which is identical to the first format except that `<slash>` characters in `<pattern>` shall not be  
 91283 preceded by escaping `<backslash>` characters, and `<question-mark>` characters in `<pattern>`  
 91284 shall be preceded by `<backslash>` characters.  
 91285 A second alternative format is:  
 91286 `"%s\t%s\t%d\n", <identifier>, <filename>, <lineno>`  
 91287 where `<lineno>` is a decimal line number that could be used by an editor to find `<identifier>` in  
 91288 `<filename>`.  
 91289 Neither alternative format shall be produced by `ctags` when it is used as described by  
 91290 POSIX.1-2024, but the standard utilities that process tags files shall be able to process those  
 91291 formats as well as the first format.  
 91292 In any of these formats, the file shall be sorted by identifier, based on the collation sequence in

91293 the POSIX locale.

91294 **EXTENDED DESCRIPTION**

91295 The *ctags* utility shall attempt to produce an output line for each of the following objects:

- 91296 • Function definitions
- 91297 • Type definitions
- 91298 • Macros with arguments

91299 It may also produce output for any of the following objects:

- 91300 • Function prototypes
- 91301 • Structures
- 91302 • Unions
- 91303 • Global variable definitions
- 91304 • Enumeration types
- 91305 • Macros without arguments
- 91306 • **#define** statements
- 91307 • **#line** statements

91308 Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C  
91309 programs. The tag formed shall be created by prefixing **M** to the name of the file, with the  
91310 trailing *.c*, and leading pathname components (if any) removed.

91311 It is implementation-defined what other objects (including duplicate identifiers) produce output.

91312 On systems that do not support the C-Language Development Utilities option, if *ctags* is  
91313 supported it produces unspecified results for C-language source code files. It should write to  
91314 standard error a message identifying this condition and cause a non-zero exit status to be  
91315 produced.

91316 **EXIT STATUS**

91317 The following exit values shall be returned:

- 91318 0 Successful completion.
- 91319 >0 An error occurred.

91320 **CONSEQUENCES OF ERRORS**

91321 Default.

91322 **APPLICATION USAGE**

91323 The output with *-x* is meant to be a simple index that can be written out as an off-line readable  
91324 function index. If the input files to *ctags* (such as *.c* files) were not created using the same locale  
91325 as that in effect when *ctags -x* is run, results might not be as expected.

91326 The description of C-language processing says “attempts to” because the C language can be  
91327 greatly confused, especially through the use of **#defines**, and this utility would be of no use if  
91328 the real C preprocessor were run to identify them. The output from *ctags* may be fooled and  
91329 incorrect for various constructs.

91330 **EXAMPLES**

91331 None.

91332 **RATIONALE**

91333 The option list was significantly reduced from that provided by historical implementations. The  
 91334 `-F` option was omitted as redundant, since it is the default. The `-B` option was omitted as being  
 91335 of very limited usefulness. The `-t` option was omitted since the recognition of **typedefs** is now  
 91336 required for C source files. The `-u` option was omitted because the update function was judged  
 91337 to be not only inefficient, but also rarely needed.

91338 An early proposal included a `-w` option to suppress warning diagnostics. Since the types of such  
 91339 diagnostics could not be described, the option was omitted as being not useful.

91340 The text for `LC_CTYPE` about compatibility with the C locale acknowledges that the ISO C  
 91341 standard imposes requirements on the locale used to process C source. This could easily be a  
 91342 superset of that known as “the C locale” by way of implementation extensions, or one of a few  
 91343 alternative locales for systems supporting different codesets.

91344 The collation sequence of the tags file is not affected by `LC_COLLATE` because it is typically not  
 91345 used by human readers, but only by programs such as *vi* to locate the tag within the source files.  
 91346 Using the POSIX locale eliminates some of the problems of coordinating locales between the  
 91347 *ctags* file creator and the *vi* file reader.

91348 Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file  
 91349 has been published to encourage other programs to use the tags in new ways. The format allows  
 91350 either patterns or line numbers to find the identifiers because the historical *vi* recognizes either.  
 91351 The *ctags* utility does not produce the format using line numbers because it is not useful  
 91352 following any source file changes that add or delete lines. The documented search patterns  
 91353 match historical practice. It should be noted that literal leading `<circumflex>` or trailing `<dollar-`  
 91354 `sign>` characters in the search pattern will only behave correctly if anchored to the beginning of  
 91355 the line or end of the line by an additional `<circumflex>` or `<dollar-sign>` character.

91356 Historical implementations also understand the objects used by the languages FORTRAN,  
 91357 Pascal, and sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags*  
 91358 utility is not required to accommodate these languages, although implementors are encouraged  
 91359 to do so.

91360 The following historical option was not specified, as *vgrind* is not included in this volume of  
 91361 POSIX.1-2024:

91362 `-v` If the `-v` flag is given, an index of the form expected by *vgrind* is produced on the  
 91363 standard output. This listing contains the function name, filename, and page  
 91364 number (assuming 64-line pages). Since the output is sorted into lexicographic  
 91365 order, it may be desired to run the output through `sort -f`. Sample use:

```
91366     ctags -v files | sort -f > index
91367     vgrind -x index
```

91368 The special treatment of the tag **main** makes the use of *ctags* practical in directories with more  
 91369 than one program.

91370 **FUTURE DIRECTIONS**

91371 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 91372 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
 91373 format being used, implementations are encouraged to treat this as an error. A future version of  
 91374 this standard may require implementations to treat this as an error.

91375 If this utility is directed to create a new directory entry that contains any bytes that have the



91376 encoded value of a <newline> character, implementations are encouraged to treat this as an  
91377 error. A future version of this standard may require implementations to treat this as an error.

91378 **SEE ALSO**

91379 *c17, vi*

91380 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

91381 **CHANGE HISTORY**

91382 First released in Issue 4.

91383 **Issue 5**

91384 The FUTURE DIRECTIONS section is added.

91385 **Issue 6**

91386 This utility is marked as part of the User Portability Utilities option.

91387 The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

91388 The normative text is reworded to avoid use of the term “must” for application requirements.

91389 IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the  
91390 DESCRIPTION.

91391 **Issue 7**

91392 The *ctags* utility is no longer dependent on support for the User Portability Utilities option.

91393 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91394 **Issue 8**

91395 Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

91396 a. Report an error if a utility is directed to display a pathname that contains any bytes that  
91397 have the encoded value of a <newline> character when <newline> is a terminator or  
91398 separator in the output format being used.

91399 b. Disallow the creation of filenames containing any bytes that have the encoded value of a  
91400 <newline> character.

91401 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

91402 Austin Group Defect 1312 is applied, inserting a missing line break in the example commands in  
91403 the RATIONALE section.

91404 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

91405 **NAME**

91406 cut — cut out selected fields of each line of a file

91407 **SYNOPSIS**91408 cut -b *list* [-n] [*file...*]91409 cut -c *list* [*file...*]91410 cut -f *list* [-d *delim*] [-s] [*file...*]91411 **DESCRIPTION**

91412 The *cut* utility shall cut out bytes (**-b** option), characters (**-c** option), or character-delimited fields  
 91413 (**-f** option) from each line in one or more files, concatenate them, and write them to standard  
 91414 output.

91415 **OPTIONS**91416 The *cut* utility shall conform to XBD [Section 12.2](#) (on page 215).

91417 The application shall ensure that the option-argument *list* (see options **-b**, **-c**, and **-f** below) is a  
 91418 <comma>-separated list or <blank>-separated list of positive numbers and ranges. Ranges can  
 91419 be in three forms. The first is two positive numbers separated by a <hyphen-minus> (*low-high*),  
 91420 which represents all fields from the first number to the second number. The second is a positive  
 91421 number preceded by a <hyphen-minus> (*-high*), which represents all fields from field number 1  
 91422 to that number. The third is a positive number followed by a <hyphen-minus> (*low-*), which  
 91423 represents that number to the last field, inclusive. The elements in *list* can be repeated, can  
 91424 overlap, and can be specified in any order, but the bytes, characters, or fields selected shall be  
 91425 written in the order of the input data. If an element appears in the selection list more than once,  
 91426 it shall be written exactly once.

91427 The following options shall be supported:

91428 **-b list** Cut based on a *list* of bytes. Each selected byte shall be output unless the **-n** option  
 91429 is also specified. It shall not be an error to select bytes not present in the input line.

91430 **-c list** Cut based on a *list* of characters. Each selected character shall be output. It shall  
 91431 not be an error to select characters not present in the input line.

91432 **-d delim** Set the field delimiter to the character *delim*. The default is the <tab>.

91433 **-f list** Cut based on a *list* of fields, assumed to be separated in the file by a delimiter  
 91434 character (see **-d**). Each selected field shall be output. Output fields shall be  
 91435 separated by a single occurrence of the field delimiter character. Lines with no field  
 91436 delimiters shall be passed through intact, unless **-s** is specified. It shall not be an  
 91437 error to select fields not present in the input line.

91438 **-n** Do not split characters. When specified with the **-b** option, each element in *list* of  
 91439 the form *low-high* (<hyphen-minus>-separated numbers) shall be modified as  
 91440 follows:

- 91441 • If the byte selected by *low* is not the first byte of a character, *low* shall be  
 91442 decremented to select the first byte of the character originally selected by *low*.  
 91443 If the byte selected by *high* is not the last byte of a character, *high* shall be  
 91444 decremented to select the last byte of the character prior to the character  
 91445 originally selected by *high*, or zero if there is no prior character. If the  
 91446 resulting range element has *high* equal to zero or *low* greater than *high*, the list  
 91447 element shall be dropped from *list* for that input line without causing an  
 91448 error.

91449 Each element in *list* of the form *low-* shall be treated as above with *high* set to the

91450 number of bytes in the current line, not including the terminating <newline>. Each  
 91451 element in *list* of the form *-high* shall be treated as above with *low* set to 1. Each  
 91452 element in *list* of the form *num* (a single number) shall be treated as above with *low*  
 91453 set to *num* and *high* set to *num*.

91454 **-s** Suppress lines with no delimiter characters, when used with the *-f* option. Unless  
 91455 specified, lines with no delimiters shall be passed through untouched.

#### 91456 OPERANDS

91457 The following operand shall be supported:

91458 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is  
 91459 '-', the standard input shall be used.

#### 91460 STDIN

91461 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
 91462 See the INPUT FILES section.

#### 91463 INPUT FILES

91464 The input files shall be text files, except that line lengths shall be unlimited.

#### 91465 ENVIRONMENT VARIABLES

91466 The following environment variables shall affect the execution of *cut*:

91467 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 91468 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 91469 variables used to determine the values of locale categories.)

91470 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 91471 internationalization variables.

91472 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 91473 characters (for example, single-byte as opposed to multi-byte characters in  
 91474 arguments and input files).

#### 91475 *LC\_MESSAGES*

91476 Determine the locale that should be used to affect the format and contents of  
 91477 diagnostic messages written to standard error.

91478 *XSI* *NLSPATH* Determine the location of messages objects and message catalogs.

#### 91479 ASYNCHRONOUS EVENTS

91480 Default.

#### 91481 STDOUT

91482 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of  
 91483 the following):

91484 "%s\n", <concatenation of bytes>

91485 "%s\n", <concatenation of characters>

91486 "%s\n", <concatenation of fields and field delimiters>

#### 91487 STDERR

91488 The standard error shall be used only for diagnostic messages.

91489 **OUTPUT FILES**

91490 None.

91491 **EXTENDED DESCRIPTION**

91492 None.

91493 **EXIT STATUS**

91494 The following exit values shall be returned:

91495 0 All input files were output successfully.

91496 &gt;0 An error occurred.

91497 **CONSEQUENCES OF ERRORS**

91498 Default.

91499 **APPLICATION USAGE**

91500 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.  
 91501 The *cut* utility should be used when the number of lines (or records) needs to remain constant.  
 91502 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

91503 Earlier versions of the *cut* utility worked in an environment where bytes and characters were  
 91504 considered equivalent (modulo <backspace> and <tab> processing in some implementations). In  
 91505 the extended world of multi-byte characters, the new **-b** option has been added. The **-n** option  
 91506 (used with **-b**) allows it to be used to act on bytes rounded to character boundaries. The  
 91507 algorithm specified for **-n** guarantees that:

```
91508 cut -b 1-500 -n file > file1
91509 cut -b 501- -n file > file2
```

91510 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,  
 91511 however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

91512 **EXAMPLES**

91513 Examples of the option qualifier list:

91514 1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

91515 1-3,8 Equivalent to 1,2,3,8.

91516 -5,10 Equivalent to 1,2,3,4,5,10.

91517 3- Equivalent to third to last, inclusive.

91518 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte  
 91519 characters; see the description of **-n**.

91520 The following command:

91521 

```
cut -d : -f 1,6 /etc/passwd
```

91522 reads the System V password file (user database) and produces lines of the form:

91523 

```
<user ID>:<home directory>
```

91524 Most utilities in this volume of POSIX.1-2024 work on text files. The *cut* utility can be used to  
 91525 turn files with arbitrary line lengths into a set of text files containing the same data. The *paste*  
 91526 utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**  
 91527 contains long lines:

```
91528 cut -b 1-500 -n file > file1
91529 cut -b 501- -n file > file2
```

91530 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that  
91531 contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in  
91532 **file** that are longer than 500 + {LINE\_MAX} bytes.) The original file can be recreated from **file1**  
91533 and **file2** using the command:

```
91534 paste -d "\0" file1 file2 > file
```

#### 91535 RATIONALE

91536 Some historical implementations do not count <backspace> characters in determining character  
91537 counts with the `-c` option. This may be useful for using `cut` for processing `nroff` output. It was  
91538 deliberately decided not to have the `-c` option treat either <backspace> or <tab> characters in  
91539 any special fashion. The `fold` utility does treat these characters specially.

91540 Unlike other utilities, some historical implementations of `cut` exit after not finding an input file,  
91541 rather than continuing to process the remaining `file` operands. This behavior is prohibited by this  
91542 volume of POSIX.1-2024, where only the exit status is affected by this problem.

91543 The behavior of `cut` when provided with either mutually-exclusive options or options that do  
91544 not work logically together has been deliberately left unspecified in favor of global wording in  
91545 [Section 1.4](#) (on page 2462).

91546 The OPTIONS section was changed in response to IEEE PASC Interpretation 1003.2 #149. The  
91547 change represents historical practice on all known systems. The original standard was  
91548 ambiguous on the nature of the output.

91549 The `list` option-arguments are historically used to select the portions of the line to be written, but  
91550 do not affect the order of the data. For example:

```
91551 echo abcdefghi | cut -c6,2,4-7,1
```

91552 yields "abdefg".

91553 A proposal to enhance `cut` with the following option:

91554 `-o` Preserve the selected field order. When this option is specified, each byte, character, or field  
91555 (or ranges of such) shall be written in the order specified by the `list` option-argument, even if  
91556 this requires multiple outputs of the same bytes, characters, or fields.

91557 was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft  
91558 standard.

#### 91559 FUTURE DIRECTIONS

91560 None.

#### 91561 SEE ALSO

91562 [Section 2.5](#) (on page 2478), [fold](#), [grep](#), [paste](#)

91563 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 91564 CHANGE HISTORY

91565 First released in Issue 2.

#### 91566 Issue 6

91567 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

91568 The normative text is reworded to avoid use of the term “must” for application requirements.

91569 **Issue 7**

91570 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91571 SD5-XCU-ERN-171 is applied, adding APPLICATION USAGE.

91572 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0080 [584] is applied.

91573 **Issue 8**

91574 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

91575 **NAME**91576 cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)91577 **SYNOPSIS**

```
91578 XSI  cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]...
91579      [-U name]... file...
```

91580 **DESCRIPTION**

91581 The *cxref* utility shall analyze a collection of C-language *files* and attempt to build a cross-  
 91582 reference table. Information from **#define** lines shall be included in the symbol table. A sorted  
 91583 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*  
 91584 separately, or with the **-c** option, in combination. Each symbol shall contain an <asterisk> before  
 91585 the declaring reference.

91586 **OPTIONS**

91587 The *cxref* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the **-D**,  
 91588 **-I**, and **-U** options (which are identical to their interpretation by *c17*) is significant. The  
 91589 following options shall be supported:

- 91590 **-c** Write a combined cross-reference of all input files.
- 91591 **-s** Operate silently; do not print input filenames.
- 91592 **-o file** Direct output to named *file*.
- 91593 **-w num** Format output no wider than *num* (decimal) columns. This option defaults to 80 if  
 91594 *num* is not specified or is less than 51.
- 91595 **-D** Equivalent to *c17*.
- 91596 **-I** Equivalent to *c17*.
- 91597 **-U** Equivalent to *c17*.

91598 **OPERANDS**

91599 The following operand shall be supported:

- 91600 *file* A pathname of a C-language source file.

91601 **STDIN**

91602 Not used.

91603 **INPUT FILES**

91604 The input files are C-language source files.

91605 **ENVIRONMENT VARIABLES**

91606 The following environment variables shall affect the execution of *cxref*:

- 91607 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 91608 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 91609 variables used to determine the values of locale categories.)
- 91610 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 91611 internationalization variables.
- 91612 **LC\_COLLATE**  
 91613 Determine the locale for the ordering of the output.
- 91614 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 91615 characters (for example, single-byte as opposed to multi-byte characters in  
 91616 arguments and input files).

- 91617 *LC\_MESSAGES*
- 91618 Determine the locale that should be used to affect the format and contents of
- 91619 diagnostic messages written to standard error.
- 91620 *NLSPATH* Determine the location of messages objects and message catalogs.
- 91621 **ASYNCHRONOUS EVENTS**
- 91622 Default.
- 91623 **STDOUT**
- 91624 The standard output shall be used for the cross-reference listing, unless the `-o` option is used to
- 91625 select a different output file.
- 91626 The format of standard output is unspecified, except that the following information shall be
- 91627 included:
- 91628 • If the `-c` option is not specified, each portion of the listing shall start with the name of the
  - 91629 input file on a separate line.
  - 91630 • The name line shall be followed by a sorted list of symbols, each with its associated
  - 91631 location pathname, the name of the function in which it appears (if it is not a function
  - 91632 name itself), and line number references.
  - 91633 • Each line number may be preceded by an <asterisk> ('\*') flag, meaning that this is the
  - 91634 declaring reference. Other single-character flags, with implementation-defined meanings,
  - 91635 may be included.
- 91636 **STDERR**
- 91637 The standard error shall be used only for diagnostic messages.
- 91638 **OUTPUT FILES**
- 91639 The output file named by the `-o` option shall be used instead of standard output.
- 91640 **EXTENDED DESCRIPTION**
- 91641 None.
- 91642 **EXIT STATUS**
- 91643 The following exit values shall be returned:
- 91644 0 Successful completion.
  - 91645 >0 An error occurred.
- 91646 **CONSEQUENCES OF ERRORS**
- 91647 Default.
- 91648 **APPLICATION USAGE**
- 91649 None.
- 91650 **EXAMPLES**
- 91651 None.
- 91652 **RATIONALE**
- 91653 None.
- 91654 **FUTURE DIRECTIONS**
- 91655 If this utility is directed to display a pathname that contains any bytes that have the encoded
- 91656 value of a <newline> character when <newline> is a terminator or separator in the output
- 91657 format being used, implementations are encouraged to treat this as an error. A future version of
- 91658 this standard may require implementations to treat this as an error.



91659 **SEE ALSO**91660 [c17](#)91661 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)91662 **CHANGE HISTORY**

91663 First released in Issue 2.

91664 **Issue 5**91665 In the SYNOPSIS, `[-U dir]` is changed to `[-U name]`.91666 **Issue 6**

91667 The APPLICATION USAGE section is added.

91668 **Issue 7**

91669 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91670 **Issue 8**91671 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
91672 directed to display a pathname that contains any bytes that have the encoded value of a  
91673 `<newline>` character when `<newline>` is a terminator or separator in the output format being  
91674 used.91675 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

91676 **NAME**

91677 date — write the date and time

91678 **SYNOPSIS**

91679 date [-u] [+format]

91680 XSI date [-u] *mmddhhmm* [[*cc*]*yy*]91681 **DESCRIPTION**

91682 XSI The *date* utility shall write the date and time to standard output or attempt to set the system  
 91683 date and time. By default, the current date and time shall be written. If an operand beginning  
 91684 with '+' is specified, the output format of *date* shall be controlled by the conversion  
 91685 specifications and other text in the operand.

91686 **OPTIONS**91687 The *date* utility shall conform to XBD Section 12.2 (on page 215).

91688 The following option shall be supported:

91689 **-u** Perform operations as if the *TZ* environment variable was set to the string "UTC0",  
 91690 or its equivalent historical value of "GMT0". Otherwise, *date* shall use the timezone  
 91691 indicated by the *TZ* environment variable or the system default if that variable is  
 91692 unset or null.

91693 **OPERANDS**

91694 The following operands shall be supported:

91695 **+format** When the format is specified, the output shall be formatted as if by *strftime()* with  
 91696 the specified format string, and a *timeptr* argument that is the equivalent of  
 91697 *localtime(&now)* if **-u** is not specified or *gmtime(&now)* if **-u** is specified, where *now*  
 91698 is an object of type **time\_t** containing the return value of *time(0)*.

91699 A <newline> shall always be appended to the output of *strftime()*.91700 XSI *mmddhhmm*[[*cc*]*yy*]

91701 Attempt to set the system date and time from the value given in the operand. This  
 91702 is only possible if the user has appropriate privileges and the system permits the  
 91703 setting of the system date and time. The first *mm* is the month (number); *dd* is the  
 91704 day (number); *hh* is the hour (number, 24-hour system); the second *mm* is the  
 91705 minute (number); *cc* is the century and is the first two digits of the year (this is  
 91706 optional); *yy* is the last two digits of the year and is optional. If century is not  
 91707 specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive,  
 91708 and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The  
 91709 current year is the default if *yy* is omitted.

91710 **Note:** It is expected that in a future version of this standard the default century inferred  
 91711 from a 2-digit year will change. (This would apply to all commands accepting a  
 91712 2-digit year as input.)

91713 **STDIN**

91714 Not used.

91715 **INPUT FILES**

91716 None.

91717 **ENVIRONMENT VARIABLES**91718 The following environment variables shall affect the execution of *date*:

91719 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 91720 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 91721 variables used to determine the values of locale categories.)

91722 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 91723 internationalization variables.

91724 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 91725 characters (for example, single-byte as opposed to multi-byte characters in  
 91726 arguments).

91727 *LC\_MESSAGES*

91728 Determine the locale that should be used to affect the format and contents of  
 91729 diagnostic messages written to standard error.

91730 *LC\_TIME* Determine the format and contents of date and time strings written by *date*.

91731 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

91732 *TZ* Determine the timezone in which the time and date are written, unless the *-u*  
 91733 option is specified. If the *TZ* variable is unset or null and *-u* is not specified, an  
 91734 unspecified system default timezone is used.

91735 **ASYNCHRONOUS EVENTS**

91736 Default.

91737 **STDOUT**

91738 When no formatting operand is specified, the output in the POSIX locale shall be equivalent to  
 91739 specifying:

91740 `date "+%a %b %e %H:%M:%S %Z %Y"`91741 **STDERR**

91742 The standard error shall be used only for diagnostic messages.

91743 **OUTPUT FILES**

91744 None.

91745 **EXTENDED DESCRIPTION**

91746 None.

91747 **EXIT STATUS**

91748 The following exit values shall be returned:

91749 0 The date was written successfully.

91750 &gt;0 An error occurred.

91751 **CONSEQUENCES OF ERRORS**

91752 Default.

## 91753 APPLICATION USAGE

91754 Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can  
 91755 contain <newline> characters in some locales, so it may be difficult to use the format shown in  
 91756 standard output for parsing the output of *date* in those locales.

91757 Since the default *date* utility format for locales other than the POSIX or C locale is not required to  
 91758 include anything beyond the date and time, whereas for the POSIX or C locale it also includes  
 91759 the day name and time zone, it may be necessary to specify a format (or override the locale-  
 91760 selection environment variables) to ensure this information is included when desired.

91761 The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap  
 91762 second.

91763 Although certain of the conversion specifiers in the POSIX locale (such as the name of the  
 91764 month) are shown with initial capital letters, this need not be the case in other locales. Programs  
 91765 using these fields may need to adjust the capitalization if the output is going to be used at the  
 91766 beginning of a sentence.

91767 The date string formatting capabilities are intended for use in Gregorian-style calendars,  
 91768 possibly with a different starting year (or years). The %x and %c conversion specifications,  
 91769 however, are intended for local representation; these may be based on a different, non-Gregorian  
 91770 calendar.

91771 The %C conversion specification was introduced to allow a fallback for the %EC (alternative year  
 91772 format base year); it can be viewed as the base of the current subdivision in the Gregorian  
 91773 calendar. The century number is calculated as the year divided by 100 and truncated to an  
 91774 integer; it should not be confused with the use of ordinal numbers for centuries (for example,  
 91775 ``twenty-first century``.) Both the %Ey and %y can then be viewed as the offset from %EC and %C,  
 91776 respectively.

91777 The E and O modifiers modify the traditional conversion specifiers, so that they can always be  
 91778 used, even if the implementation (or the current locale) does not support the modifier.

91779 The E modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as  
 91780 these are based on the Gregorian calendar system. Extending the E modifiers to other date  
 91781 elements may provide an implementation-defined extension capable of supporting other  
 91782 calendar systems, especially in combination with the O modifier.

91783 The O modifier supports time and date formats using the locale's alternative numerical symbols,  
 91784 such as Kanji or Hindi digits or ordinal number representation.

91785 Non-European locales, whether they use Latin digits in computational items or not, often have  
 91786 local forms of the digits for use in date formats. This is not totally unknown even in Europe; a  
 91787 variant of dates uses Roman numerals for the months: the third day of September 1991 would be  
 91788 written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking  
 91789 countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %y conversion  
 91790 specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O  
 91791 modifier was introduced to support the use for display purposes of non-Latin digits. In the  
 91792 *LC\_TIME* category in *localedef*, the optional **alt\_digits** keyword is intended for this purpose. As  
 91793 an example, assume the following (partial) *localedef* source:

```
91794 alt_digits  " "; "I"; "II"; "III"; "IV"; "V"; "VI"; "VII"; "VIII" \  

  91795            "IX"; "X"; "XI"; "XII"  

  91796 d_fmt      "%e.%Om.%Y"
```

91797 With the above date, the command:

```
91798 date "+%x"
```

91799 would yield 3.IX.1991. With the same `d_fmt`, but without the `alt_digits`, the command would  
 91800 yield 3.9.1991.

## 91801 EXAMPLES

91802 1. The following are input/output examples of `date` used at arbitrary times in the POSIX  
 91803 locale:

```
91804 $ date
```

```
91805 Tue Jun 26 09:58:10 PDT 1990
```

```
91806 $ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
```

```
91807 DATE: 11/02/91
```

```
91808 TIME: 13:36:16
```

```
91809 $ date "+TIME: %r"
```

```
91810 TIME: 01:36:32 PM
```

91811 2. Examples for Denmark, where the default date and time format is `%a %d %b %Y %T %Z`:

```
91812 $ LANG=da_DK.iso_8859-1 date
```

```
91813 ons 02 okt 1991 15:03:32 CET
```

```
91814 $ LANG=da_DK.iso_8859-1 \
```

```
91815 date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
```

```
91816 DATO: onsdag den 2. oktober 1991
```

```
91817 KLOKKEN: 15:03:56
```

91818 3. Examples for Germany, where the default date and time format is `%a %d.%h.%Y, %T %Z`:

```
91819 $ LANG=De_DE.88591 date
```

```
91820 Mi 02.Okt.1991, 15:01:21 MEZ
```

```
91821 $ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
```

```
91822 DATUM: Mittwoch, 02. Oktober 1991
```

```
91823 ZEIT: 15:02:02
```

91824 4. Examples for France, where the default date and time format is `%a %d %h %Y %Z %T`:

```
91825 $ LANG=Fr_FR.88591 date
```

```
91826 Mer 02 oct 1991 MET 15:03:32
```

```
91827 $ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
```

```
91828 JOUR: Mercredi 02 octobre 1991
```

```
91829 HEURE: 15:03:56
```

## 91830 RATIONALE

91831 Some of the new options for formatting are from the ISO C standard. The `-u` option was  
 91832 introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is  
 91833 allowed as an equivalent `TZ` value to be compatible with all of the systems using the BSD  
 91834 implementation, where this option originated.

91835 The `%e` format conversion specification (adopted from System V) was added because the ISO C  
 91836 standard conversion specifications did not provide any way to produce the historical default  
 91837 `date` output during the first nine days of any month.

91838 There are two varieties of day and week numbering supported (in addition to any others created  
 91839 with the locale-dependent `%E` and `%O` modifier characters):

91840 • The historical variety in which Sunday is the first day of the week and the weekdays  
 91841 preceding the first Sunday of the year are considered week 0. These are represented by %w  
 91842 and %U. A variant of this is %W, using Monday as the first day of the week, but still  
 91843 referring to week 0. This view of the calendar was retained because so many historical  
 91844 applications depend on it and the ISO C standard *strftime()* function, on which many *date*  
 91845 implementations are based, was defined in this way.

91846 • The international standard, based on the ISO 8601-1:2019 standard where Monday is the  
 91847 first weekday and the algorithm for the first week number is more complex: If the week  
 91848 (Monday to Sunday) containing January 1 has four or more days in the new year, then it is  
 91849 week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. These  
 91850 are represented by the new conversion specifications %u and %v, added as a result of  
 91851 international comments.

91852 Although this standard only requires the default *date* utility format, for locales other than the  
 91853 POSIX or C locale, to include the date and time, it is common for implementations to include  
 91854 day name and time zone information as well. (For the POSIX locale this is required, with the day  
 91855 name in %a format at the beginning and the time zone in %Z format before the year.)  
 91856 Implementations are encouraged to include the day name (in %a or %A format) and the time  
 91857 zone (in %Z or %z format) in the default *date* utility format for all of the locales they provide.

91858 Some implementations have a **date\_fmt** locale keyword (see [Section 7.3.5](#), on page 152) as an  
 91859 extension, to specify the default *date* utility format for each locale. On such implementations, if  
 91860 the *localedef* utility is used to create a locale that does not have this information, the *date* utility  
 91861 must by default still produce output for that locale that includes both the time and the date.

#### 91862 **FUTURE DIRECTIONS**

91863 None.

#### 91864 **SEE ALSO**

91865 XBD [Section 7.3.5](#) (on page 152), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

91866 XSH [fprintf\(\)](#), [strftime\(\)](#)

#### 91867 **CHANGE HISTORY**

91868 First released in Issue 2.

#### 91869 **Issue 5**

91870 Changes are made for Year 2000 alignment.

#### 91871 **Issue 6**

91872 The following new requirements on POSIX implementations derive from alignment with the  
 91873 Single UNIX Specification:

- 91874 • The %EX modified conversion specification is added.

91875 The Open Group Corrigendum U048/2 is applied, correcting the examples.

91876 The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors  
 91877 for consistency with the *LC\_TIME* category.

91878 A clarification is made such that the current year is the default if the *yy* argument is omitted  
 91879 when setting the system date and time.

91880 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/19 is applied, correcting the CHANGE  
 91881 HISTORY section.

91882 **Issue 8**

91883 Austin Group Defect 466 is applied, replacing the list of conversion specifications for the *+format*  
91884 operand with a requirement that the output is formatted as if by a call to *strftime()* with specific  
91885 arguments.

91886 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

91887 Austin Group Defect 1345 is applied, adding paragraphs to APPLICATION USAGE and  
91888 RATIONALE about the default *date* utility format.

91889 **NAME**

91890 dd — convert and copy a file

91891 **SYNOPSIS**91892 dd [*operand...*]91893 **DESCRIPTION**

91894 The *dd* utility shall copy the specified input file to the specified output file with possible  
91895 conversions using specific input and output block sizes. It shall read the input one block at a  
91896 time, using the specified input block size; it shall then process the block of data actually  
91897 returned, which could be smaller than the requested block size. It shall apply any conversions  
91898 that have been specified and write the resulting data to the output in blocks of the specified  
91899 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,  
91900 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a  
91901 separate output block; if the read returns less than a full block and the **sync** conversion is not  
91902 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**  
91903 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the  
91904 input shall be processed and collected into full-sized output blocks until the end of the input is  
91905 reached.

91906 The processing order shall be as follows:

- 91907 1. An input block is read. If the **iflags=fullblock** operand is specified, this might entail  
91908 multiple reads; otherwise, the input block shall be used even if the read was shorter than  
91909 the specified block size.
- 91910 2. If the input block is shorter than the specified input block size and the **sync** conversion is  
91911 specified, null bytes shall be appended to the input data up to the specified size. (If either  
91912 **block** or **unblock** is also specified, <space> characters shall be appended instead of null  
91913 bytes.) The remaining conversions and output shall include the pad characters as if they  
91914 had been read from the input.
- 91915 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is  
91916 requested, the resulting data shall be written to the output as a single block, and the  
91917 remaining steps are omitted.
- 91918 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If  
91919 there is an odd number of bytes in the input block, the last byte in the input record shall  
91920 not be swapped.
- 91921 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These  
91922 conversions shall operate on the input data independently of the input blocking; an input  
91923 or output fixed-length record may span block boundaries.
- 91924 6. The data resulting from input or conversion or both shall be aggregated into output  
91925 blocks of the specified size. After the end of input is reached, any remaining output shall  
91926 be written as a block without padding if **conv=sync** is not specified; thus, the final output  
91927 block may be shorter than the output block size.

91928 **OPTIONS**

91929 None.

91930 **OPERANDS**

91931 All of the operands shall be processed before any input is read. The following operands shall be  
91932 supported:



91933	<b>if=file</b>	Specify the input pathname; the default is standard input.
91934	<b>of=file</b>	Specify the output pathname; the default is standard output. If the <b>seek=expr</b>
91935		conversion is not also specified, the output file shall be truncated before the copy
91936		begins if an explicit <b>of=file</b> operand is specified, unless <b>conv=notrunc</b> is specified.
91937		If <b>seek=expr</b> is specified, but <b>conv=notrunc</b> is not, the effect of the copy shall be to
91938		preserve the blocks in the output file over which <i>dd</i> seeks, but no other portion of
91939		the output file shall be preserved. (If the size of the seek plus the size of the input
91940		file is less than the previous size of the output file, the output file shall be
91941		shortened by the copy. If the input file is empty and either the size of the seek is
91942		greater than the previous size of the output file or the output file did not
91943		previously exist, the size of the output file shall be set to the file offset after the
91944		seek.)
91945	<b>ibs=expr</b>	Specify the input block size, in bytes, by <i>expr</i> (default is 512).
91946	<b>obs=expr</b>	Specify the output block size, in bytes, by <i>expr</i> (default is 512).
91947	<b>bs=expr</b>	Set both input and output block sizes to <i>expr</i> bytes, superseding <b>ibs=</b> and <b>obs=</b> . If
91948		no conversion other than <b>sync</b> , <b>noerror</b> , and <b>notrunc</b> is specified, each input block
91949		shall be copied to the output as a single block without aggregating short blocks.
91950	<b>cbs=expr</b>	Specify the conversion block size for <b>block</b> and <b>unblock</b> in bytes by <i>expr</i> (default is
91951		zero). If <b>cbs=</b> is omitted or given a value of zero, using <b>block</b> or <b>unblock</b> produces
91952		unspecified results.
91953	XSI	The application shall ensure that this operand is also specified if the <b>conv=</b>
91954		operand is specified with a value of <b>ascii</b> , <b>ebcdic</b> , or <b>ibm</b> . For a <b>conv=</b> operand
91955		with an <b>ascii</b> value, the input is handled as described for the <b>unblock</b> value, except
91956		that characters are converted to ASCII before any trailing <space> characters are
91957		deleted. For <b>conv=</b> operands with <b>ebcdic</b> or <b>ibm</b> values, the input is handled as
91958		described for the <b>block</b> value except that the characters are converted to EBCDIC
91959		or IBM EBCDIC, respectively, after any trailing <space> characters are added.
91960	<b>skip=n</b>	Skip <i>n</i> input blocks (using the specified input block size) before starting to copy.
91961		On seekable files, the implementation shall read the blocks or seek past them; on
91962		non-seekable files, the blocks shall be read and the data shall be discarded.
91963	<b>seek=n</b>	Skip <i>n</i> blocks (using the specified output block size) from the beginning of the
91964		output file before copying. On non-seekable files, existing blocks shall be read and
91965		space from the current end-of-file to the specified offset, if any, filled with null
91966		bytes; on seekable files, the implementation shall seek to the specified offset or
91967		read the blocks as described for non-seekable files.
91968	<b>count=n</b>	Copy only <i>n</i> input blocks. If <i>n</i> is zero, it is unspecified whether no blocks or all
91969		blocks are copied.
91970	<b>conv=value[,value ...]</b>	
91971		Where <i>values</i> are <comma>-separated symbols from the following list:
91972	XSI	<b>ascii</b> Convert EBCDIC to ASCII; see <a href="#">Table 3-8</a> (on page 2780).
91973	XSI	<b>ebcdic</b> Convert ASCII to EBCDIC; see <a href="#">Table 3-8</a> (on page 2780).
91974	XSI	<b>ibm</b> Convert ASCII to a different EBCDIC set; see <a href="#">Table 3-9</a> (on page 2781).
91975	XSI	The <b>ascii</b> , <b>ebcdic</b> , and <b>ibm</b> values are mutually-exclusive.

91976	<b>block</b>	Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space> characters shall be appended to lines that are shorter than their conversion block size to fill the block. Lines that are longer than the conversion block size shall be truncated to the largest number of characters that fit into that size; the number of truncated lines shall be reported (see the STDERR section).
91977		
91978		
91979		
91980		
91981		
91982		
91983		
91984		
91985		The <b>block</b> and <b>unblock</b> values are mutually-exclusive.
91986	<b>unblock</b>	Convert fixed-length records to variable length. Read a number of bytes equal to the conversion block size (or the number of bytes remaining in the input, if less than the conversion block size), delete all trailing <space> characters, and append a <newline>.
91987		
91988		
91989		
91990	<b>lcase</b>	Map uppercase characters specified by the <i>LC_CTYPE</i> keyword <b>tolower</b> to the corresponding lowercase character. Characters for which no mapping is specified shall not be modified by this conversion.
91991		
91992		
91993		The <b>lcase</b> and <b>ucase</b> symbols are mutually-exclusive.
91994	<b>ucase</b>	Map lowercase characters specified by the <i>LC_CTYPE</i> keyword <b>toupper</b> to the corresponding uppercase character. Characters for which no mapping is specified shall not be modified by this conversion.
91995		
91996		
91997	<b>swab</b>	Swap every pair of input bytes.
91998	<b>noerror</b>	Do not stop processing on an input error. When an input error occurs, a diagnostic message shall be written on standard error, followed by the current input and output block counts in the same format as used at completion (see the STDERR section). If the <b>sync</b> conversion is specified, the missing input shall be replaced with null bytes and processed normally; otherwise, the input block shall be omitted from the output.
91999		
92000		
92001		
92002		
92003		
92004		
92005	<b>notrunc</b>	Do not truncate the output file. Preserve blocks in the output file not explicitly written by this invocation of the <i>dd</i> utility. (See also the preceding <b>of=file</b> operand.)
92006		
92007		
92008	<b>sync</b>	Pad every input block to the size of the <b>ibs=</b> buffer, appending null bytes. (If either <b>block</b> or <b>unblock</b> is also specified, append <space> characters, rather than null bytes.)
92009		
92010		
92011	<b>iflags=fullblock</b>	
92012		Perform as many reads as required to reach the full input block size or end of file, rather than acting on partial reads. If this operand is in effect, then the <b>count=</b> operand refers to the number of full input blocks rather than reads. The behavior is unspecified if <b>iflags=fullblock</b> is requested alongside the <b>sync</b> , <b>block</b> , or <b>unblock</b> conversions.
92013		
92014		
92015		
92016		
92017		The behavior is unspecified if operands other than <b>conv=</b> are specified more than once.
92018		For the <b>bs=</b> , <b>cbs=</b> , <b>ibs=</b> , and <b>obs=</b> operands, the application shall supply an expression specifying a size in bytes. The expression, <i>expr</i> , can be:
92019		

- 92020 1. A positive decimal number
- 92021 2. A positive decimal number followed by  $k$ , specifying multiplication by 1 024
- 92022 3. A positive decimal number followed by  $b$ , specifying multiplication by 512
- 92023 4. Two or more positive decimal numbers (with or without  $k$  or  $b$ ) separated by  $x$ , specifying
- 92024 the product of the indicated values

92025 All of the operands are processed before any input is read.

92026 XSI The following two tables display the octal number character values used for the **ascii** and **ebcdic**

92027 conversions (first table) and for the **ibm** conversion (second table). In both tables, the ASCII

92028 values are the row and column headers and the EBCDIC values are found at their intersections.

92029 For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The

92030 inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one

92031 correspondence with these tables. The differences between the two tables are highlighted by

92032 small boxes drawn around five entries.

Table 3-8 ASCII to EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 \$	0154 %	0120 &	0175 .
0050	0115 (	0135 )	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [	0340 \	0275 ]	0232	0155 -
0140	0171 ,	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117	0320 }	0137 ~	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0152 i	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0112 ¢	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0241	0276	0277
0350	0312	0313	0314 J	0315	0316 y	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 H	0355
0370	0356	0357	0372	0373	0374	0375	0376	0377 EO

Table 3-9 ASCII to IBM EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 \$	0154 %	0120 &	0175 .
0050	0115 (	0135 )	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [	0340 \	0275 ]	0137 ^	0155 _
0140	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117	0320 }	0241 ~	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0232	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0255 [	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0275 ]	0276	0277
0350	0312	0313	0314 J	0315	0316 y	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 H	0355
0370	0356	0357	0372	0373	0374	0375	0376	0377 EO

92035 **STDIN**

92036 If no **if=** operand is specified, the standard input shall be used. See the INPUT FILES section.

92037 **INPUT FILES**

92038 The input file can be any file type.

92039 **ENVIRONMENT VARIABLES**

92040 The following environment variables shall affect the execution of *dd*:

92041 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 92042 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 92043 variables used to determine the values of locale categories.)

92044 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 92045 internationalization variables.

92046 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 92047 characters (for example, single-byte as opposed to multi-byte characters in  
 92048 arguments and input files), the classification of characters as uppercase or  
 92049 lowercase, and the mapping of characters from one case to the other.

92050 **LC\_MESSAGES**

92051 Determine the locale that should be used to affect the format and contents of  
 92052 diagnostic messages written to standard error and informative messages written to  
 92053 standard output.

92054 XSI **NLSPATH** Determine the location of messages objects and message catalogs.

92055 **ASYNCHRONOUS EVENTS**

92056 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to  
 92057 standard error, and terminate abnormally as if by the default action for SIGINT. One or more  
 92058 implementation defined non-job-control signals other than SIGABRT, SIGHUP, and SIGTERM  
 92059 may write status information to standard error and continue processing. All other signals  
 92060 (including job control signals, SIGABRT, SIGHUP, and SIGTERM) shall take their default action;  
 92061 see the ASYNCHRONOUS EVENTS section in Section 1.4 (on page 2462).

92062 **STDOUT**

92063 If no **of=** operand is specified, the standard output shall be used. The nature of the output  
 92064 depends on the operands selected.

92065 **STDERR**

92066 On completion, *dd* shall write the number of input and output blocks to standard error. In the  
 92067 POSIX locale the following formats shall be used:

92068 "%u+%u records in\n", <number of whole input blocks>,  
 92069 <number of partial input blocks>

92070 "%u+%u records out\n", <number of whole output blocks>,  
 92071 <number of partial output blocks>

92072 A partial input block is one for which *read()* returned less than the input block size. A partial  
 92073 output block is one that was written with fewer bytes than specified by the output block size.

92074 In addition, when there is at least one truncated block, the number of truncated blocks shall be  
 92075 written to standard error. In the POSIX locale, the format shall be:

92076 "%u truncated %s\n", <number of truncated blocks>, "record" (if  
 92077 <number of truncated blocks> is one) "records" (otherwise)

92078 Diagnostic messages may also be written to standard error.

92079 **OUTPUT FILES**

92080 If the **of=** operand is used, the output shall be the same as described in the STDOUT section.

92081 **EXTENDED DESCRIPTION**

92082 None.

92083 **EXIT STATUS**

92084 The following exit values shall be returned:

92085 0 Successful completion.

92086 >0 An error occurred.

92087 **CONSEQUENCES OF ERRORS**

92088 If an input error is detected and the **noerror** conversion has not been specified, any partial  
92089 output block shall be written to the output file, a diagnostic message shall be written, and the  
92090 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall  
92091 be written and the copy operation shall be discontinued.

92092 **APPLICATION USAGE**

92093 The input and output block size can be specified to take advantage of raw physical I/O.

92094 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions  
92095 specified for the *dd* utility perform conversions for the version specified by the tables.

92096 Using the **count=** operand of *dd* with a pipe or FIFO as the input can lead to surprising results,  
92097 since these file types are prone to encountering short reads for any input block size other than 1.  
92098 Unless the **iflags=fullblock** operand is in effect, *dd* will stop after the specified number of reads,  
92099 rather than full input blocks, and therefore can often result in fewer bytes being output than the  
92100 product of the count and input block size.

92101 **EXAMPLES**

92102 The following command:

```
92103 dd if=/dev/rmt0h of=/dev/rmt1h
```

92104 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

92105 The following command:

```
92106 dd ibs=10 skip=1
```

92107 strips the first 10 bytes from standard input.

92108 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the  
92109 ASCII file *x*:

```
92110 dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

92111 **RATIONALE**

92112 The OPTIONS section is listed as “None” because there are no options recognized by historical  
92113 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax  
92114 Guidelines, which would have resulted in the classic hyphenated option letters. In this version  
92115 of this volume of POSIX.1-2024, *dd* retains its curious JCL-like syntax due to the large number of  
92116 applications that depend on the historical implementation.

92117 A suggested implementation technique for **conv=noerror, sync** is to zero (or <space>-fill, if  
92118 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input  
92119 buffer to the output even after an error. In this manner, any data transferred to the input buffer  
92120 before the error was detected is preserved. Another point is that a failed read on a regular file or  
92121 a disk generally does not increment the file offset, and *dd* must then seek past the block on which

92122 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic  
 92123 tape, however, the tape normally has passed the block containing the error when the error is  
 92124 reported, and thus no seek is necessary.

92125 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely  
 92126 portable) scripts that assume these values. If they were left unspecified, unusual results could  
 92127 occur if an implementation chose an odd block size.

92128 Historical implementations of *dd* used *creat()* when processing **of=file**. This makes the **seek=**  
 92129 operand unusable except on special files. The **conv=notrunc** feature was added because more  
 92130 recent BSD-based implementations use *open()* (without `O_TRUNC`) instead of *creat()*, but they  
 92131 fail to delete output file contents after the data copied.

92132 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to  
 92133 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of  
 92134 POSIX.1-2024.

92135 Standard EBCDIC does not have the characters '[' and ']'. The values used in the table are  
 92136 taken from a common print train that does contain them. Other than those characters, the print  
 92137 train values are not filled in, but appear to provide some of the motivation for the historical  
 92138 choice of translations reflected here.

92139 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.

92140 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ  
 92141 in such a way that:

- 92142 1. EBCDIC 0112 ('ϕ') and 0152 (broken pipe) do not appear in the table.
- 92143 2. EBCDIC 0137 ('↵') translates to/from ASCII 0236 ('^'). In the standard table, EBCDIC  
 92144 0232 (no graphic) is used.
- 92145 3. EBCDIC 0241 ('~') translates to/from ASCII 0176 ('~'). In the standard table, EBCDIC  
 92146 0137 ('↵') is used.
- 92147 4. 0255 ('[') and 0275 (']') appear twice, once in the same place as for the standard table  
 92148 and once in place of 0112 ('ϕ') and 0241 ('~').

92149 In net result:

92150 EBCDIC 0275 (']') displaced EBCDIC 0241 ('~') in cell 0345.

92151 That displaced EBCDIC 0137 ('↵') in cell 0176.

92152 That displaced EBCDIC 0232 (no graphic) in cell 0136.

92153 That replaced EBCDIC 0152 (broken pipe) in cell 0313.

92154 EBCDIC 0255 ('[') replaced EBCDIC 0112 ('ϕ').

92155 This translation, however, reflects historical practice that (ASCII) '~' and '↵' were often  
 92156 mapped to each other, as were '[' and 'ϕ'; and ']' and (EBCDIC) '~'.

92157 The **chs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the  
 92158 **ascii** operand, the input is handled as described for the **unblock** operand except that characters  
 92159 are converted to ASCII before the trailing <space> characters are deleted. For the **ebcdic** and  
 92160 **ibm** operands, the input is handled as described for the **block** operand except that the characters  
 92161 are converted to EBCDIC or IBM EBCDIC after the trailing <space> characters are added.

92162 The **block** and **unblock** keywords are from historical BSD practice.



- 92163 The consistent use of the word **record** in standard error messages matches most historical  
92164 practice. An earlier version of System V used **block**, but this has been updated in more recent  
92165 releases.
- 92166 Early proposals only allowed two numbers separated by **x** to be used in a product when  
92167 specifying **bs=**, **cbs=**, **ibs=**, and **obs=** sizes. This was changed to reflect the historical practice of  
92168 allowing multiple numbers in the product as provided by Version 7 and all releases of System V  
92169 and BSD.
- 92170 A change to the **swab** conversion is required to match historical practice and is the result of IEEE  
92171 PASC Interpretations 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.
- 92172 A change to the handling of SIGINT is required to match historical practice and is the result of  
92173 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.
- 92174 **FUTURE DIRECTIONS**
- 92175 If this utility is directed to create a new directory entry that contains any bytes that have the  
92176 encoded value of a <newline> character, implementations are encouraged to treat this as an  
92177 error. A future version of this standard may require implementations to treat this as an error.
- 92178 A future version of this standard may introduce the SIGINFO signal; on platforms where such a  
92179 signal is available, it is recommended that this signal be used for reporting status without  
92180 terminating the process.
- 92181 **SEE ALSO**
- 92182 [Section 1.4](#) (on page 2462), *sed*, *tr*
- 92183 [XBD Chapter 8](#) (on page 167)
- 92184 **CHANGE HISTORY**
- 92185 First released in Issue 2.
- 92186 **Issue 5**
- 92187 The second paragraph of the **cbs=** description is reworded and marked EX.
- 92188 The FUTURE DIRECTIONS section is added.
- 92189 **Issue 6**
- 92190 Changes are made to **swab** conversion and SIGINT handling to align with the IEEE P1003.2b  
92191 draft standard.
- 92192 The normative text is reworded to avoid use of the term “must” for application requirements.
- 92193 IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between **dd of=file** and  
92194 **conv=notrunc**.
- 92195 **Issue 7**
- 92196 Austin Group Interpretation 1003.1-2001 #102 is applied.
- 92197 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 92198 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0081 [907] is applied.
- 92199 **Issue 8**
- 92200 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
92201 filenames containing any bytes that have the encoded value of a <newline> character.
- 92202 Austin Group Defect 406 is applied, adding the **iflags=fullblock** operand.
- 92203 Austin Group Defect 1122 is applied, changing the description of **NLSPATH**.
- 92204 Austin Group Defect 1159 is applied, changing the ASYNCHRONOUS EVENTS and FUTURE

92205

DIRECTIONS sections.

92206

Austin Group Defect 1497 is applied, changing the EXIT STATUS section.

92207 **NAME**92208 delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)92209 **SYNOPSIS**92210 XSI `delta [-nps] [-g list] [-m mrlist] [-r SID] [-y[comment]] file...`92211 **DESCRIPTION**92212 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that  
92213 were made to the files retrieved by *get* (called the *g-files*, or generated files).92214 **OPTIONS**92215 The *delta* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the `-y` option has an  
92216 optional option-argument. This optional option-argument shall not be presented as a separate  
92217 argument.

92218 The following options shall be supported:

92219 `-r SID` Uniquely identify which delta is to be made to the SCCS file. The use of this option  
92220 shall be necessary only if two or more outstanding *get* commands for editing (*get*  
92221 `-e`) on the same SCCS file were done by the same person (login name). The SID  
92222 value specified with the `-r` option can be either the SID specified on the *get*  
92223 command line or the SID to be made as reported by the *get* utility; see *get* (on page  
92224 2964).92225 `-s` Suppress the report to standard output of the activity associated with each *file*. See  
92226 the `STDOUT` section.92227 `-n` Specify retention of the edited *g-file* (normally removed at completion of delta  
92228 processing).92229 `-g list` Specify a *list* (see *get* for the definition of *list*) of deltas that shall be ignored when  
92230 the file is accessed at the change level (SID) created by this delta.92231 `-m mrlist` Specify a modification request (MR) number that the application shall supply as  
92232 the reason for creating the new delta. This shall be used if the SCCS file has the `v`  
92233 flag set; see *admin*.92234 If `-m` is not used and `'-'` is not specified as a file argument, and the standard  
92235 input is a terminal, the prompt described in the `STDOUT` section shall be written  
92236 to standard output before the standard input is read; if the standard input is not a  
92237 terminal, no prompt shall be issued.92238 MRs in a list shall be separated by `<blank>` characters or escaped `<newline>`  
92239 characters. An unescaped `<newline>` shall terminate the MR list. The escape  
92240 character is `<backslash>`.92241 If the `v` flag has a value, it shall be taken to be the name of a program which  
92242 validates the correctness of the MR numbers. If a non-zero exit status is returned  
92243 from the MR number validation program, the *delta* utility shall terminate. (It is  
92244 assumed that the MR numbers were not all valid.)92245 `-y[comment]` Describe the reason for making the delta. The *comment* shall be an arbitrary group  
92246 of lines that would meet the definition of a text file. Implementations shall support  
92247 *comments* from zero to 512 bytes and may support longer values. A null string  
92248 (specified as either `-y`, `-y"`, or in response to a prompt for a comment) shall be  
92249 considered a valid *comment*.92250 If `-y` is not specified and `'-'` is not specified as a file argument, and the standard

92251 input is a terminal, the prompt described in the STDOUT section shall be written  
 92252 to standard output before the standard input is read; if the standard input is not a  
 92253 terminal, no prompt shall be issued. An unescaped <newline> shall terminate the  
 92254 comment text. The escape character is <backslash>.

92255 The `-y` option shall be required if the *file* operand is specified as `'-'`.

92256 **-p** Write (to standard output) the SCCS file differences before and after the delta is  
 92257 applied in *diff* format; see *diff*.

## 92258 OPERANDS

92259 The following operand shall be supported:

92260 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*  
 92261 utility shall behave as though each file in the directory were specified as a named  
 92262 file, except that non-SCCS files (last component of the pathname does not begin  
 92263 with **s**.) and unreadable files shall be silently ignored.

92264 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;  
 92265 each line of the standard input shall be taken to be the name of an SCCS file to be  
 92266 processed. Non-SCCS files and unreadable files shall be silently ignored.

## 92267 STDIN

92268 The standard input shall be a text file used only in the following cases:

- 92269 • To read an *mrlist* or a *comment* (see the `-m` and `-y` options).
- 92270 • A *file* operand shall be specified as `'-'`. In this case, the `-y` option needs to be used to  
 92271 specify the comment, and if the SCCS file has the **v** flag set, the `-m` option also needs to be  
 92272 used to specify the MR list.

## 92273 INPUT FILES

92274 Input files shall be text files whose data is to be included in the SCCS files. If the first character of  
 92275 any line of an input file is <SOH> in the POSIX locale, the results are unspecified. If this file  
 92276 contains more than 99 999 lines, the number of lines recorded in the header for this file shall be  
 92277 99 999 for this delta.

## 92278 ENVIRONMENT VARIABLES

92279 The following environment variables shall affect the execution of *delta*:

92280 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 92281 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 92282 variables used to determine the values of locale categories.)

92283 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 92284 internationalization variables.

92285 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 92286 characters (for example, single-byte as opposed to multi-byte characters in  
 92287 arguments and input files).

92288 **LC\_MESSAGES**

92289 Determine the locale that should be used to affect the format and contents of  
 92290 diagnostic messages written to standard error, and informative messages written  
 92291 to standard output.

92292 **NLSPATH** Determine the location of messages objects and message catalogs.

92293 *TZ* Determine the timezone in which the time and date are written in the SCCS file. If  
 92294 the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

### 92295 **ASYNCHRONOUS EVENTS**

92296 If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit  
 92297 code. The standard action shall be taken for all other signals; see [Section 1.4](#) (on page 2462).

### 92298 **STDOUT**

92299 The standard output shall be used only for the following messages in the POSIX locale:

- 92300 • Prompts (see the *-m* and *-y* options) in the following formats:

92301 "MRs? "

92302 "comments? "

92303 The MR prompt, if written, shall always precede the comments prompt.

- 92304 • A report of each file's activities (unless the *-s* option is specified) in the following format:

92305 "%s\n%d inserted\n%d deleted\n%d unchanged\n", *<New SID>*,  
 92306 *<number of lines inserted>*, *<number of lines deleted>*,  
 92307 *<number of lines unchanged>*

### 92308 **STDERR**

92309 The standard error shall be used only for diagnostic messages.

### 92310 **OUTPUT FILES**

92311 Any SCCS files updated shall be files of an unspecified format.

### 92312 **EXTENDED DESCRIPTION**

#### 92313 **System Date and Time**

92314 When a *delta* is added to an SCCS file, the system date and time shall be recorded for the new  
 92315 *delta*. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the  
 92316 behavior is unspecified.

### 92317 **EXIT STATUS**

92318 The following exit values shall be returned:

92319 0 Successful completion.

92320 >0 An error occurred.

### 92321 **CONSEQUENCES OF ERRORS**

92322 Default.

### 92323 **APPLICATION USAGE**

92324 Problems can arise if the system date and time have been modified (for example, put forward  
 92325 and then back again, or unsynchronized clocks across a network) and can also arise when  
 92326 different values of the *TZ* environment variable are used.

92327 Problems of a similar nature can also arise for the operation of the *get* utility, which records the  
 92328 date and time in the file body.

### 92329 **EXAMPLES**

92330 None.

**92331 RATIONALE**

92332 None.

**92333 FUTURE DIRECTIONS**

92334 If this utility is directed to display a pathname that contains any bytes that have the encoded  
92335 value of a <newline> character when <newline> is a terminator or separator in the output  
92336 format being used, implementations are encouraged to treat this as an error. A future version of  
92337 this standard may require implementations to treat this as an error.

92338 If this utility is directed to create a new directory entry that contains any bytes that have the  
92339 encoded value of a <newline> character, implementations are encouraged to treat this as an  
92340 error. A future version of this standard may require implementations to treat this as an error.

**92341 SEE ALSO**

92342 [Section 1.4](#) (on page 2462), *admin*, *diff*, *get*, *prs*, *rmdel*

92343 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**92344 CHANGE HISTORY**

92345 First released in Issue 2.

**92346 Issue 5**

92347 The output format description in the STDOUT section is corrected.

**92348 Issue 6**

92349 The APPLICATION USAGE section is added.

92350 The normative text is reworded to avoid use of the term “must” for application requirements.

92351 The Open Group Base Resolution bwg2001-007 is applied as follows:

- 92352 • The use of '-' as a file argument is clarified.
- 92353 • The use of STDIN is added.
- 92354 • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement  
92355 that implementations re-signal themselves when catching a normally fatal signal.
- 92356 • New text is added to the INPUT FILES section warning that the maximum lines recorded  
92357 in the file is 99 999.

92358 New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections  
92359 regarding how the system date and time may be taken into account, and the TZ environment  
92360 variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base  
92361 Resolution bwg2001-007.

**92362 Issue 7**

92363 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**92364 Issue 8**

92365 Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

- 92366 a. Report an error if a utility is directed to display a pathname that contains any bytes that  
92367 have the encoded value of a <newline> character when <newline> is a terminator or  
92368 separator in the output format being used.
- 92369 b. Disallow the creation of filenames containing any bytes that have the encoded value of a  
92370 <newline> character.

92371 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

92372 **NAME**

92373 df — report free disk space

92374 **SYNOPSIS**92375 XSI df [-k] [-P|-t] [*file...*]92376 **DESCRIPTION**

92377 XSI The *df* utility shall write the amount of available space and file slots for file systems on which  
 92378 the invoking user has appropriate read access. File systems shall be specified by the *file*  
 92379 operands; when none are specified, information shall be written for all file systems. The format  
 92380 of the default output from *df* is unspecified, but all space figures are reported in 512-byte units,  
 92381 unless the **-k** option is specified. This output shall contain at least the file system names, amount  
 92382 XSI of available space on each of these file systems, and, if no options other than **-t** are specified, the  
 92383 number of free file slots, or *inodes*, available; when **-t** is specified, the output shall contain the  
 92384 total allocated space as well.

92385 **OPTIONS**92386 The *df* utility shall conform to XBD Section 12.2 (on page 215).

92387 The following options shall be supported:

92388 **-k** Use 1024-byte units, instead of the default 512-byte units, when writing space  
 92389 figures.

92390 **-P** Produce output in the format described in the STDOUT section.

92391 XSI **-t** Include total allocated-space figures in the output.

92392 **OPERANDS**

92393 The following operand shall be supported:

92394 *file* A pathname of a file within the hierarchy of the desired file system. If a file other  
 92395 XSI than a FIFO, a regular file, a directory, or a special file representing the device  
 92396 containing the file system (for example, */dev/dsk/0s1*) is specified, the results are  
 92397 unspecified. If the *file* operand names a file other than a special file containing a file  
 92398 system, *df* shall write the amount of free space in the file system containing the  
 92399 XSI specified *file* operand. Otherwise, *df* shall write the amount of free space in that  
 92400 file system.

92401 **STDIN**

92402 Not used.

92403 **INPUT FILES**

92404 None.

92405 **ENVIRONMENT VARIABLES**92406 The following environment variables shall affect the execution of *df*:

92407 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 92408 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 92409 variables used to determine the values of locale categories.)

92410 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 92411 internationalization variables.

92412 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 92413 characters (for example, single-byte as opposed to multi-byte characters in  
 92414 arguments).

92415 *LC\_MESSAGES*

92416 Determine the locale that should be used to affect the format and contents of

92417 diagnostic messages written to standard error and informative messages written to

92418 standard output.

92419 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

92420 **ASYNCHRONOUS EVENTS**

92421 Default.

92422 **STDOUT**

92423 When both the **-k** and **-P** options are specified, the following header line shall be written (in the

92424 POSIX locale):

92425 "Filesystem 1024-blocks Used Available Capacity Mounted on\n"

92426 When the **-P** option is specified without the **-k** option, the following header line shall be written

92427 (in the POSIX locale):

92428 "Filesystem 512-blocks Used Available Capacity Mounted on\n"

92429 The implementation may adjust the spacing of the header line and the individual data lines so

92430 that the information is presented in orderly columns.

92431 The remaining output with **-P** shall consist of one line of information for each specified file

92432 system. These lines shall be formatted as follows:

92433 "%s %d %d %d %d%% %s\n", <file system name>, <total space>,  
 92434 <space used>, <space free>, <percentage used>,  
 92435 <file system root>

92436 In the following list, all quantities expressed in 512-byte units (1 024-byte when **-k** is specified)

92437 shall be rounded up to the next higher unit. The fields are:

92438 <file system name>

92439 The name of the file system, in an implementation-defined format.

92440 <total space> The total size of the file system in 512-byte units. The exact meaning of this figure

92441 is implementation-defined, but should include <space used>, <space free>, plus any

92442 space reserved by the system not normally available to a user.

92443 <space used> The total amount of space allocated to existing files in the file system, in 512-byte

92444 units.

92445 <space free> The total amount of space available within the file system for the creation of new

92446 files by unprivileged users, in 512-byte units. When this figure is less than or equal

92447 to zero, it shall not be possible to create any new files on the file system without

92448 first deleting others, unless the process has appropriate privileges. The figure

92449 written may be less than zero.

92450 <percentage used>

92451 The percentage of the normally available space that is currently allocated to all files

92452 on the file system. This shall be calculated using the fraction:

92453 
$$\frac{\text{<space used>}}{(\text{<space used>} + \text{<space free>})}$$

92454 expressed as a percentage. This percentage may be greater than 100 if <space free>

92455 is less than zero. The percentage value shall be expressed as a positive integer, with

92456 any fractional result causing it to be rounded to the next highest integer.



92457 <file system root>  
 92458 The directory below which the file system hierarchy appears.

92459 XSI The output format is unspecified when `-t` is used.

#### 92460 **STDERR**

92461 The standard error shall be used only for diagnostic messages.

#### 92462 **OUTPUT FILES**

92463 None.

#### 92464 **EXTENDED DESCRIPTION**

92465 None.

#### 92466 **EXIT STATUS**

92467 The following exit values shall be returned:

92468 0 Successful completion.

92469 >0 An error occurred.

#### 92470 **CONSEQUENCES OF ERRORS**

92471 Default.

#### 92472 **APPLICATION USAGE**

92473 On most systems, the “name of the file system, in an implementation-defined format” is the  
 92474 special file on which the file system is mounted.

92475 On large file systems, the calculation specified for percentage used can create huge rounding  
 92476 errors.

#### 92477 **EXAMPLES**

92478 1. The following example writes portable information about the `/usr` file system:

```
92479 df -P /usr
```

92480 2. Assuming that `/usr/src` is part of the `/usr` file system, the following produces the same  
 92481 output as the previous example:

```
92482 df -P /usr/src
```

#### 92483 **RATIONALE**

92484 The behavior of `df` with the `-P` option is the default action of the 4.2 BSD `df` utility. The uppercase  
 92485 `-P` was selected to avoid collision with a known industry extension using `-p`.

92486 Historical `df` implementations vary considerably in their default output. It was therefore  
 92487 necessary to describe the default output in a loose manner to accommodate all known historical  
 92488 implementations and to add a portable option (`-P`) to provide information in a portable format.

92489 The use of 512-byte units is historical practice and maintains compatibility with `ls` and other  
 92490 utilities in this volume of POSIX.1-2024. This does not mandate that the file system itself be  
 92491 based on 512-byte blocks. The `-k` option was added as a compromise measure. It was agreed by  
 92492 the standard developers that 512 bytes was the best default unit because of its complete  
 92493 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and  
 92494 that a `-k` option to switch to 1024-byte units was a good compromise. Users who prefer the  
 92495 more logical 1024-byte quantity can easily alias `df` to `df -k` without breaking many historical  
 92496 scripts relying on the 512-byte units.

92497 It was suggested that `df` and the various related utilities be modified to access a `BLOCKSIZE`  
 92498 environment variable to achieve consistency and user acceptance. Since this is not historical

92499 practice on any system, it is left as a possible area for system extensions and will be re-evaluated  
92500 in a future version if it is widely implemented.

92501 **FUTURE DIRECTIONS**

92502 If this utility is directed to display a pathname that contains any bytes that have the encoded  
92503 value of a <newline> character when <newline> is a terminator or separator in the output  
92504 format being used, implementations are encouraged to treat this as an error. A future version of  
92505 this standard may require implementations to treat this as an error.

92506 **SEE ALSO**

92507 *find*

92508 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

92509 **CHANGE HISTORY**

92510 First released in Issue 2.

92511 **Issue 6**

92512 This utility is marked as part of the User Portability Utilities option.

92513 **Issue 7**

92514 Austin Group Interpretation 1003.1-2001 #099 is applied.

92515 The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is  
92516 now an option for interactive utilities.

92517 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92518 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0082 [156] is applied.

92519 **Issue 8**

92520 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
92521 directed to display a pathname that contains any bytes that have the encoded value of a  
92522 <newline> character when <newline> is a terminator or separator in the output format being  
92523 used.

92524 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

92525 **NAME**

92526 diff — compare two files

92527 **SYNOPSIS**92528 diff [-c|-e|-f|-u|-C *n*|-U *n*] [-br] *file1 file2*92529 **DESCRIPTION**

92530 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of  
 92531 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be  
 92532 produced if the files are identical.

92533 **OPTIONS**92534 The *diff* utility shall conform to XBD [Section 12.2](#) (on page 215).

92535 The following options shall be supported:

92536 **-b** Cause any amount of white space at the end of a line to be treated as a single  
 92537 <newline> (that is, the white-space characters preceding the <newline> are  
 92538 ignored) and other strings of white-space characters, not including <newline>  
 92539 characters, to compare equal.

92540 **-c** Produce output in a form that provides three lines of copied context.

92541 **-C *n*** Produce output in a form that provides *n* lines of copied context (where *n* shall be  
 92542 interpreted as a positive decimal integer).

92543 **-e** Produce output in a form suitable as input for the *ed* utility, which can then be used  
 92544 to convert *file1* into *file2*.

92545 **-f** Produce output in an alternative form, similar in format to **-e**, but not intended to  
 92546 be suitable as input for the *ed* utility, and in the opposite order.

92547 **-r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*  
 92548 are both directories.

92549 The *diff* utility shall detect infinite loops; that is, entering a previously visited  
 92550 directory that is an ancestor of the last file encountered. When it detects an infinite  
 92551 loop, *diff* shall write a diagnostic message to standard error and shall either recover  
 92552 its position in the hierarchy or terminate.

92553 **-u** Produce output in a form that provides three lines of unified context.

92554 **-U *n*** Produce output in a form that provides *n* lines of unified context (where *n* shall be  
 92555 interpreted as a non-negative decimal integer).

92556 **OPERANDS**

92557 The following operands shall be supported:

92558 *file1, file2* A pathname of a file to be compared. If either the *file1* or *file2* operand is '-', the  
 92559 standard input shall be used in its place.

92560 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special  
 92561 files, or FIFO special files to any files and shall not compare regular files to directories. Further  
 92562 details are as specified in [Diff Directory Comparison Format](#) (on page 2796). The behavior of *diff*  
 92563 on other file types is implementation-defined when found in directories.

92564 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file  
 92565 contained in the directory file with a filename that is the same as the last component of the non-  
 92566 directory file.

92567 **STDIN**

92568 The standard input shall be used only if one of the *file1* or *file2* operands references standard  
 92569 input. See the INPUT FILES section.

92570 **INPUT FILES**

92571 The input files may be of any type.

92572 **ENVIRONMENT VARIABLES**

92573 The following environment variables shall affect the execution of *diff*:

92574 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 92575 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 92576 variables used to determine the values of locale categories.)

92577 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 92578 internationalization variables.

92579 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 92580 characters (for example, single-byte as opposed to multi-byte characters in  
 92581 arguments and input files).

92582 **LC\_MESSAGES**

92583 Determine the locale that should be used to affect the format and contents of  
 92584 diagnostic messages written to standard error and informative messages written to  
 92585 standard output.

92586 **LC\_TIME** Determine the locale for affecting the format of file timestamps written with the **-C**  
 92587 and **-c** options.

92588 XSI **NLSPATH** Determine the location of messages objects and message catalogs.

92589 **TZ** Determine the timezone used for calculating file timestamps written with a context  
 92590 format. If *TZ* is unset or null, an unspecified default timezone shall be used.

92591 **ASYNCHRONOUS EVENTS**

92592 Default.

92593 **STDOUT**92594 **Diff Directory Comparison Format**

92595 If both *file1* and *file2* are directories, the following output formats shall be used.

92596 In the POSIX locale, each file that is present in only one directory shall be reported using the  
 92597 following format:

92598 "Only in %s: %s\n", <directory pathname>, <filename>

92599 In the POSIX locale, subdirectories that are common to the two directories may be reported with  
 92600 the following format:

92601 "Common subdirectories: %s and %s\n", <directory1 pathname>,  
 92602 <directory2 pathname>

92603 For each file common to the two directories, if the two files are not to be compared: if the two  
 92604 files have the same device ID and file serial number, or are both block special files that refer to  
 92605 the same device, or are both character special files that refer to the same device, in the POSIX  
 92606 locale the output format is unspecified. Otherwise, in the POSIX locale an unspecified format  
 92607 shall be used that contains the pathnames of the two files.

92608 For each file common to the two directories, if the files are compared and are identical, no

92609 output shall be written. If the two files differ, the following format is written:

92610 "diff %s %s %s\n", <diff\_options>, <filename1>, <filename2>

92611 where <diff\_options> are the options as specified on the command line.

92612 All directory pathnames listed in this section shall be relative to the original command line  
 92613 arguments. All other names of files listed in this section shall be filenames (pathname  
 92614 components).

### 92615 **Diff Binary Output Format**

92616 In the POSIX locale, if one or both of the files being compared are not text files, it is  
 92617 implementation-defined whether *diff* uses the binary file output format or the other formats as  
 92618 specified below. The binary file output format shall contain the pathnames of two files being  
 92619 compared and the string "differ".

92620 If both files being compared are text files, depending on the options specified, one of the  
 92621 following formats shall be used to write the differences.

### 92622 **Diff Default Output Format**

92623 The default (without *-e*, *-f*, *-c*, *-C*, *-u*, or *-U* options) *diff* utility output shall contain lines of  
 92624 these forms:

92625 "%da%d\n", <num1>, <num2>

92626 "%da%d,%d\n", <num1>, <num2>, <num3>

92627 "%dd%d\n", <num1>, <num2>

92628 "%d,%dd%d\n", <num1>, <num2>, <num3>

92629 "%dc%d\n", <num1>, <num2>

92630 "%d,%dc%d\n", <num1>, <num2>, <num3>

92631 "%dc%d,%d\n", <num1>, <num2>, <num3>

92632 "%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>

92633 These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the  
 92634 action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d*  
 92635 and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in  
 92636 *ed*, identical pairs (where *num1*=*num2*) are abbreviated as a single number.

92637 Following each of these lines, *diff* shall write to standard output all lines affected in the first file  
 92638 using the format:

92639 "<Δ%s", <line>

92640 and all lines affected in the second file using the format:

92641 ">Δ%s", <line>

92642 If there are lines affected in both *file1* and *file2* (as with the *c* subcommand), the changes are  
 92643 separated with a line consisting of three <hyphen-minus> characters:

92644 "---\n"

92645 **Diff -e Output Format**

92646 With the `-e` option, a script shall be produced that shall, when provided as input to `ed`, along  
 92647 with an appended `w` (write) command, convert *file1* into *file2*. Only the `a` (append), `c` (change), `d`  
 92648 (delete), `i` (insert), and `s` (substitute) commands of `ed` shall be used in this script. Text lines,  
 92649 except those consisting of the single character `<period>` (`'.'`), shall be output as they appear in  
 92650 the file.

92651 **Diff -f Output Format**

92652 With the `-f` option, an alternative format of script shall be produced. It is similar to that  
 92653 produced by `-e`, with the following differences:

- 92654 1. It is expressed in reverse sequence; the output of `-e` orders changes from the end of the  
 92655 file to the beginning; the `-f` from beginning to end.
- 92656 2. The command form `<lines> <command-letter>` used by `-e` is reversed. For example,  
 92657 `10c` with `-e` would be `c10` with `-f`.
- 92658 3. The form used for ranges of line numbers is `<space>`-separated, rather than  
 92659 `<comma>`-separated.

92660 **Diff -c or -C Output Format**

92661 With the `-c` or `-C` option, the output format shall consist of affected lines along with  
 92662 surrounding lines of context. The affected lines shall show which ones need to be deleted or  
 92663 changed in *file1*, and those added from *file2*. With the `-c` option, three lines of context, if  
 92664 available, shall be written before and after the affected lines. With the `-C` option, the user can  
 92665 specify how many lines of context are written. The exact format follows.

92666 The name and last modification time of each file shall be output in the following format:

```
92667 "*** %s %s\n", file1, <file1 timestamp>
92668 "--- %s %s\n", file2, <file2 timestamp>
```

92669 Each `<file>` field shall be the pathname of the corresponding file being compared. The pathname  
 92670 written for standard input is unspecified.

92671 In the POSIX locale, each `<timestamp>` field shall be equivalent to the output from the following  
 92672 command:

```
92673 date "+%a %b %e %T %Y"
```

92674 without the trailing `<newline>`, executed at the time of last modification of the corresponding  
 92675 file (or the current time, if the file is standard input).

92676 Then, the following output formats shall be applied for every set of changes.

92677 First, a line shall be written in the following format:

```
92678 "*****\n"
```

92679 Next, the range of lines in *file1* shall be written in the following format if the range contains two  
 92680 or more lines:

```
92681 "*** %d,%d ***\n", <beginning line number>, <ending line number>
```

92682 and the following format otherwise:

```
92683 "*** %d ***\n", <ending line number>
```

92684 The ending line number of an empty range shall be the number of the preceding line, or 0 if the

92685 range is at the start of the file.

92686 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected  
92687 lines shall be written in the following format:

92688 " $\Delta$ %s", <unaffected\_line>

92689 Deleted lines shall be written as:

92690 "- $\Delta$ %s", <deleted\_line>

92691 Changed lines shall be written as:

92692 "! $\Delta$ %s", <changed\_line>

92693 Next, the range of lines in *file2* shall be written in the following format if the range contains two  
92694 or more lines:

92695 "--- %d,%d ----\n", <beginning line number>, <ending line number>

92696 and the following format otherwise:

92697 "--- %d ----\n", <ending line number>

92698 Then, lines of context and changed lines shall be written as described in the previous formats.  
92699 Lines added from *file2* shall be written in the following format:

92700 "+ $\Delta$ %s", <added\_line>

#### 92701 **Diff -u or -U Output Format**

92702 The -u or -U options behave like the -c or -C options, except that the context lines are not  
92703 repeated; instead, the context, deleted, and added lines are shown together, interleaved. The  
92704 exact format follows.

92705 The name and last modification time of each file shall be output in the following format:

92706 "--- $\Delta$ %s\t%s%s $\Delta$ %s\n", file1, <file1 timestamp>, <file1 frac>, <file1 zone>  
92707 "+++ $\Delta$ %s\t%s%s $\Delta$ %s\n", file2, <file2 timestamp>, <file2 frac>, <file2 zone>

92708 Each <file> field shall be the pathname of the corresponding file being compared, or the single  
92709 character '-' if standard input is being compared. However, if the pathname contains a <tab>  
92710 or a <newline>, or if it does not consist entirely of characters taken from the portable character  
92711 set, the behavior is implementation-defined.

92712 Each <timestamp> field shall be equivalent to the output from the following command:

92713 date '+%Y-%m-%d $\Delta$ %H:%M:%S'

92714 without the trailing <newline>, executed at the time of last modification of the corresponding  
92715 file (or the current time, if the file is standard input).

92716 Each <frac> field shall be either empty, or a decimal point followed by at least one decimal digit,  
92717 indicating the fractional-seconds part (if any) of the file timestamp. The number of fractional  
92718 digits shall be at least the number needed to represent the file's timestamp without loss of  
92719 information.

92720 Each <zone> field shall be of the form "shhmm", where "shh" is a signed two-digit decimal  
92721 number in the range -24 through +25, and "mm" is an unsigned two-digit decimal number in the  
92722 range 00 through 59. It represents the timezone of the timestamp as the number of hours (hh)  
92723 and minutes (mm) east (+) or west (-) of UTC for the timestamp. If the hours and minutes are  
92724 both zero, the sign shall be '+'. However, if the timezone is not an integral number of minutes

92725 away from UTC, the `<zone>` field is implementation-defined.

92726 Then, the following output formats shall be applied for every set of changes.

92727 First, the range of lines in each file shall be written in the following format:

92728 `"@@Δ-%sΔ+%sΔ@@", <file1 range>, <file2 range>`

92729 Each `<range>` field shall be of the form:

92730 `"%ld", <beginning line number>`

92731 or:

92732 `"%ld,1", <beginning line number>`

92733 if the range contains exactly one line, and:

92734 `"%ld,%ld", <beginning line number>, <number of lines>`

92735 otherwise. If a range is empty, its beginning line number shall be the number of the line just

92736 before the range, or 0 if the empty range starts the file.

92737 Next, the affected lines along with lines of context shall be written. Each non-empty unaffected

92738 line shall be written in the following format:

92739 `"Δ%s", <unaffected_line>`

92740 where the contents of the unaffected line shall be taken from *file1*. It is implementation-defined

92741 whether an empty unaffected line is written as an empty line or a line containing a single

92742 `<space>` character. This line also represents the same line of *file2*, even though *file2*'s line may

92743 contain different contents due to the `-b`. Deleted lines shall be written as:

92744 `"-%s", <deleted_line>`

92745 Added lines shall be written as:

92746 `"+%s", <added_line>`

92747 The order of lines written shall be the same as that of the corresponding file. A deleted line shall

92748 never be written immediately after an added line.

92749 If `-U n` is specified, the output shall contain no more than  $2n$  consecutive unaffected lines; and if

92750 the output contains an affected line and this line is adjacent to up to  $n$  consecutive unaffected

92751 lines in the corresponding file, the output shall contain these unaffected lines. `-u` shall act like

92752 `-U3`.

92753 **STDERR**

92754 The standard error shall be used only for diagnostic messages.

92755 **OUTPUT FILES**

92756 None.

92757 **EXTENDED DESCRIPTION**

92758 None.

92759 **EXIT STATUS**

92760 The following exit values shall be returned:

92761 0 No differences were found.

92762 1 Differences were found and all differences were successfully output.



92763 >1 An error occurred.

## 92764 CONSEQUENCES OF ERRORS

92765 Default.

## 92766 APPLICATION USAGE

92767 If lines at the end of a file are changed and other lines are added, *diff* output may show this as a  
 92768 delete and add, as a change, or as a change and add; *diff* is not expected to know which  
 92769 happened and users should not care about the difference in output as long as it clearly shows  
 92770 the differences between the files.

## 92771 EXAMPLES

92772 If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory  
 92773 named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**,  
 92774 the command:

```
92775 diff -r dir1 dir2
```

92776 could produce output similar to:

```
92777 Common subdirectories: dir1/x and dir2/x
92778 Only in dir2/x: y
92779 diff -r dir1/x/date.out dir2/x/date.out
92780 1c1
92781 < Mon Jul  2 13:12:16 PDT 1990
92782 ---
92783 > Tue Jun 19 21:41:39 PDT 1990
```

## 92784 RATIONALE

92785 The **-h** option was omitted because it was insufficiently specified and does not add to  
 92786 applications portability.

92787 Historical implementations employ algorithms that do not always produce a minimum list of  
 92788 differences; the current language about making every effort is the best this volume of  
 92789 POSIX.1-2024 can do, as there is no metric that could be employed to judge the quality of  
 92790 implementations against any and all file contents. The statement “This list should be minimal”  
 92791 clearly implies that implementations are not expected to provide the following output when  
 92792 comparing two 100-line files that differ in only one character on a single line:

```
92793 1,100c1,100
92794 all 100 lines from file1 preceded with "< "
92795 ---
92796 all 100 lines from file2 preceded with "> "
```

92797 The “Only in” messages required when the **-r** option is specified are not used by most historical  
 92798 implementations if the **-e** option is also specified. It is required here because it provides useful  
 92799 information that must be provided to update a target directory hierarchy to match a source  
 92800 hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when  
 92801 the **-r** option is specified. They are allowed here but are not required because they are reporting  
 92802 on something that is the same, not reporting a difference, and are not needed to update a target  
 92803 hierarchy.

92804 The **-c** option, which writes output in a format using lines of context, has been included. The  
 92805 format is useful for a variety of reasons, among them being much improved readability and the  
 92806 ability to understand difference changes when the target file has line numbers that differ from  
 92807 another similar, but slightly different, copy. The *patch* utility is most valuable when working  
 92808 with difference listings using a context format. The BSD version of **-c** takes an optional

92809 argument specifying the amount of context. Rather than overloading `-c` and breaking the Utility  
 92810 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for  
 92811 specifying a context *diff* with a specified amount of context (`-C`). Also, the format for context  
 92812 *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines  
 92813 from each other to be merged together. The output format contains an additional four `<asterisk>`  
 92814 characters after the range of affected lines in the first filename. This was to provide a flag for old  
 92815 programs (like old versions of *patch*) that only understand the old context format. The version of  
 92816 context described here does not require that multiple changes within context lines be merged,  
 92817 but it does not prohibit it either. The extension is upwards-compatible, so any vendors that wish  
 92818 to retain the old version of *diff* can do so by adding the extra four `<asterisk>` characters (that is,  
 92819 utilities that currently use *diff* and understand the new merged format will also understand the  
 92820 old unmerged format, but not *vice versa*).

92821 The `-u` and `-U` options of GNU *diff* have been included. Their output format, designed by  
 92822 Wayne Davison, takes up less space than `-c` and `-C` format, and in many cases is easier to read.  
 92823 The format's timestamps do not vary by locale, so `LC_TIME` does not affect it. The format's line  
 92824 numbers are rendered with the `%1d` format, not `%d`, because the file format notation rules would  
 92825 allow extra `<blank>` characters to appear around the numbers.

92826 The substitute command was added as an additional format for the `-e` option. This was added  
 92827 to provide implementations with a way to fix the classic ```dot alone on a line``` bug present in  
 92828 many versions of *diff*. Since many implementations have fixed this bug, the standard developers  
 92829 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing  
 92830 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then  
 92831 terminate the append command with a period, and then use the substitute command to convert  
 92832 the two periods into one period.

92833 The BSD-derived `-r` option was added to provide a mechanism for using *diff* to compare two file  
 92834 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not  
 92835 easily reproducible with the *find* utility.

92836 The requirement that *diff* not compare files in some circumstances, even though they have the  
 92837 same name, is based on the actual output of historical implementations. The specified behavior  
 92838 precludes the problems arising from running into FIFOs and other files that would cause *diff*  
 92839 to hang waiting for input with no indication to the user that *diff* was hung. An earlier version of  
 92840 this standard specified the output format more precisely, but in practice this requirement was  
 92841 widely ignored and the benefit of standardization seemed small, so it is now unspecified. In  
 92842 most common usage, *diff -r* should indicate differences in the file hierarchies, not the difference  
 92843 of contents of devices pointed to by the hierarchies.

92844 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of  
 92845 POSIX.1-2024 supports named pipes, the standard developers decided that such a restriction  
 92846 was unreasonable. Note also that the allowed filename – almost always refers to a pipe.

92847 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in  
 92848 that it prints out all of the differences at the current level, including the statements about all  
 92849 common subdirectories before recursing into those subdirectories.

92850 The message:

92851 `"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`

92852 does not vary by locale because it is the representation of a command, not an English sentence.

**92853 FUTURE DIRECTIONS**

92854 If this utility is directed to display a pathname that contains any bytes that have the encoded  
92855 value of a <newline> character when <newline> is a terminator or separator in the output  
92856 format being used, implementations are encouraged to treat this as an error. A future version of  
92857 this standard may require implementations to treat this as an error.

**92858 SEE ALSO**

92859 *cmp, comm, ed, find*

92860 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

**92861 CHANGE HISTORY**

92862 First released in Issue 2.

**92863 Issue 5**

92864 The FUTURE DIRECTIONS section is added.

**92865 Issue 6**

92866 The following new requirements on POSIX implementations derive from alignment with the  
92867 Single UNIX Specification:

- 92868 • The `-f` option is added.

92869 The output format for `-c` or `-C` format is changed to align with changes to the IEEE P1003.2b  
92870 draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

92871 The normative text is reworded to avoid use of the term “must” for application requirements.

92872 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/20 is applied, changing the STDOUT  
92873 section. This changes the specification of *diff -c* so that it agrees with existing practice when  
92874 contexts contain zero lines or one line.

**92875 Issue 7**

92876 Austin Group Interpretations 1003.1-2001 #115 and #114 are applied.

92877 Austin Group Interpretation 1003.1-2001 #192 is applied, clarifying the behavior if both files are  
92878 non-text files.

92879 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92880 SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` option.

92881 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0082 [584], XCU/TC2-2008/0083  
92882 [950], XCU/TC2-2008/0084 [969], and XCU/TC2-2008/0085 [929] are applied.

**92883 Issue 8**

92884 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
92885 directed to display a pathname that contains any bytes that have the encoded value of a  
92886 <newline> character when <newline> is a terminator or separator in the output format being  
92887 used.

92888 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

92889 Austin Group Defect 1498 is applied, changing the EXIT STATUS section.

92890 **NAME**

92891           dirname — return the directory portion of a pathname

92892 **SYNOPSIS**92893           dirname *string*92894 **DESCRIPTION**

92895           The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.254](#) (on page 68),  
 92896           and shall be converted to a pathname of the directory containing the entry of the final pathname  
 92897           component. The resulting string shall be written to standard output. The *dirname* utility shall not  
 92898           perform pathname resolution; the result shall not be affected by whether or not a file with the  
 92899           pathname *string* exists or by its file type. Trailing '/' characters in *string* that are not also  
 92900           leading '/' characters shall not be counted as part of the pathname. If the pathname does not  
 92901           contain a '/', the resulting string shall be ".". If *string* is an empty string, the resulting string  
 92902           shall be ".".

92903           It is unspecified whether redundant '/' characters and '.' pathname components in *string* are  
 92904           removed after determining the pathname to output. However, "." pathname components  
 92905           occurring prior to the final component shall not be removed.

92906 **OPTIONS**

92907           None.

92908 **OPERANDS**

92909           The following operand shall be supported:

92910           *string*           A string.92911 **STDIN**

92912           Not used.

92913 **INPUT FILES**

92914           None.

92915 **ENVIRONMENT VARIABLES**92916           The following environment variables shall affect the execution of *dirname*:

92917           *LANG*           Provide a default value for the internationalization variables that are unset or null.  
 92918           (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 92919           variables used to determine the values of locale categories.)

92920           *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
 92921           internationalization variables.

92922           *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
 92923           characters (for example, single-byte as opposed to multi-byte characters in  
 92924           arguments).

92925           *LC\_MESSAGES*

92926           Determine the locale that should be used to affect the format and contents of  
 92927           diagnostic messages written to standard error.

92928   XSI       *NLSPATH*   Determine the location of messages objects and message catalogs.

92929 **ASYNCHRONOUS EVENTS**

92930           Default.

92931 **STDOUT**

92932 The *dirname* utility shall write a line to the standard output in the following format:

92933 "%s\n", <resulting string>

92934 **STDERR**

92935 The standard error shall be used only for diagnostic messages.

92936 **OUTPUT FILES**

92937 None.

92938 **EXTENDED DESCRIPTION**

92939 None.

92940 **EXIT STATUS**

92941 The following exit values shall be returned:

92942 0 Successful completion.

92943 >0 An error occurred.

92944 **CONSEQUENCES OF ERRORS**

92945 Default.

92946 **APPLICATION USAGE**

92947 The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
92948 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters  
92949 to the beginning of a pathname unless they can ensure that there are more or less than two or are  
92950 prepared to deal with the implementation-defined consequences.

92951 **EXAMPLES**

92952 The EXAMPLES section of the *basename()* function (see XSH *basename()*) includes a table  
92953 showing examples of the results of processing several sample pathnames by the *basename()* and  
92954 *dirname()* functions and by the *basename* and *dirname* utilities.

92955 See also the examples for the *basename* utility.

92956 **RATIONALE**

92957 The behaviors of *basename* and *dirname* in this volume of POSIX.1-2024 have been coordinated so  
92958 that when *string* is a valid pathname:

92959 `$(basename -- "string")`

92960 would be a valid filename for the file in the directory:

92961 `$(dirname -- "string")`

92962 This would not work for the versions of these utilities in early proposals due to the way  
92963 processing of trailing <slash> characters was specified. Consideration was given to leaving  
92964 processing unspecified if there were trailing <slash> characters, but this cannot be done; XBD  
92965 [Section 3.254](#) (on page 68) allows trailing <slash> characters. The *basename* and *dirname* utilities  
92966 have to specify consistent handling for all valid pathnames.

92967 The *dirname* utility is not specified in terms of the *dirname()* function, because the two may  
92968 produce slightly different output where both output forms are still compliant. An  
92969 implementation should prefer the shortest output possible; however, this is not required, in part  
92970 because earlier versions of the standard did not permit elision of redundant <slash> characters  
92971 or dot ('.') components. Removal of the dot-dot ("..") pathname component is not permitted,  
92972 because eliding it correctly would require performing pathname resolution to ensure the  
92973 resulting string would still point to the correct pathname if the original string resolved as a

92974 pathname. On implementations where pathname "/" has an implementation-defined meaning  
92975 distinct from the pathname "/", the dirname of "/" will be "/".

#### 92976 **FUTURE DIRECTIONS**

92977 If this utility is directed to display a pathname that contains any bytes that have the encoded  
92978 value of a <newline> character when <newline> is a terminator or separator in the output  
92979 format being used, implementations are encouraged to treat this as an error. A future version of  
92980 this standard may require implementations to treat this as an error.

#### 92981 **SEE ALSO**

92982 [Section 2.5](#) (on page 2478), *basename*  
92983 [XBD Section 3.254](#) (on page 68), [Chapter 8](#) (on page 167)  
92984 XSH *basename()*, *dirname()*

#### 92985 **CHANGE HISTORY**

92986 First released in Issue 2.

#### 92987 **Issue 7**

92988 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0083 [192,430], XCU/TC1-2008/0084  
92989 [192], and XCU/TC1-2008/0085 [192] are applied.

92990 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0086 [612], XCU/TC2-2008/0087  
92991 [620], and XCU/TC2-2008/0088 [612] are applied.

#### 92992 **Issue 8**

92993 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
92994 directed to display a pathname that contains any bytes that have the encoded value of a  
92995 <newline> character when <newline> is a terminator or separator in the output format being  
92996 used.

92997 Austin Group Defect 1073 is applied, replacing the DESCRIPTION section with one that matches  
92998 the *dirname()* function.

92999 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

93000 **NAME**

93001 du — estimate file space usage

93002 **SYNOPSIS**93003 du [-a|-s] [-kx] [-H|-L] [*file*...]93004 **DESCRIPTION**

93005 By default, the *du* utility shall write to standard output the size of the file space allocated to, and  
 93006 the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the  
 93007 specified files. By default, when a symbolic link is encountered on the command line or in the  
 93008 file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the  
 93009 link), and shall not follow the link to another portion of the file hierarchy. The size of the file  
 93010 space allocated to a file of type directory shall be defined as the sum total of space allocated to  
 93011 all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

93012 When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the  
 93013 final exit status is affected. A file that occurs multiple times shall be counted and written for only  
 93014 one entry, even if the occurrences are under different file operands. The directory entry that is  
 93015 selected in the report is unspecified. By default, file sizes shall be written in 512-byte units,  
 93016 rounded up to the next 512-byte unit.

93017 **OPTIONS**93018 The *du* utility shall conform to XBD [Section 12.2](#) (on page 215).

93019 The following options shall be supported:

93020 **-a** In addition to the default output, report the size of each file not of type directory in  
 93021 the file hierarchy rooted in the specified file. The **-a** option shall not affect whether  
 93022 non-directories given as *file* operands are listed.

93023 **-H** If a symbolic link is specified on the command line, *du* shall count the size of the  
 93024 file or file hierarchy referenced by the link.

93025 **-k** Write the files sizes in units of 1024 bytes, rather than the default 512-byte units.

93026 **-L** If a symbolic link is specified on the command line or encountered during the  
 93027 traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy  
 93028 referenced by the link.

93029 **-s** Instead of the default output, report only the total sum for each of the specified  
 93030 files.

93031 **-x** When evaluating file sizes, evaluate only those files that have the same device as  
 93032 the file specified by the *file* operand.

93033 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 93034 an error. The last option specified shall determine the behavior of the utility.

93035 **OPERANDS**

93036 The following operand shall be supported:

93037 *file* The pathname of a file whose size is to be written. If no *file* is specified, the current  
 93038 directory shall be used.

93039 **STDIN**

93040 Not used.

93041 **INPUT FILES**

93042 None.

93043 **ENVIRONMENT VARIABLES**93044 The following environment variables shall affect the execution of *du*:

93045 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 93046 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 93047 variables used to determine the values of locale categories.)

93048 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 93049 internationalization variables.

93050 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 93051 characters (for example, single-byte as opposed to multi-byte characters in  
 93052 arguments).

93053 *LC\_MESSAGES*

93054 Determine the locale that should be used to affect the format and contents of  
 93055 diagnostic messages written to standard error.

93056 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

93057 **ASYNCHRONOUS EVENTS**

93058 Default.

93059 **STDOUT**

93060 The output from *du* shall consist of the amount of space allocated to a file and the name of the  
 93061 file, in the following format:

93062 "%d %s\n", &lt;size&gt;, &lt;pathname&gt;

93063 **STDERR**

93064 The standard error shall be used only for diagnostic messages.

93065 **OUTPUT FILES**

93066 None.

93067 **EXTENDED DESCRIPTION**

93068 None.

93069 **EXIT STATUS**

93070 The following exit values shall be returned:

93071 0 Successful completion.

93072 &gt;0 An error occurred.

93073 **CONSEQUENCES OF ERRORS**

93074 Default.



## 93075 APPLICATION USAGE

93076 None.

## 93077 EXAMPLES

93078 None.

## 93079 RATIONALE

93080 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other  
93081 utilities in this volume of POSIX.1-2024. This does not mandate that the file system itself be  
93082 based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed by  
93083 the standard developers that 512 bytes was the best default unit because of its complete  
93084 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and  
93085 that a **-k** option to switch to 1024-byte units was a good compromise. Users who prefer the  
93086 1024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts  
93087 relying on the 512-byte units.

93088 The **-b** option was added to an early proposal to provide a resolution to the situation where  
93089 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-  
93090 defined concept. (In common usage, the block size is 512 bytes for System V and 1024 bytes for  
93091 BSD systems.) However, **-b** was later deleted, since the default was eventually decided as  
93092 512-byte units.

93093 Historical file systems provided no way to obtain exact figures for the space allocation given to  
93094 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*  
93095 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is  
93096 space used by the file system in the storage of the file, but that need not be counted in the space  
93097 allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position  
93098 beyond the end of the file and data has subsequently been written at that point. A file system  
93099 need not allocate all the intervening zero-filled blocks to such a file. It is up to the  
93100 implementation to define exactly how accurate its methods are.

93101 The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell  
93102 and Utilities description is implied by the language in the SVID where **-s** is described as causing  
93103 “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly  
93104 Conforming POSIX Shell and Utilities Application cannot use that combination.

93105 The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not  
93106 listing non-directories explicitly given as operands, unless the **-a** option is specified, was  
93107 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default  
93108 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no  
93109 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and  
93110 Utilities default behavior shall be to produce such messages. These messages can be turned off  
93111 with shell redirection to achieve the System V behavior.

93112 The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of  
93113 POSIX.1-2024 because there was no other historical method of limiting the *du* search to a single  
93114 file hierarchy. This limitation of the search is necessary to make it possible to obtain file space  
93115 usage information about a file system on which other file systems are mounted, without having  
93116 to resort to a lengthy *find* and *awk* script.

93117 The use of the **-L** option, or of multiple *file* operands, requires that *du* track all file entries  
93118 encountered, even with a link count of one. However, when **-L** is not used and only a single *file*  
93119 operand is given, an implementation can optimize by only tracking files with a link count  
93120 greater than one, since in that scenario, those are the only files that could be encountered more  
93121 than once.

**93122 FUTURE DIRECTIONS**

93123 If this utility is directed to display a pathname that contains any bytes that have the encoded  
93124 value of a <newline> character when <newline> is a terminator or separator in the output  
93125 format being used, implementations are encouraged to treat this as an error. A future version of  
93126 this standard may require implementations to treat this as an error.

**93127 SEE ALSO**

93128 *ls*

93129 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

93130 XSH *fstatat()*

**93131 CHANGE HISTORY**

93132 First released in Issue 2.

**93133 Issue 6**

93134 This utility is marked as part of the User Portability Utilities option.

93135 The APPLICATION USAGE section is added.

93136 The obsolescent `-r` option is removed.

93137 The Open Group Corrigendum U025/3 is applied. The *du* utility is reinstated, as it had  
93138 incorrectly been marked LEGACY in Issue 5.

93139 The `-H` and `-L` options for symbolic links are added as described in the IEEE P1003.2b draft  
93140 standard.

**93141 Issue 7**

93142 The *du* utility is moved from the User Portability Utilities option to the Base. User Portability  
93143 Utilities is now an option for interactive utilities.

93144 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

93145 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0089 [527] is applied.

**93146 Issue 8**

93147 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
93148 directed to display a pathname that contains any bytes that have the encoded value of a  
93149 <newline> character when <newline> is a terminator or separator in the output format being  
93150 used.

93151 Austin Group Defect 539 is applied, requiring a file that occurs multiple times to be counted and  
93152 written for only one entry.

93153 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

93154 **NAME**

93155 echo — write arguments to standard output

93156 **SYNOPSIS**93157 echo [*string...*]93158 **DESCRIPTION**93159 The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are  
93160 no arguments, only the <newline> is written.93161 **OPTIONS**93162 The *echo* utility shall not recognize the "--" argument in the manner specified by Guideline 10  
93163 of XBD [Section 12.2](#) (on page 215); "--" shall be recognized as a string operand.

93164 Implementations shall not support any options.

93165 **OPERANDS**

93166 The following operands shall be supported:

93167 *string* A string to be written to standard output. If the first operand consists of a '-'  
93168 followed by one or more characters from the set {'e', 'E', 'n'}, or if any of the  
93169 operands contain a <backslash> character, the results are implementation-defined.93170 XSI On XSI-conformant systems, if the first operand consists of a '-' followed by one  
93171 or more characters from the set {'e', 'E', 'n'}, it shall be treated as a string to be  
93172 written. The following character sequences shall be recognized on XSI-conformant  
93173 systems within any of the arguments:

93174	\a	Write an <alert>.
93175	\b	Write a <backspace>.
93176	\c	Suppress the <newline> that otherwise follows the final argument in the 93177 output. All characters following the '\c' in the arguments shall be 93178 ignored.
93179	\f	Write a <form-feed>.
93180	\n	Write a <newline>.
93181	\r	Write a <carriage-return>.
93182	\t	Write a <tab>.
93183	\v	Write a <vertical-tab>.
93184	\\	Write a <backslash> character.
93185	\0 <i>num</i>	Write an 8-bit value that is the zero, one, two, or three-digit octal number 93186 <i>num</i> .

93187 **STDIN**

93188 Not used.

93189 **INPUT FILES**

93190 None.

93191 **ENVIRONMENT VARIABLES**93192 The following environment variables shall affect the execution of *echo*:

93193		<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)
93194			
93195			
93196		<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
93197			
93198	XSI	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
93199			
93200			
93201		<i>LC_MESSAGES</i>	
93202			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
93203			
93204	XSI	<i>NLSPATH</i>	Determine the location of messages objects and message catalogs.
93205		<b>ASYNCHRONOUS EVENTS</b>	
93206			Default.
93207		<b>STDOUT</b>	
93208			The <i>echo</i> utility arguments shall be separated by single <space> characters and a <newline> character shall follow the last argument. Output transformations shall occur based on the escape sequences in the input. See the OPERANDS section.
93209	XSI		
93210			
93211		<b>STDERR</b>	
93212			The standard error shall be used only for diagnostic messages.
93213		<b>OUTPUT FILES</b>	
93214			None.
93215		<b>EXTENDED DESCRIPTION</b>	
93216			None.
93217		<b>EXIT STATUS</b>	
93218			The following exit values shall be returned:
93219		0	Successful completion.
93220		>0	An error occurred.
93221		<b>CONSEQUENCES OF ERRORS</b>	
93222			Default.
93223		<b>APPLICATION USAGE</b>	
93224			It is not possible to use <i>echo</i> portably across all POSIX systems unless escape sequences are omitted, and the first argument does not consist of a '-' followed by one or more characters from the set {'e', 'E', 'n'}.
93225			
93226			
93227			The <i>printf</i> utility can be used portably to emulate any of the traditional behaviors of the <i>echo</i> utility as follows (assuming that <i>IFS</i> has its standard value or is unset):
93228			
93229			• The historic System V <i>echo</i> and the requirements on XSI implementations in this volume of POSIX.1-2024 are equivalent to:
93230			
93231			<code>printf "%b\n" "\$*"</code>
93232			• The BSD <i>echo</i> is equivalent to:
93233			<code>if [ "X\$1" = "X-n" ]</code>
93234			<code>then</code>

```

93235         shift
93236         printf "%s" "$*"
93237     else
93238         printf "%s\n" "$*"
93239     fi

```

93240 New applications are encouraged to use *printf* instead of *echo*.

#### 93241 EXAMPLES

93242 None.

#### 93243 RATIONALE

93244 The *echo* utility has not been made obsolescent because of its extremely widespread use in  
 93245 historical applications. Conforming applications that wish to do prompting without <newline>  
 93246 characters or that could possibly be expecting to echo a string consisting of a '-' followed by  
 93247 one or more characters from the set {'e', 'E', 'n'} should use the *printf* utility.

93248 At the time that the IEEE Std 1003.2-1992 standard was being developed, the two different  
 93249 historical versions of *echo* that were considered for standardization varied in incompatible ways.

93250 The BSD *echo* checked the first argument for the string *-n* which caused it to suppress the  
 93251 <newline> that would otherwise follow the final argument in the output.

93252 The System V *echo* treated all arguments as strings to be written, but allowed escape sequences  
 93253 within them, as described for XSI implementations in the OPERANDS section, including *\c* to  
 93254 suppress a trailing <newline>.

93255 Thus the IEEE Std 1003.2-1992 standard said that the behavior was implementation-defined if  
 93256 the first operand is *-n* or if any of the operands contain a <backslash> character. It also specified  
 93257 that the *echo* utility does not support Utility Syntax Guideline 10 because historical applications  
 93258 depended on *echo* to echo *all* of its arguments, except for the *-n* first argument in the BSD  
 93259 version.

93260 The Single UNIX Specification, Version 1 required the System V behavior, and this became the  
 93261 XSI requirement when Version 2 and POSIX.2 were merged with POSIX.1 to form the joint  
 93262 IEEE Std 1003.1-2001 / Single UNIX Specification, Version 3 standard.

93263 This standard now treats a first operand of *-e* or *-E* the same as *-n* in recognition that support  
 93264 for them has become more widespread in non-XSI implementations. Where supported, *-e*  
 93265 enables processing of escape sequences in the remaining operands (in situations where it is  
 93266 disabled by default), and *-E* disables it (in situations where it is enabled by default). A first  
 93267 operand containing a combination of these three letters, in the same manner as option grouping,  
 93268 also results in implementation-defined behavior.

#### 93269 FUTURE DIRECTIONS

93270 None.

#### 93271 SEE ALSO

93272 *printf*

93273 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

#### 93274 CHANGE HISTORY

93275 First released in Issue 2.

93276 **Issue 5**93277  
93278

In the OPTIONS section, the last sentence is changed to indicate that implementations “do not” support any options; in the previous issue this said “need not”.

93279 **Issue 6**93280  
93281

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

93282  
93283  
93284

- A set of character sequences is defined as *string* operands.
- *LC\_CTYPE* is added to the list of environment variables affecting *echo*.
- In the OPTIONS section, implementations shall not support any options.

93285  
93286

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/21 is applied, so that the *echo* utility can accommodate historical BSD behavior.

93287 **Issue 7**

93288

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

93289 **Issue 8**

93290

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

93291  
93292  
93293

Austin Group Defect 1222 is applied, making the results implementation-defined, on systems that are not XSI-conformant, if the first operand consists of a '-' followed by one or more characters from the set {'e', 'E', 'n'}.

93294 **NAME**

93295 ed — edit text

93296 **SYNOPSIS**93297 ed [-p *string*] [-s] [*file*]93298 **DESCRIPTION**

93299 The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*. In  
 93300 command mode the input characters shall be interpreted as commands, and in input mode they  
 93301 shall be interpreted as text. See the EXTENDED DESCRIPTION section.

93302 If an operand is '-', the results are unspecified.

93303 **OPTIONS**

93304 The *ed* utility shall conform to XBD [Section 12.2](#) (on page 215), except for the unspecified usage  
 93305 of '-'.

93306 The following options shall be supported:

93307 **-p** *string* Use *string* as the prompt string when in command mode. By default, there shall be  
 93308 no prompt string.

93309 **-s** Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the '!'  
 93310 prompt after a *!command*.

93311 **OPERANDS**

93312 The following operand shall be supported:

93313 *file* If the *file* argument is given, *ed* shall perform the effect of an **e** command on the  
 93314 pathname *file* before accepting commands from the standard input, except that *file*  
 93315 can contain a <newline>, even though this is not possible for the argument to the **e**  
 93316 command.

93317 **STDIN**

93318 The standard input shall be a text file consisting of commands, as described in the EXTENDED  
 93319 DESCRIPTION section.

93320 **INPUT FILES**

93321 The input files shall be text files.

93322 **ENVIRONMENT VARIABLES**93323 The following environment variables shall affect the execution of *ed*:

93324 **HOME** Determine the pathname of the user's home directory.

93325 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 93326 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 93327 variables used to determine the values of locale categories.)

93328 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 93329 internationalization variables.

93330 **LC\_COLLATE**

93331 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 93332 character collating elements within regular expressions.

93333 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 93334 characters (for example, single-byte as opposed to multi-byte characters in  
 93335 arguments and input files) and the behavior of character classes within regular  
 93336 expressions.

- 93337 *LC\_MESSAGES*
- 93338 Determine the locale that should be used to affect the format and contents of
- 93339 diagnostic messages written to standard error and informative messages written to
- 93340 standard output.
- 93341 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 93342 **ASYNCHRONOUS EVENTS**
- 93343 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS
- 93344 section in [Section 1.4](#), on page 2462) with the following exceptions:
- 93345 **SIGINT** The *ed* utility shall interrupt its current activity, write the string "?\n" to standard
- 93346 output, and return to command mode (see the EXTENDED DESCRIPTION
- 93347 section).
- 93348 **SIGHUP** If the buffer is not empty and the buffer change flag is currently set to either
- 93349 **changed** or **changed-and-warned** (see the EXTENDED DESCRIPTION section),
- 93350 the *ed* utility shall attempt to write a copy of the buffer in a file. First, the file
- 93351 named **ed.hup** in the current directory shall be used; if that fails, the file named
- 93352 **ed.hup** in the directory named by the *HOME* environment variable shall be used.
- 93353 In any case, the *ed* utility shall exit without writing the file to the currently
- 93354 remembered pathname and without returning to command mode.
- 93355 **SIGQUIT** The *ed* utility shall ignore this event.
- 93356 **STDOUT**
- 93357 Various editing commands and the prompting feature (see **-p**) write to standard output, as
- 93358 described in the EXTENDED DESCRIPTION section.
- 93359 **STDERR**
- 93360 The standard error shall be used for diagnostic messages and may be used for warning
- 93361 messages.
- 93362 **OUTPUT FILES**
- 93363 The output files shall be text files whose formats are dependent on the editing commands given.
- 93364 **EXTENDED DESCRIPTION**
- 93365 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have
- 93366 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.
- 93367 The *ed* utility shall keep track of whether the buffer has been modified. This shall be maintained
- 93368 as if via a tri-state internal flag with the state values **unchanged**, **changed**, and **changed-and-**
- 93369 **warned**, which is:
- 93370 • Initially set to **unchanged**
  - 93371 • Set to **changed** by any command that modifies the buffer
  - 93372 • Set to **unchanged** by an **e** or **E** command that reloads (or empties) the buffer, or a **w**
  - 93373 command that writes the entire buffer
  - 93374 • Set to either **changed-and-warned** or **unchanged** by an **e** or **q** command that warns an
  - 93375 attempt was made to destroy the editor buffer
- 93376 A command that makes changes to the buffer in such a way that its contents are the same after
- 93377 the command (for example **s/a/a/**) shall be considered to have modified the buffer, unless
- 93378 explicitly stated otherwise. In the remainder of the description, this flag is referred to as the
- 93379 *buffer change flag*.
- 93380 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a



93381 single-character *command*, possibly followed by parameters to that command. These addresses  
 93382 specify one or more lines in the buffer. Every command that requires addresses has default  
 93383 addresses, so that the addresses very often can be omitted. If the `-p` option is specified, the  
 93384 prompt string shall be written to standard output before each command is read.

93385 In general, only one command can appear on a line. Certain commands allow text to be input.  
 93386 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be  
 93387 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.  
 93388 Input mode is terminated by entering a line consisting of two characters: a <period> ( ' . ' )  
 93389 followed by a <newline>. This line is not considered part of the input text.

### 93390 **Regular Expressions in ed**

93391 The *ed* utility shall support basic regular expressions, as described in XBD [Section 9.3](#) (on page  
 93392 181). Since regular expressions in *ed* are always matched against single lines (excluding the  
 93393 terminating <newline> characters), never against any larger section of text, there is no way for a  
 93394 regular expression to match a <newline>.

93395 A null RE shall be equivalent to the last RE encountered.

93396 Regular expressions are used in addresses to specify lines, and in some commands (for example,  
 93397 the *s* substitute command) to specify portions of a line to be substituted.

93398 The start and end of a regular expression (RE) are marked by a delimiter character (although in  
 93399 some circumstances the end delimiter can be omitted). In addresses, the delimiter is either  
 93400 <slash> or <question-mark>. In commands, other characters can be used as the delimiter, as  
 93401 specified in the description of the command. Within the RE (as an *ed* extension to the BRE  
 93402 syntax), the delimiter shall not terminate the RE if it is the second character of an escape  
 93403 sequence (see XBD [Section 9.1](#), on page 179) and the escaped delimiter shall be treated as that  
 93404 literal character in the RE (losing any special meaning it would have had if it was not used as the  
 93405 delimiter and was not escaped). In addition, the delimiter character shall not terminate the RE  
 93406 when it appears within a bracket expression, and shall have its normal meaning in the bracket  
 93407 expression. For example, the command `"g%[%]p"` is equivalent to `"g/[ ]/p"`, and the  
 93408 command `"s-[0-9]--g"` is equivalent to `"s/[0-9]/g"`.

### 93409 **Addresses in ed**

93410 Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a  
 93411 command. The current line number is the address of the current line. If the edit buffer is not  
 93412 empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

93413 Addresses shall be constructed as follows:

- 93414 1. The <period> character ( ' . ' ) shall address the current line.
- 93415 2. The <dollar-sign> character ( ' \$ ' ) shall address the last line of the edit buffer.
- 93416 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 93417 4. The <apostrophe>-*x* character pair ( " ' x " ) shall address the line marked with the mark  
 93418 name character *x*, which shall be a lowercase letter from the portable character set. It shall  
 93419 be an error if the character has not been set to mark a line or if the line that was marked is  
 93420 not currently present in the edit buffer.
- 93421 5. A BRE enclosed by <slash> characters ( ' / ' ) shall address the first line found by  
 93422 searching forwards from the line following the current line toward the end of the edit  
 93423 buffer and stopping at the first line for which the line excluding the terminating  
 93424 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of

93425 <slash> characters shall address the next line for which the line excluding the terminating  
 93426 <newline> matches the last BRE encountered. In addition, the second <slash> can be  
 93427 omitted at the end of a command line. Within the BRE, a <backslash>-<slash> pair ("`\`")  
 93428 shall represent a literal <slash> instead of the BRE delimiter. If necessary, the search shall  
 93429 wrap around to the beginning of the buffer and continue up to and including the current  
 93430 line, so that the entire buffer is searched.

93431 6. A BRE enclosed by <question-mark> characters ('?') shall address the first line found by  
 93432 searching backwards from the line preceding the current line toward the beginning of the  
 93433 edit buffer and stopping at the first line for which the line excluding the terminating  
 93434 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of  
 93435 <question-mark> characters ("??") shall address the previous line for which the line  
 93436 excluding the terminating <newline> matches the last BRE encountered. In addition, the  
 93437 second <question-mark> can be omitted at the end of a command line. Within the BRE, a  
 93438 <backslash>-<question-mark> pair ("\?") shall represent a literal <question-mark>  
 93439 instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the  
 93440 buffer and continue up to and including the current line, so that the entire buffer is  
 93441 searched.

93442 7. A <plus-sign> ('+') or <hyphen-minus> character ('-') followed by a decimal number  
 93443 shall address the current line plus or minus the number. A <plus-sign> or <hyphen-  
 93444 minus> character not followed by a decimal number shall address the current line plus or  
 93445 minus 1.

93446 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.  
 93447 Address offsets are constructed as follows:

- 93448 • A <plus-sign> or <hyphen-minus> character followed by a decimal number shall add or  
 93449 subtract, respectively, the indicated number of lines to or from the address. A <plus-sign>  
 93450 or <hyphen-minus> character not followed by a decimal number shall add or subtract 1 to  
 93451 or from the address.
- 93452 • A decimal number shall add the indicated number of lines to the address.

93453 It shall not be an error for an intermediate address value to be less than zero or greater than the  
 93454 last line in the edit buffer. It shall be an error for the final address value to be less than zero or  
 93455 greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a  
 93456 matching line.

93457 Commands accept zero, one, or two addresses. If one or more addresses are provided to a  
 93458 command that requires zero addresses, it shall be an error. Otherwise, if more than the  
 93459 maximum number of accepted addresses are provided to a command, the addresses shall be  
 93460 evaluated from first to last and then discarded, until the maximum number of accepted  
 93461 addresses for that command remain.

93462 Addresses shall be separated from each other by a <comma> (',') or <semicolon> character  
 93463 (';'). In the case of a <semicolon> separator, the current line ('.') shall be set to the first  
 93464 address, and only then shall the second address be calculated. This feature can be used to  
 93465 determine the starting line for forwards and backwards searches; see rules 5. and 6.

93466 Addresses can be omitted on either side of the <comma> or <semicolon> separator, in which  
 93467 case the resulting address pairs shall be as follows:

93468  
93469  
93470  
93471  
93472  
93473  
93474

Specified	Resulting
,	1 , \$
, addr	1 , addr
addr ,	addr , addr
;	. ; \$
; addr	. ; addr
addr ;	addr ; addr

93475  
93476  
93477  
93478

If an address is omitted between two separators, the rule shall be applied to the first separator and the resulting second address shall be used as the first address for the second separator. For example, with the address list ", , " the first ', ' becomes "1, \$" and the '\$' is treated as the first address for the second ', ', resulting in "1, \$, \$".

93479  
93480

Any <blank> characters included between addresses, address separators, or address offsets shall be ignored.

93481

### Commands in ed

93482  
93483  
93484

In the following list of *ed* commands, the default addresses are shown in parentheses. The number of addresses shown in the default shall be the number expected by the command. The parentheses are not part of the address; they show that the given addresses are the default.

93485  
93486  
93487  
93488  
93489  
93490  
93491  
93492  
93493

It is generally invalid for more than one command to appear on a line. However, any command (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but it is unspecified whether the suffix writes the current line again in the requested format or whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l** suffix) shall either write just the current line or write it twice—once as specified for **p** and once as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

93494  
93495  
93496

Each address component can be preceded by zero or more <blank> characters. The command letter can be preceded by zero or more <blank> characters. If a suffix letter (**l**, **n**, or **p**) is given, the application shall ensure that it immediately follows the command.

93497  
93498

The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the command letter by one or more <blank> characters.

93499  
93500

If the buffer change flag is currently set to **changed**, *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands. The *ed* utility shall write the string:

93501

"?\n"

93502  
93503  
93504  
93505  
93506

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the buffer change flag set to either **changed-and-warned** or **unchanged** and the current line number unchanged. If another **e** or **q** command is then attempted with no intervening command that sets the buffer change flag to **changed**, it shall take effect.

93507  
93508

If a terminal disconnect (see XBD Chapter 11 (on page 199), Modem Disconnect and Closing a Device Terminal), is detected:

93509  
93510

- If accompanied by a SIGHUP signal, the *ed* utility shall operate as described in the ASYNCHRONOUS EVENTS section for a SIGHUP signal.

- 93511           • If not accompanied by a SIGHUP signal, the *ed* utility shall act as if an end-of-file had been  
93512 detected on standard input.

93513 If an end-of-file is detected on standard input:

- 93514           • If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command  
93515 mode. It is unspecified if any partially entered lines (that is, input text without a  
93516 terminating <newline>) are discarded from the input text.
- 93517           • If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

93518 In the following commands, if a closing delimiter would be the last character before a <newline>  
93519 then that character can be omitted with the behavior shown:

- 93520           • For the **g** and **v** commands the addressed line(s) shall be written as if a closing delimiter  
93521 followed by a **p** were appended to the command.
- 93522           • For the **G** and **V** commands no additional action shall be taken.
- 93523           • For the **s** command, only the closing delimiter of the replacement can be omitted, in which  
93524 case the result of the substitution shall be written as if the **p** flag were appended.
- 93525           • For the null command, the addressed line(s) shall be written as if the closing delimiter  
93526 were appended.

93527 For example, the following pairs of commands are equivalent:

```
93528 s/s1/s2    s/s1/s2/p
93529 g/s1       g/s1/p
93530 ?s1       ?s1?
```

93531 If an invalid command is entered, *ed* shall write the string:

93532 "?\n"

93533 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to  
93534 standard output and shall continue in command mode with the current line number unchanged.

### 93535 **Append Command**

```
93536 Synopsis:  (.) a
93537             <text>
93538             .
```

93539 The **a** command shall read the given text and append it after the addressed line; the current line  
93540 number shall become the address of the last inserted line or, if there were none, the addressed  
93541 line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at  
93542 the beginning of the buffer. If <text> is empty (that is, the terminating ' .' immediately follows  
93543 the 'a'), the buffer change flag shall not be altered.

### 93544 **Change Command**

```
93545 Synopsis:  (.,.) c
93546             <text>
93547             .
```

93548 The **c** command shall delete the addressed lines, then accept input text that replaces these lines;  
93549 the current line shall be set to the address of the last line input; or, if there were none, at the line  
93550 after the last line deleted; if the lines deleted were originally at the end of the buffer, the current  
93551 line number shall be set to the address of the new last line; if no lines remain in the buffer, the

93552 current line number shall be set to zero.

### 93553 **Delete Command**

93554 *Synopsis:* ( , . ) d

93555 The **d** command shall delete the addressed lines from the buffer. The address of the line after the  
 93556 last line deleted shall become the current line number; if the lines deleted were originally at the  
 93557 end of the buffer, the current line number shall be set to the address of the new last line; if no  
 93558 lines remain in the buffer, the current line number shall be set to zero.

### 93559 **Edit Command**

93560 *Synopsis:* e [*file*]

93561 The **e** command shall delete the entire contents of the buffer and then read in the file named by  
 93562 the pathname *file*. The current line number shall be set to the address of the last line of the  
 93563 buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see  
 93564 the **f** command). If the pathname names a file that does not exist and the buffer change flag is  
 93565 currently set to **unchanged**, it is unspecified whether this is treated as an error, or whether the  
 93566 resulting buffer is emptied and a warning is written to standard error instead of writing the byte  
 93567 count to standard out. The number of bytes read shall be written to standard output, unless the  
 93568 **-s** option was specified, in the following format:

93569 "%d\n", <number of bytes read>

93570 The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**,  
 93571 and **w** commands. If *file* is replaced by '!', the rest of the line shall be taken to be a shell  
 93572 command line whose output is to be read. Such a shell command line shall not be remembered  
 93573 as the current *file*. All marks shall be discarded upon the completion of a successful **e** command.  
 93574 If the buffer change flag is currently set to **changed**, the user shall be warned, as described  
 93575 previously.

### 93576 **Edit Without Checking Command**

93577 *Synopsis:* E [*file*]

93578 The **E** command shall possess all properties and restrictions of the **e** command except that the  
 93579 editor shall not check the current state of the buffer change flag.

### 93580 **Filename Command**

93581 *Synopsis:* f [*file*]

93582 If *file* is given, the **f** command shall change the currently remembered pathname to *file*, whether  
 93583 or not *file* names an existing file; whether the name is changed or not, it shall then write the  
 93584 (possibly new) currently remembered pathname to the standard output in the following format:

93585 "%s\n", <pathname>

93586 The current line number shall be unchanged.

93587 **Global Command**93588 *Synopsis:* (1, \$)g/RE/command list

93589 In the **g** command, the first step shall be to mark every line for which the line excluding the  
 93590 terminating <newline> matches the given RE. Then, going sequentially from the beginning of  
 93591 the file to the end of the file, the given *command list* shall be executed for each marked line, with  
 93592 the current line number set to the address of that line. Any line modified by the *command list*  
 93593 shall be unmarked. When the **g** command completes, the current line number shall have the  
 93594 value assigned by the last command in the *command list*. If there were no matching lines, the  
 93595 current line number shall not be changed. A single command or the first of a list of commands  
 93596 shall appear on the same line as the global command. All lines of a multi-line list except the last  
 93597 line shall be ended with a <backslash> preceding the terminating <newline>; the **a**, **i**, and **c**  
 93598 commands and associated input are permitted. The '.' terminating input mode can be omitted  
 93599 if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p**  
 93600 command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined  
 93601 results. Any character other than <backslash>, <space>, or <newline> can be used instead of a  
 93602 <slash> to delimit the RE. Within the RE, in certain circumstances the RE delimiter can be used  
 93603 as a literal character; see [Regular Expressions in ed](#) (on page 2817).

93604 **Interactive Global Command**93605 *Synopsis:* (1, \$)G/RE/

93606 In the **G** command, the first step shall be to mark every line for which the line excluding the  
 93607 terminating <newline> matches the given RE. Then, for every such line, that line shall be  
 93608 written, the current line number shall be set to the address of that line, and any one command  
 93609 (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline>  
 93610 shall act as a null command (causing no action to be taken on the current line); an '&' shall  
 93611 cause the re-execution of the most recent non-null command executed within the current  
 93612 invocation of **G**. Note that the commands input as part of the execution of the **G** command can  
 93613 address and affect any lines in the buffer. Any line modified by the command shall be  
 93614 unmarked. The final value of the current line number shall be the value set by the last command  
 93615 successfully executed. (Note that the last command successfully executed shall be the **G**  
 93616 command itself if a command fails or the null command is specified.) If there were no matching  
 93617 lines, the current line number shall not be changed. The **G** command can be terminated by a  
 93618 SIGINT signal. Any character other than <backslash>, <space>, or <newline> can be used  
 93619 instead of a <slash> to delimit the RE. Within the RE, in certain circumstances the RE delimiter  
 93620 can be used as a literal character; see [Regular Expressions in ed](#) (on page 2817).

93621 **Help Command**93622 *Synopsis:* h

93623 The **h** command shall write a short message to standard output that explains the reason for the  
 93624 most recent '?' notification. The current line number shall be unchanged.

93625 **Help-Mode Command**93626 *Synopsis:*     H

93627 The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command)  
 93628 shall be written to standard output for all subsequent '?' notifications. The **H** command  
 93629 alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on,  
 93630 the **H** command also explains the previous '?' notification, if there was one. The current line  
 93631 number shall be unchanged.

93632 **Insert Command**

93633 *Synopsis:*     ( . ) i  
 93634                 <text>  
 93635                 .

93636 The **i** command shall insert the given text before the addressed line; the current line is set to the  
 93637 last inserted line or, if there was none, to the addressed line. This command differs from the **a**  
 93638 command only in the placement of the input text. Address 0 shall be valid for this command; it  
 93639 is unspecified whether it causes the inserted text to be placed at the beginning of the buffer or it  
 93640 is interpreted as if address 1 were specified. (These two allowed behaviors differ in the case that  
 93641 the buffer is empty.) If <text> is empty (that is, the terminating '.' immediately follows the  
 93642 'i'), the buffer change flag shall not be altered.

93643 **Join Command**93644 *Synopsis:*     ( . , .+1 ) j

93645 The **j** command shall join contiguous lines by removing the appropriate <newline> characters. If  
 93646 exactly one address is given, this command shall do nothing. If lines are joined, the current line  
 93647 number shall be set to the address of the joined line; otherwise, the current line number shall be  
 93648 unchanged.

93649 **Mark Command**93650 *Synopsis:*     ( . ) k x

93651 The **k** command shall mark the addressed line with name *x*, which the application shall ensure  
 93652 is a lowercase letter from the portable character set. The address "'x" shall then refer to this  
 93653 line; the current line number shall be unchanged.

93654 **List Command**93655 *Synopsis:*     ( . , . ) l

93656 The **l** command shall write to standard output the addressed lines in a visually unambiguous  
 93657 form. The characters listed in XBD Table 5-1 (on page 113) ('\ ', '\a', '\b', '\f', '\r',  
 93658 '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not  
 93659 applicable. Non-printable characters not in the table shall be written as one three-digit octal  
 93660 number (with a preceding <backslash> character) for each byte in the character (most significant  
 93661 byte first).

93662 Long lines shall be folded, with the point of folding indicated by <newline> preceded by a  
 93663 <backslash>; the length at which folding occurs is unspecified, but should be appropriate for the  
 93664 output device. The end of each line shall be marked with a '\$', and '\$' characters within the  
 93665 text shall be written with a preceding <backslash>. An **l** command can be appended to any  
 93666 other command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the

93667 address of the last line written.

### 93668 **Move Command**

93669 *Synopsis:* ( , . ) *address*

93670 The **m** command shall reposition the addressed lines after the line addressed by *address*.  
93671 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning  
93672 of the buffer. It shall be an error if address *address* falls within the range of moved lines. The  
93673 current line number shall be set to the address of the last line moved.

### 93674 **Number Command**

93675 *Synopsis:* ( , . ) *n*

93676 The **n** command shall write to standard output the addressed lines, preceding each line by its  
93677 line number and a <tab>; the current line number shall be set to the address of the last line  
93678 written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

### 93679 **Print Command**

93680 *Synopsis:* ( , . ) *p*

93681 The **p** command shall write to standard output the addressed lines; the current line number shall  
93682 be set to the address of the last line written. The **p** command can be appended to any command  
93683 other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

### 93684 **Prompt Command**

93685 *Synopsis:* **P**

93686 The **P** command shall cause *ed* to prompt with an <asterisk> ('\*') (or *string*, if **-p** is specified)  
93687 for all subsequent commands. The **P** command alternatively shall turn this mode on and off; it  
93688 shall be initially on if the **-p** option is specified; otherwise, off. The current line number shall be  
93689 unchanged.

### 93690 **Quit Command**

93691 *Synopsis:* **q**

93692 The **q** command shall cause *ed* to exit. If the buffer change flag is currently set to **changed**, the  
93693 user shall be warned, as described previously.

### 93694 **Quit Without Checking Command**

93695 *Synopsis:* **Q**

93696 The **Q** command shall cause *ed* to exit without checking the current state of the buffer change  
93697 flag.



93698 **Read Command**93699 *Synopsis:* (\$) **r** [*file*]

93700 The **r** command shall read in the file named by the pathname *file* and append it after the  
 93701 addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be  
 93702 used (see the **e** and **f** commands). The currently remembered pathname shall not be changed  
 93703 unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file  
 93704 to be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the  
 93705 number of bytes read shall be written to standard output in the following format:

93706 "%d\n", &lt;number of bytes read&gt;

93707 The current line number shall be set to the address of the last line read in. If *file* is replaced by  
 93708 '!', the rest of the line shall be taken to be a shell command line whose output is to be read.  
 93709 Such a shell command line shall not be remembered as the current pathname.

93710 If the number of bytes read is 0 it is unspecified whether the buffer change flag is set to **changed**  
 93711 or left unaltered.

93712 **Substitute Command**93713 *Synopsis:* (.,.) **s**/RE/replacement/flags

93714 The **s** command shall search each addressed line for an occurrence of the specified RE and  
 93715 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the  
 93716 following description of the **g** suffix. Any character other than <backslash>, <space>, or  
 93717 <newline> can be used instead of a <slash> to delimit the RE and the replacement. Within the  
 93718 RE, in certain circumstances the RE delimiter can be used as a literal character; see [Regular](#)  
 93719 [Expressions in ed](#) (on page 2817). Within the replacement, the delimiter shall not terminate the  
 93720 replacement if it is the second character of an escape sequence (see [XBD Section 9.1](#), on page 179)  
 93721 and the escaped delimiter shall be treated as that literal character in the replacement (losing any  
 93722 special meaning it would have had if it was not used as the delimiter and was not escaped). It  
 93723 shall be an error if the substitution fails on every addressed line. The current line shall be set to  
 93724 the address of the last line on which a substitution occurred.

93725 An unescaped <ampersand> ('&') appearing in the replacement shall be replaced by the string  
 93726 matching the RE on the current line. As a more general feature, the characters '\n', where the  
 93727 <backslash> is unescaped and *n* is a digit, shall be replaced by the text matched by the  
 93728 corresponding back-reference expression. If the corresponding back-reference expression does  
 93729 not match, then the characters '\n' shall be replaced by the empty string. When the character  
 93730 '%' is the only character in *replacement*, the *replacement* used in the most recent substitute  
 93731 command shall be used as *replacement* in the current substitute command; if there was no  
 93732 previous substitute command, the use of '%' in this manner shall be an error. The '%' shall lose  
 93733 its special meaning when it is in a replacement string of more than one character or is escaped. It  
 93734 is unspecified what special meaning is given to any character other than <backslash>, '&', '%',  
 93735 or digits.

93736 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the  
 93737 <newline> in the *replacement* by preceding it by <backslash>. Such substitution cannot be done  
 93738 as part of a **g** or **v** *command list*. The current line number shall be set to the address of the last  
 93739 line on which a substitution is performed. If no substitution is performed, the current line  
 93740 number shall be unchanged. If a line is split, a substitution shall be considered to have been  
 93741 performed on each of the new lines for the purpose of determining the new current line number.  
 93742 A substitution shall be considered to have been performed even if the replacement string is  
 93743 identical to the string that it replaces.

- 93744 The application shall ensure that the value of *flags* is zero or more of:
- 93745 *count* Substitute for the *count*th occurrence only of the RE found on each addressed line.
- 93746 **g** Globally substitute for all non-overlapping instances of the RE rather than just the first  
93747 one. If both **g** and *count* are specified, the results are unspecified.
- 93748 **l** Write to standard output the final line in which a substitution was made. The line shall  
93749 be written in the format specified for the **l** command.
- 93750 **n** Write to standard output the final line in which a substitution was made. The line shall  
93751 be written in the format specified for the **n** command.
- 93752 **p** Write to standard output the final line in which a substitution was made. The line shall  
93753 be written in the format specified for the **p** command.

### 93754 **Copy Command**

93755 *Synopsis:* (*, .*)*taddress*

93756 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines  
93757 shall be placed after address *address* (which can be 0); the current line number shall be set to the  
93758 address of the last line added.

### 93759 **Undo Command**

93760 *Synopsis:* *u*

93761 The **u** command shall nullify the effect of the most recent command that modified anything in  
93762 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes  
93763 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no  
93764 changes were made by the global command (such as with **g/RE/p**), the **u** command shall have  
93765 no effect. The current line number shall be set to the value it had immediately before the  
93766 command being undone started.

### 93767 **Global Non-Matched Command**

93768 *Synopsis:* (1, \$)*v/RE/command list*

93769 This command shall be equivalent to the global command **g** except that the lines that are marked  
93770 during the first step shall be those for which the line excluding the terminating <newline> does  
93771 not match the RE.

### 93772 **Interactive Global Not-Matched Command**

93773 *Synopsis:* (1, \$)*V/RE/*

93774 This command shall be equivalent to the interactive global command **G** except that the lines that  
93775 are marked during the first step shall be those for which the line excluding the terminating  
93776 <newline> does not match the RE.

93777 **Write Command**93778 *Synopsis:* (1, \$)w [*file*]

93779 The **w** command shall write the addressed lines into the file named by the pathname *file*. The  
 93780 command shall create the file, if it does not exist, or shall replace the contents of the existing file.  
 93781 The currently remembered pathname shall not be changed unless there is no remembered  
 93782 pathname. If no pathname is given, the currently remembered pathname, if any, shall be used  
 93783 (see the **e** and **f** commands); the current line number shall be unchanged. If the command is  
 93784 successful, the number of bytes written shall be written to standard output, unless the **-s** option  
 93785 was specified, in the following format:

93786 "%d\n", &lt;number of bytes written&gt;

93787 If *file* begins with '!', the rest of the line shall be taken to be a shell command line whose  
 93788 standard input shall be the addressed lines. Such a shell command line shall not be remembered  
 93789 as the current pathname. This usage of the **w** command with '!' shall not alter the buffer  
 93790 change flag; thus, this alone shall not prevent the warning to the user if an attempt is made to  
 93791 destroy the editor buffer via the **e** or **q** commands.

93792 **Line Number Command**93793 *Synopsis:* (\$) =

93794 The line number of the addressed line shall be written to standard output in the following  
 93795 format:

93796 "%d\n", &lt;line number&gt;

93797 The current line number shall be unchanged by this command.

93798 **Shell Escape Command**93799 *Synopsis:* !*command*

93800 The remainder of the line after the '!' shall be sent to the command interpreter to be  
 93801 interpreted as a shell command line. Within the text of that shell command line, the unescaped  
 93802 character '%' shall be replaced with the remembered pathname; if a '!' appears as the first  
 93803 character of the command, it shall be replaced with the text of the previous shell command  
 93804 executed via '!'. Thus, "!!" shall repeat the previous !*command*. If any replacements of '%' or  
 93805 '!' are performed, the modified line shall be written to the standard output before *command* is  
 93806 executed. The ! command shall write:

93807 "!\n"

93808 to standard output upon completion, unless the **-s** option is specified. The current line number  
 93809 shall be unchanged.

93810 **Null Command**93811 *Synopsis:* (.+1)

93812 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall  
 93813 be equivalent to "+1p". The current line number shall be set to the address of the written line.

93814 **EXIT STATUS**

93815 The following exit values shall be returned:

93816 0 Successful completion without any file or command errors.

93817 >0 An error occurred.

### 93818 CONSEQUENCES OF ERRORS

93819 When an error in the input script is encountered, or when an error is detected that is a  
93820 consequence of the data (not) present in the file or due to an external condition such as a read or  
93821 write error:

93822 • If the standard input is a terminal device file, all input shall be flushed, and a new  
93823 command read.

93824 • If the standard input is not a terminal device file, *ed* shall behave as described under  
93825 CONSEQUENCES OF ERRORS in [Section 1.4](#) (on page 2462).

### 93826 APPLICATION USAGE

93827 Because of the extremely terse nature of the default error messages, the prudent script writer  
93828 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some  
93829 clue as to the cause is made available.

93830 In earlier versions of this standard, an obsolescent `-` option was described. This is no longer  
93831 specified. Applications should use the `-s` option. Using `-` as a *file* operand now produces  
93832 unspecified results. This allows implementations to continue to support the former required  
93833 behavior.

### 93834 EXAMPLES

93835 None.

### 93836 RATIONALE

93837 The initial description of this utility was adapted from the SVID. It contains some features not  
93838 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and  
93839 BSD *ed* utilities include, but need not be limited to:

- 93840 • The BSD `-` option does not suppress the `!'` prompt after a `!` command.
- 93841 • BSD does not support the special meanings of the `'%` and `!'` characters within a `!`  
93842 command.
- 93843 • BSD does not support the *addresses* `';` and `',`.
- 93844 • BSD allows the command/suffix pairs `pp`, `ll`, and so on, which are unspecified in this  
93845 volume of POSIX.1-2024.
- 93846 • BSD does not support the `!'` character part of the `e`, `r`, or `w` commands.
- 93847 • A failed `g` command in BSD sets the line number to the last line searched if there are no  
93848 matches.
- 93849 • BSD does not default the *command list* to the `p` command.
- 93850 • BSD does not support the `G`, `h`, `H`, `n`, or `V` commands.
- 93851 • On BSD, if there is no inserted text, the insert command changes the current line to the  
93852 referenced line `-1`; that is, the line before the specified line.
- 93853 • On BSD, the `j` command with only a single address changes the current line to that  
93854 address.
- 93855 • BSD does not support the `P` command; moreover, in BSD it is synonymous with the `p`  
93856 command.

- 93857           • BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
- 93858           • The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in
- 93859           this volume of POSIX.1-2024.
  
- 93860           The **-s** option was added to allow the functionality of the removed **-** option in a manner
- 93861           compatible with the Utility Syntax Guidelines.
  
- 93862           In early proposals there was a limit, {ED\_FILE\_MAX}, that described the historical limitations of
- 93863           some *ed* utilities in their handling of large files; some of these have had problems with files
- 93864           larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a
- 93865           *split* command in this volume of POSIX.1-2024. Since this limit was removed, this volume of
- 93866           POSIX.1-2024 requires that implementations document the file size limits imposed by *ed* in the
- 93867           conformance document. The limit {ED\_LINE\_MAX} was also removed; therefore, the global
- 93868           limit {LINE\_MAX} is used for input and output lines.
  
- 93869           The manner in which the **l** command writes non-printable characters was changed to avoid the
- 93870           historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous
- 93871           because most terminals simply replace overstruck characters, making the **l** format not useful for
- 93872           its intended purpose of unambiguously understanding the content of the line. The historical
- 93873           <backslash>-escapes were also ambiguous. (The string "a\0011" could represent a line
- 93874           containing those six characters or a line containing the three characters 'a', a byte with a binary
- 93875           value of 1, and a 1.) In the format required here, a <backslash> appearing in the line is written as
- 93876           "\" so that the output is truly unambiguous. The method of marking the ends of lines was
- 93877           adopted from the *ex* editor and is required for any line ending in <space> characters; the '\$' is
- 93878           placed on all lines so that a real '\$' at the end of a line cannot be misinterpreted.
  
- 93879           Earlier versions of this standard allowed for implementations with bytes other than eight bits,
- 93880           but this has been modified in this version.
  
- 93881           The description of how a NUL is written was removed. The NUL character cannot be in text
- 93882           files, and this volume of POSIX.1-2024 should not dictate behavior in the case of undefined,
- 93883           erroneous input.
  
- 93884           Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands
- 93885           are not patterns.
  
- 93886           Early proposals stated that the **-p** option worked only when standard input was associated with
- 93887           a terminal device. This has been changed to conform to historical implementations, thereby
- 93888           allowing applications to interpose themselves between a user and the *ed* utility.
  
- 93889           The form of the substitute command that uses the **n** suffix was limited in some historical
- 93890           documentation (where this was described incorrectly as “backreferencing”). This limit has been
- 93891           omitted because there is no reason why an editor processing lines of {LINE\_MAX} length should
- 93892           have this restriction. The command **s/x/X/2047** should be able to substitute the 2 047th occurrence
- 93893           of 'x' on a line.
  
- 93894           The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made
- 93895           unspecified because BSD-based systems allow this, whereas System V does not.
  
- 93896           Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file
- 93897           have been deleted. Since this volume of POSIX.1-2024 refers to the **q** command in this instance,
- 93898           such behavior is not allowed.
  
- 93899           Some historical implementations returned exit status zero even if command errors had occurred;
- 93900           this is not allowed by this volume of POSIX.1-2024.
  
- 93901           Some historical implementations contained a bug that allowed a single <period> to be entered in

93902 input mode as `<backslash> <period> <newline>`. This is not allowed by *ed* because there is no  
 93903 description of escaping any of the characters in input mode; `<backslash>` characters are entered  
 93904 into the buffer exactly as typed. The typical method of entering a single `<period>` has been to  
 93905 precede it with another character and then use the substitute command to delete that character.

93906 It is difficult under some modes of some versions of historical operating system terminal drivers  
 93907 to distinguish between an end-of-file condition and terminal disconnect. POSIX.1-2024 does not  
 93908 require implementations to distinguish between the two situations, which permits historical  
 93909 implementations of the *ed* utility on historical platforms to conform. Implementations are  
 93910 encouraged to distinguish between the two, if possible, and take appropriate action on terminal  
 93911 disconnect.

93912 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the  
 93913 start of the edit buffer. When the buffer was empty the command `.=` returned zero. POSIX.1-2024  
 93914 requires conformance to historical practice.

93915 For consistency with the **a** and **r** commands and better user functionality, the **i** command also  
 93916 accepts an address of 0. However, it is unspecified if `0i` is treated as `1i` (which will fail if the  
 93917 buffer is empty), or means insert at the beginning of the buffer (which will succeed even if the  
 93918 buffer is empty). Earlier versions of this standard required address 0 for the **c** command to be  
 93919 treated as 1 also, but this requirement has been removed, though implementations are permitted  
 93920 to do this as an extension.

93921 All of the following are valid addresses:

93922	<code>+++</code>	Three lines after the current line.
93923	<code>/pattern/-</code>	One line before the next occurrence of pattern.
93924	<code>-2</code>	Two lines before the current line.
93925	<code>3 ---- 2</code>	Line one (note the intermediate negative address).
93926	<code>1 2 3</code>	Line six.

93927 More than the maximum number of accepted addresses can be provided to commands taking  
 93928 addresses; for example, `"1,2,3,4,5p"` prints lines 4 and 5, because two is the maximum  
 93929 number of addresses accepted by the **print** command. This, in combination with the  
 93930 `<semicolon>` delimiter, permits users to create commands based on ordered patterns in the file.  
 93931 For example, the command `"3;/foo/;+2p"` will display the first line after line 3 that contains  
 93932 the pattern `foo`, plus the next two lines. Note that the address `"3;"` must still be evaluated before  
 93933 being discarded, because the search origin for the `"/foo/"` address depends on this.

93934 Historically, *ed* disallowed address chains, as discussed above, consisting solely of `<comma>` or  
 93935 `<semicolon>` separators; for example, `","` or `";;"` were considered an error. For  
 93936 consistency of address specification, this restriction is removed. The following table lists some of  
 93937 the address forms now possible:

	Address	Addr1	Addr2	Status	Comment
93938					
93939	7,	7	7	Historical	
93940	7,5,	5	5	Historical	
93941	7,5,9	5	9	Historical	
93942	7,9	7	9	Historical	
93943	7,+	7	.+	Historical	
93944	,	1	\$	Historical	
93945	,7	1	7	Extension	
93946	,,	\$	\$	Extension	
93947	,;	\$	\$	Extension	
93948	7;	7	7	Historical	
93949	7;5;	5	5	Historical	
93950	7;5;9	5	9	Historical	
93951	7;5,9	5	9	Historical	
93952	7;\$;4	\$	4	Historical	Valid, but erroneous.
93953	7;9	7	9	Historical	
93954	7;+	7	8	Historical	
93955	;	.	\$	Historical	
93956	;7	.	7	Extension	
93957	;;	\$	\$	Extension	
93958	;,	\$	\$	Extension	

93959 Historically, *ed* accepted the '^' character as an address, in which case it was identical to the  
 93960 <hyphen-minus> character. POSIX.1-2024 does not require or prohibit this behavior.

93961 Implementations are encouraged to set the buffer change flag to **changed-and-warned** when an  
 93962 **e** or **q** command warns that an attempt was made to destroy the editor buffer. Some existing  
 93963 implementations set it to **unchanged**, but this has the undesirable side-effect that a SIGHUP  
 93964 received after the warning is given does not write the buffer to **ed.hup**.

#### 93965 FUTURE DIRECTIONS

93966 If this utility is directed to create a new directory entry that contains any bytes that have the  
 93967 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 93968 error. A future version of this standard may require implementations to treat this as an error.

93969 A future version of this standard may require that the buffer change flag is set to **changed-and-**  
 93970 **warned** when an **e** or **q** command warns that an attempt was made to destroy the editor buffer.

#### 93971 SEE ALSO

93972 [Section 1.4](#) (on page 2462), *ex*, *sed*, *sh*, *vi*

93973 [XBD Table 5-1](#) (on page 113), [Chapter 8](#) (on page 167), [Section 9.3](#) (on page 181), [Chapter 11](#) (on  
 93974 page 199), [Section 12.2](#) (on page 215)

#### 93975 CHANGE HISTORY

93976 First released in Issue 2.

#### 93977 Issue 5

93978 In the OPTIONS section, the meaning of **-s** and **-** is clarified.

93979 A second FUTURE DIRECTION is added.

#### 93980 Issue 6

93981 The obsolescent single-minus form is removed.

93982 A second APPLICATION USAGE note is added.

- 93983 The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.
- 93984 The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition  
93985 of the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing  
93986 when end-of-file is detected and when terminal disconnect is detected.
- 93987 The normative text is reworded to avoid use of the term “must” for application requirements.
- 93988 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/22 is applied, adding the text: “Any line  
93989 modified by the *command list* shall be unmarked.” to the **G** command. This change corresponds  
93990 to a similar change made to the **g** command in the first version of this standard.
- 93991 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/7 is applied, removing text describing  
93992 behavior on systems with bytes consisting of more than eight bits.
- 93993 **Issue 7**
- 93994 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is  
93995 ‘-’.
- 93996 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.
- 93997 SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal  
93998 disconnect is detected (in Commands in *ed*).
- 93999 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 94000 SD5-XCU-ERN-135 is applied, removing some RATIONALE text that is no longer applicable.
- 94001 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0090 [584], XCU/TC2-2008/0091  
94002 [584], and XCU/TC2-2008/0092 [584] are applied.
- 94003 **Issue 8**
- 94004 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
94005 filenames containing any bytes that have the encoded value of a <newline> character.
- 94006 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 94007 Austin Group Defect 1130 is applied, removing the requirement for the **c** command to accept an  
94008 address of 0 and updating the information about address 0 in the RATIONALE section.
- 94009 Austin Group Defect 1131 is applied, changing the address 0 requirements for the **i** command.
- 94010 Austin Group Defect 1204 is applied, clarifying the behavior when a closing delimiter that  
94011 would be the last character before a <newline> is omitted.
- 94012 Austin Group Defect 1281 is applied, moving some text in the description of the **s** command and  
94013 changing it to use “shall”.
- 94014 Austin Group Defect 1298 is applied, changing the CONSEQUENCES OF ERRORS section.
- 94015 Austin Group Defect 1308 is applied, changing the **Addr2** value for address 7, + in the table of  
94016 address forms in the RATIONALE section.
- 94017 Austin Group Defect 1311 is applied, changing “*join* command” to “**j** command” in the  
94018 RATIONALE section.
- 94019 Austin Group Defect 1582 is applied, clarifying the behavior when an address is omitted  
94020 between two address separators.
- 94021 Austin Group Defect 1607 is applied, clarifying the behavior when more than the maximum  
94022 number of accepted addresses are provided to a command.
- 94023 Austin Group Defect 1662 is applied, clarifying requirements relating to delimiters in addresses



94024 and in `s` commands.

94025 Austin Group Defect 1786 is applied, clarifying the behavior when an `e` command names a file  
94026 that does not exist, and clarifying how the `ed` utility keeps track of whether the buffer has been  
94027 modified.

94028 **NAME**94029 `env` — set the environment for command invocation94030 **SYNOPSIS**94031 `env [-i] [name=value]... [utility [argument...]]`94032 **DESCRIPTION**94033 The `env` utility shall obtain the current environment, modify it according to its arguments, then  
94034 invoke the utility named by the *utility* operand with the modified environment.94035 Optional arguments shall be passed to *utility*.94036 If no *utility* operand is specified, the resulting environment shall be written to the standard  
94037 output, with one *name=value* pair per line.

94038 If the first argument is '-', the results are unspecified.

94039 **OPTIONS**94040 The `env` utility shall conform to XBD [Section 12.2](#) (on page 215), except for the unspecified usage  
94041 of '-'.  
94042 The following options shall be supported:

94042 The following options shall be supported:

94043 **-i** Invoke *utility* with exactly the environment specified by the arguments; the  
94044 inherited environment shall be ignored completely.94045 **OPERANDS**

94046 The following operands shall be supported:

94047 *name=value* Arguments of the form *name=value* shall modify the execution environment, and  
94048 shall be placed into the inherited environment before the *utility* is invoked.94049 *utility* The name of the utility to be invoked. If the *utility* operand names any of the  
94050 special built-in utilities in [Section 2.15](#) (on page 2526), the results are undefined.94051 *argument* A string to pass as an argument for the invoked utility.94052 **STDIN**

94053 Not used.

94054 **INPUT FILES**

94055 None.

94056 **ENVIRONMENT VARIABLES**94057 The following environment variables shall affect the execution of `env`:94058 **LANG** Provide a default value for the internationalization variables that are unset or null.  
94059 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
94060 variables used to determine the values of locale categories.)94061 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
94062 internationalization variables.94063 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
94064 characters (for example, single-byte as opposed to multi-byte characters in  
94065 arguments).94066 **LC\_MESSAGES**94067 Determine the locale that should be used to affect the format and contents of  
94068 diagnostic messages written to standard error.

94069	XSI	<b>NLSPATH</b>	Determine the location of messages objects and message catalogs.
94070		<b>PATH</b>	Determine the location of the <i>utility</i> , as described in XBD <a href="#">Chapter 8</a> (on page 167).
94071			If <i>PATH</i> is specified as a <i>name=value</i> operand to <i>env</i> , the <i>value</i> given shall be used in
94072			the search for <i>utility</i> .
94073		<b>ASYNCHRONOUS EVENTS</b>	
94074			Default.
94075		<b>STDOUT</b>	
94076			If no <i>utility</i> operand is specified, each <i>name=value</i> pair in the resulting environment shall be
94077			written in the form:
94078			"%s=%s\n", <name>, <value>
94079			If the <i>utility</i> operand is specified, the <i>env</i> utility shall not write to standard output.
94080		<b>STDERR</b>	
94081			The standard error shall be used only for diagnostic messages.
94082		<b>OUTPUT FILES</b>	
94083			None.
94084		<b>EXTENDED DESCRIPTION</b>	
94085			None.
94086		<b>EXIT STATUS</b>	
94087			If <i>utility</i> is invoked, the exit status of <i>env</i> shall be the exit status of <i>utility</i> ; otherwise, the <i>env</i>
94088			utility shall exit with one of the following values:
94089		0	The <i>env</i> utility completed successfully.
94090		1–125	An error occurred in the <i>env</i> utility.
94091		126	The utility specified by <i>utility</i> was found but could not be invoked.
94092		127	The utility specified by <i>utility</i> could not be found.
94093		<b>CONSEQUENCES OF ERRORS</b>	
94094			Default.
94095		<b>APPLICATION USAGE</b>	
94096			The <i>command</i> , <i>env</i> , <i>nice</i> , <i>nohup</i> , <i>time</i> , <i>timeout</i> , and <i>xargs</i> utilities have been specified to use exit
94097			code 127 if a utility to be invoked cannot be found, so that applications can distinguish “failure
94098			to find a utility” from “invoked utility exited with an error indication”. The value 127 was
94099			chosen because it is not commonly used for other meanings; most utilities use small values for
94100			“normal error conditions” and the values above 128 can be confused with termination due to
94101			receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could
94102			be found, but not invoked. Some scripts produce meaningful error messages differentiating the
94103			126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice
94104			that uses 127 when all attempts to <i>exec</i> the utility fail with [ENOENT], and uses 126 when any
94105			attempt to <i>exec</i> the utility fails for any other reason.
94106			Historical implementations of the <i>env</i> utility use the <i>execvp()</i> or <i>execlp()</i> functions defined in the
94107			System Interfaces volume of POSIX.1-2024 to invoke the specified utility; this provides better
94108			performance and keeps users from having to escape characters with special meaning to the shell.
94109			Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are
94110			not found by this type of <i>env</i> implementation. However, <i>env</i> can be implemented as a shell built-
94111			in, in which case it may be able to execute shell functions and built-ins. An application wishing
94112			to ensure execution of a non-built-in utility can use <i>exec</i> in a subshell for this purpose.

94113 **EXAMPLES**

94114 The following command:

94115 `env -i PATH=/mybin:"$PATH" $(getconf V7_ENV) mygrep xyz myfile`94116 invokes the command *mygrep* with a new *PATH* value as the only entry in its environment other  
94117 than any variables required by the implementation for conformance. In this case, *PATH* is used  
94118 to locate *mygrep*, which is expected to reside in */mybin*.94119 **RATIONALE**94120 As with all other utilities that invoke other utilities, this volume of POSIX.1-2024 only specifies  
94121 what *env* does with standard input, standard output, standard error, input files, and output files.  
94122 If a utility is executed, it is not constrained by the specification of input and output by *env*.94123 The *-i* option was added to allow the functionality of the removed *-* option in a manner  
94124 compatible with the Utility Syntax Guidelines. It is possible to create a non-conforming  
94125 environment using the *-i* option, as it may remove environment variables required by the  
94126 implementation for conformance. The following will preserve these environment variables as  
94127 well as preserve the *PATH* for conforming utilities:

```

94128 IFS='
94129 '
94130 # The preceding value should be <space><tab><newline>.
94131 # Set IFS to its default value.

94132 set -f
94133 # disable pathname expansion

94134 \unalias -a
94135 # Unset all possible aliases.
94136 # Note that unalias is escaped to prevent an alias
94137 # being used for unalias.
94138 # This step is not strictly necessary, since aliases are not inherited,
94139 # and the ENV environment variable is only used by interactive shells,
94140 # the only way any aliases can exist in a script is if it defines them
94141 # itself.

94142 unset -f env getconf
94143 # Ensure env and getconf are not user functions.

94144 env -i $(getconf V7_ENV) PATH="$(getconf PATH)" command

```

94145 Some have suggested that *env* is redundant since the same effect is achieved by:94146 `name=value ... utility [ argument ... ]`94147 The example is equivalent to *env* when an environment variable is being added to the  
94148 environment of the command, but not when the environment is being set to the given value.  
94149 The *env* utility also writes out the current environment if invoked without arguments. There is  
94150 sufficient functionality beyond what the example provides to justify inclusion of *env*.94151 **FUTURE DIRECTIONS**

94152 None.

94153 **SEE ALSO**94154 [Section 2.15](#) (on page 2526), [Section 2.5](#) (on page 2478)94155 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

94156 **CHANGE HISTORY**

94157 First released in Issue 2.

94158 **Issue 7**94159 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first  
94160 argument is '- '.94161 Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use  
94162 the *env* utility to preserve a conforming environment.

94163 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94164 The EXAMPLES section is revised to change the use of *env -i*.94165 **Issue 8**94166 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.94167 Austin Group Defect 1157 is applied, adding a note about shell built-in implementations of *env*  
94168 to the APPLICATION USAGE section.94169 Austin Group Defect 1586 is applied, adding the *timeout* utility.

94170 Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

94171 **NAME**

94172 ex — text editor

94173 **SYNOPSIS**94174 UP `ex [-rR] [-s|-v] [-c command] [-t tagstring] [-w size] [file...]`94175 **DESCRIPTION**

94176 The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and  
 94177 visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open**  
 94178 and **visual** commands and in *vi*.

94179 If an operand is '-', the results are unspecified.

94180 This section uses the term *edit buffer* to describe the current working text. No specific  
 94181 implementation is implied by this term. All editing changes are performed on the edit buffer,  
 94182 and no changes to it shall affect any file until an editor command writes the file.

94183 Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,  
 94184 such as the full-screen editing commands (*visual mode* or *open mode*). When these commands  
 94185 cannot be supported on such terminals, this condition shall not produce an error message such  
 94186 as “not an editor command” or report a syntax error. The implementation may either accept the  
 94187 commands and produce results on the screen that are the result of an unsuccessful attempt to  
 94188 meet the requirements of this volume of POSIX.1-2024 or report an error describing the terminal-  
 94189 related deficiency.

94190 **OPTIONS**

94191 The *ex* utility shall conform to XBD [Section 12.2](#) (on page 215), except for the unspecified usage  
 94192 of '-', and that '+' may be recognized as an option delimiter as well as '-'.

94193 The following options shall be supported:

94194 **-c *command*** Specify an initial command to be executed in the first edit buffer loaded from an  
 94195 existing file (see the EXTENDED DESCRIPTION section). Implementations may  
 94196 support more than a single -c option. In such implementations, the specified  
 94197 commands shall be executed in the order specified on the command line.

94198 **-r** Recover the named files (see the EXTENDED DESCRIPTION section). Recovery  
 94199 information for a file shall be saved during an editor or system crash (for example,  
 94200 when the editor is terminated by a signal which the editor can catch), or after the  
 94201 use of an *ex* **preserve** command.

94202 A *crash* in this context is an unexpected failure of the system or utility that requires  
 94203 restarting the failed system or utility. A system crash implies that any utilities  
 94204 running at the time also crash. In the case of an editor or system crash, the number  
 94205 of changes to the edit buffer (since the most recent **preserve** command) that will be  
 94206 recovered is unspecified.

94207 If no *file* operands are given and the -t option is not specified, all other options, the  
 94208 *EXINIT* variable, and any **.exrc** files shall be ignored; a list of all recoverable files  
 94209 available to the invoking user shall be written, and the editor shall exit normally  
 94210 without further action.

94211 **-R** Set **readonly** edit option.

94212 **-s** Prepare *ex* for batch use by taking the following actions:

- 94213 • Suppress writing prompts and informational (but not diagnostic) messages.
- 94214 • Ignore the value of *TERM* and any implementation default terminal type and
- 94215 assume the terminal is a type incapable of supporting open or visual modes;
- 94216 see the **visual** command and the description of *vi*.
- 94217 • Suppress the use of the *EXINIT* environment variable and the reading of any
- 94218 **.exrc** file; see the EXTENDED DESCRIPTION section.
- 94219 • Suppress autoindentation, ignoring the value of the **autoindent** edit option.
- 94220 **-t tagstring** Edit the file containing the specified *tagstring*; see *ctags*. The tags feature
- 94221 represented by **-t tagstring** and the **tag** command is optional. It shall be provided
- 94222 on any system that also provides a conforming implementation of *ctags*; otherwise,
- 94223 the use of **-t** produces undefined results. On any system, it shall be an error to
- 94224 specify more than a single **-t** option.
- 94225 **-v** Begin in visual mode (see *vi*).
- 94226 **-w size** Set the value of the *window* editor option to *size*.

#### 94227 OPERANDS

94228 The following operand shall be supported:

94229 *file* A pathname of a file to be edited.

#### 94230 STDIN

94231 The standard input consists of a series of commands and input text, as described in the

94232 EXTENDED DESCRIPTION section. The implementation may limit each line of standard input

94233 to a length of {LINE\_MAX}.

94234 If the standard input is not a terminal device, it shall be as if the **-s** option had been specified.

94235 If a read from the standard input returns an error, or if the editor detects an end-of-file condition

94236 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

#### 94237 INPUT FILES

94238 Input files shall be text files or files that would be text files except for an incomplete last line that

94239 is not longer than {LINE\_MAX}-1 bytes in length and contains no NUL characters. By default,

94240 any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other

94241 forms of files may optionally be allowed by *ex* implementations.

94242 The **.exrc** files and source files shall be text files consisting of *ex* commands; see the EXTENDED

94243 DESCRIPTION section.

94244 By default, the editor shall read lines from the files to be edited without interpreting any of those

94245 lines as any form of editor command.

#### 94246 ENVIRONMENT VARIABLES

94247 The following environment variables shall affect the execution of *ex*:

94248 *COLUMNS* Override the system-selected horizontal screen size. See XBD Chapter 8 (on page

94249 167) for valid values and results when it is unset or null.

94250 *EXINIT* Determine a list of *ex* commands that are executed on editor start-up. See the

94251 EXTENDED DESCRIPTION section for more details of the initialization phase.

94252 *HOME* Determine a pathname of a directory that shall be searched for an editor start-up

94253 file named **.exrc**; see the EXTENDED DESCRIPTION section.

94254	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD <a href="#">Section 8.2</a> (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)
94255		
94256		
94257	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
94258		
94259	<i>LC_COLLATE</i>	
94260		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.
94261		
94262	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as uppercase or lowercase letters, the case conversion of letters, and the detection of word boundaries.
94263		
94264		
94265		
94266		
94267	<i>LC_MESSAGES</i>	
94268		Determine the locale used to process affirmative responses, and the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
94269		
94270		
94271	<i>LINES</i>	Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical screen size in visual mode. See XBD <a href="#">Chapter 8</a> (on page 167) for valid values and results when it is unset or null.
94272		
94273		
94274	XSI <i>NLSPATH</i>	Determine the location of messages objects and message catalogs.
94275	<i>PATH</i>	Determine the search path for the shell command specified in the <i>ex</i> editor commands <b>!</b> , <b>shell</b> , <b>read</b> , and <b>write</b> , and the open and visual mode command <b>!</b> ; see the description of command search and execution in <a href="#">Section 2.9.1.4</a> (on page 2502).
94276		
94277		
94278	<i>SHELL</i>	Determine the preferred command line interpreter for use as the default value of the <b>shell</b> edit option.
94279		
94280	<i>TERM</i>	Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.
94281		
94282	<b>ASYNCHRONOUS EVENTS</b>	
94283		The following term is used in this and following sections to specify command and asynchronous event actions:
94284		
94285	<i>complete write</i>	
94286		A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or the saving of the edit buffer caused by the user executing the <i>ex</i> <b>preserve</b> command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write.
94287		
94288		
94289		
94290		
94291		The following actions shall be taken upon receipt of signals:
94292	<i>SIGINT</i>	If the standard input is not a terminal device, <i>ex</i> shall not write the file or return to command or text input mode, and shall exit with a non-zero exit status.
94293		
94294		Otherwise, if executing an open or visual text input mode command, <i>ex</i> in receipt of <i>SIGINT</i> shall behave identically to its receipt of the <ESC> character.
94295		
94296		Otherwise:



- 94297 1. If executing an *ex* text input mode command, all input lines that have been  
 94298 completely entered shall be resolved into the edit buffer, and any partially  
 94299 entered line shall be discarded.
- 94300 2. If there is a currently executing command, it shall be aborted and a message  
 94301 displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,  
 94302 it is unspecified whether any lines modified by the executing command  
 94303 appear modified, or as they were before being modified by the executing  
 94304 command, in the buffer.
- 94305 If the currently executing command was a motion command, its associated  
 94306 command shall be discarded.
- 94307 3. If in open or visual command mode, the terminal shall be alerted.
- 94308 4. The editor shall then return to command mode.
- 94309 SIGCONT If *ex* is in open mode or visual mode, the actions described below for SIGWINCH  
 94310 shall be taken, except that the screen shall always be refreshed (regardless of  
 94311 whether the terminal window size changed).
- 94312 SIGHUP If the edit buffer has been modified since the last complete write, *ex* shall attempt  
 94313 to save the edit buffer so that it can be recovered later using the `-r` option or the *ex*  
 94314 **recover** command. The editor shall not write the file or return to command or text  
 94315 input mode, and shall terminate with a non-zero exit status.
- 94316 SIGTERM Refer to SIGHUP.
- 94317 SIGWINCH If *ex* is in open mode or visual mode, the current terminal window size associated  
 94318 with the terminal on standard output shall be obtained, as if by a call to XSH  
 94319 `tcgetwinsize()`. If the terminal window size is successfully obtained, it shall be used  
 94320 as follows:
- 94321 • If the *COLUMNS* environment variable is unset or does not contain a  
 94322 number, the horizontal screen size shall be set to the number of columns in  
 94323 the obtained terminal window size.
  - 94324 • If *ex* is in visual mode, the `-w` option was not specified and the *LINES*  
 94325 environment variable is unset or does not contain a number, the vertical  
 94326 screen size shall be set to the number of rows in the obtained terminal  
 94327 window size.
- 94328 If the above resulted in either the vertical screen size or the horizontal screen size  
 94329 (or both) changing to a different value, *ex* shall update the values it has for the  
 94330 number of lines and columns in the display and shall adjust the **window** edit  
 94331 option and the column number at which the **wrapmargin** edit option takes effect  
 94332 (if non-zero) accordingly (see [Edit Options in ex](#), on page 2877) and refresh the  
 94333 screen; otherwise, *ex* may refresh the screen.
- 94334 The action taken for all other signals is unspecified.
- 94335 **STDOUT**
- 94336 The standard output shall be used only for writing prompts to the user, for informational  
 94337 messages, and for writing lines from the file.

94338 **STDERR**

94339 The standard error shall be used only for diagnostic messages.

94340 **OUTPUT FILES**94341 The output from *ex* shall be text files.94342 **EXTENDED DESCRIPTION**94343 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing  
94344 capabilities available in *ex*.94345 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such  
94346 as inverse video), the message shall be written in standout mode. If the terminal does not  
94347 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the  
94348 error message.94349 By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the  
94350 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;  
94351 it can be exited (and command mode re-entered) by typing a <period> ('.') alone at the  
94352 beginning of a line.94353 **Initialization in *ex* and *vi***94354 The following symbols are used in this and following sections to specify locations in the edit  
94355 buffer:94356 *alternate and current pathnames*94357 Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex*  
94358 commands that take filenames as arguments shall set them as follows:

- 94359 1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag**  
94360 command replaces the contents of the edit buffer.
  - 94361 a. If the command replaces the contents of the edit buffer, the current pathname  
94362 shall be set to the *file* argument or the file indicated by the tag, and the  
94363 alternate pathname shall be set to the previous value of the current pathname.
  - 94364 b. Otherwise, the alternate pathname shall be set to the *file* argument.
- 94365 2. If a *file* argument is specified to the *ex* **next** command:
  - 94366 a. If the command replaces the contents of the edit buffer, the current pathname  
94367 shall be set to the first *file* argument, and the alternate pathname shall be set to  
94368 the previous value of the current pathname.
- 94369 3. If a *file* argument is specified to the *ex* **file** command, the current pathname shall be  
94370 set to the *file* argument, and the alternate pathname shall be set to the previous value  
94371 of the current pathname.
- 94372 4. If a *file* argument is specified to the *ex* **read** and **write** commands (that is, when  
94373 reading or writing a file, and not to the program named by the **shell** edit option), or a  
94374 *file* argument is specified to the *ex* **xit** command:
  - 94375 a. If the current pathname has no value, the current pathname shall be set to the  
94376 *file* argument.
  - 94377 b. Otherwise, the alternate pathname shall be set to the *file* argument.

94378 If the alternate pathname is set to the previous value of the current pathname when the  
94379 current pathname had no previous value, then the alternate pathname shall have no value  
94380 as a result.

94381 *current line*  
 94382 The line of the edit buffer referenced by the cursor. Each command description specifies the  
 94383 current line after the command has been executed, as the *current line value*. When the edit  
 94384 buffer contains no lines, the current line shall be zero; see [Addressing in ex](#) (on page 2845).

94385 *current column*  
 94386 The current display line column occupied by the cursor. (The columns shall be numbered  
 94387 beginning at 1.) Each command description specifies the current column after the command  
 94388 has been executed, as the *current column value*. This column is an *ideal* column that is  
 94389 remembered over the lifetime of the editor. The actual display line column upon which the  
 94390 cursor rests may be different from the current column; see the cursor positioning discussion  
 94391 in [Command Descriptions in vi](#) (on page 3527).

94392 *set to non-<blank>*  
 94393 A description for a current column value, meaning that the current column shall be set to  
 94394 the last display line column on which is displayed any part of the first non-<blank> of the  
 94395 line. If the line has no non-<blank> non-<newline> characters, the current column shall be  
 94396 set to the last display line column on which is displayed any part of the last non-<newline>  
 94397 character in the line. If the line is empty, the current column shall be set to column position  
 94398 1.

94399 The length of lines in the edit buffer may be limited to {LINE\_MAX} bytes. In open and visual  
 94400 mode, the length of lines in the edit buffer may be limited to the number of characters that will  
 94401 fit in the display. If either limit is exceeded during editing, an error message shall be written. If  
 94402 either limit is exceeded by a line read in from a file, an error message shall be written and the  
 94403 edit session may be terminated.

94404 If the editor stops running due to any reason other than a user command, and the edit buffer has  
 94405 been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous  
 94406 event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

94407 During initialization (before the first file is copied into the edit buffer or any user commands  
 94408 from the terminal are processed) the following shall occur:

- 94409 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands  
 94410 contained in that variable.
- 94411 2. If the *EXINIT* variable is not set, and all of the following are true:
  - 94412 a. The *HOME* environment variable is not null and not empty.
  - 94413 b. The file *.exrc* in the directory referred to by the *HOME* environment variable:
    - 94414 i. Exists
    - 94415 ii. Is owned by the same user ID as the real user ID of the process or the  
 94416 process has appropriate privileges
    - 94417 iii. Is not writable by anyone other than the owner

94418 the editor shall execute the *ex* commands contained in that file.

- 94419 3. If and only if all of the following are true:
  - 94420 a. The current directory is not referred to by the *HOME* environment variable.
  - 94421 b. A command in the *EXINIT* environment variable or a command in the *.exrc* file in  
 94422 the directory referred to by the *HOME* environment variable sets the editor option  
 94423 **exrc**.

- 94424 c. The **.exrc** file in the current directory:
- 94425 i. Exists
- 94426 ii. Is owned by the same user ID as the real user ID of the process, or by one of
- 94427 a set of implementation-defined user IDs
- 94428 iii. Is not writable by anyone other than the owner

94429 the editor shall attempt to execute the *ex* commands contained in that file.

94430 Lines in any **.exrc** file that are blank lines shall be ignored. If any **.exrc** file exists, but is not read

94431 for ownership or permission reasons, it shall be an error.

94432 After the *EXINIT* variable and any **.exrc** files are processed, the first file specified by the user

94433 shall be edited, as follows:

- 94434 1. If the user specified the **-t** option, the effect shall be as if the *ex tag* command was entered
- 94435 with the specified argument, with the exception that if tag processing does not result in a
- 94436 file to edit, the effect shall be as described in step 3. below.
- 94437 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if
- 94438 the *ex edit* command was entered with the first of those arguments as its *file* argument.
- 94439 3. Otherwise, the effect shall be as if the *ex edit* command was entered with a nonexistent
- 94440 filename as its *file* argument. It is unspecified whether this action shall set the current
- 94441 pathname. In an implementation where this action does not set the current pathname, any
- 94442 editor command using the current pathname shall fail until an editor command sets the
- 94443 current pathname.

94444 If the **-r** option was specified, the first time a file in the initial argument list or a file specified by

94445 the **-t** option is edited, if recovery information has previously been saved about it, that

94446 information shall be recovered and the editor shall behave as if the contents of the edit buffer

94447 have already been modified. If there are multiple instances of the file to be recovered, the one

94448 most recently saved shall be recovered, and an informational message that there are previous

94449 versions of the file that can be recovered shall be written. If no recovery information about a file

94450 is available, an informational message to this effect shall be written, and the edit shall proceed as

94451 usual.

94452 If the **-c** option was specified, the first time a file that already exists (including a file that might

94453 not exist but for which recovery information is available, when the **-r** option is specified)

94454 replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of

94455 the edit buffer, the current column shall be set to non-<blank>, and the *ex* commands specified

94456 with the **-c** option shall be executed. In this case, the current line and current column shall not

94457 be set as described for the command associated with the replacement or initialization of the edit

94458 buffer contents. However, if the **-t** option or a **tag** command is associated with this action, the **-c**

94459 option commands shall be executed and then the movement to the tag shall be performed.

94460 The current argument list shall initially be set to the filenames specified by the user on the

94461 command line. If no filenames are specified by the user, the current argument list shall be empty.

94462 If the **-t** option was specified, it is unspecified whether any filename resulting from tag

94463 processing shall be prepended to the current argument list. In the case where the filename is

94464 added as a prefix to the current argument list, the current argument list reference shall be set to

94465 that filename. In the case where the filename is not added as a prefix to the current argument

94466 list, the current argument list reference shall logically be located before the first of the filenames

94467 specified on the command line (for example, a subsequent *ex next* command shall edit the first

94468 filename from the command line). If the **-t** option was not specified, the current argument list

94469 reference shall be to the first of the filenames on the command line.

94470 **Addressing in ex**

94471 Addressing in *ex* relates to the current line and the current column; the address of a line is its  
 94472 1-based line number, the address of a column is its 1-based count from the beginning of the line.  
 94473 Generally, the current line is the last line affected by a command. The current line number is the  
 94474 address of the current line. In each command description, the effect of the command on the  
 94475 current line number and the current column is described.

94476 Addresses are constructed as follows:

- 94477 1. The character '.' (period) shall address the current line.
- 94478 2. The character '\$' shall address the last line of the edit buffer.
- 94479 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 94480 4. The address "'x" refers to the line marked with the mark name character 'x', which  
 94481 shall be a lowercase letter from the portable character set, the backquote character, or the  
 94482 single-quote character. It shall be an error if the line that was marked is not currently  
 94483 present in the edit buffer or the mark has not been set. Lines can be marked with the *ex*  
 94484 **mark** or **k** commands, or the *vi* **m** command.
- 94485 5. A regular expression enclosed by <slash> characters ('/') shall address the first line  
 94486 found by searching forwards from the line following the current line toward the end of  
 94487 the edit buffer and stopping at the first line for which the line excluding the terminating  
 94488 <newline> matches the regular expression. As stated in [Regular Expressions in ex](#) (on  
 94489 page 2875), an address consisting of a null regular expression delimited by <slash>  
 94490 characters ("/") shall address the next line for which the line excluding the terminating  
 94491 <newline> matches the last regular expression encountered. In addition, the second  
 94492 <slash> can be omitted at the end of a command line. If the **wrapsan** edit option is set,  
 94493 the search shall wrap around to the beginning of the edit buffer and continue up to and  
 94494 including the current line, so that the entire edit buffer is searched. Within the regular  
 94495 expression, the sequence "\/" shall represent a literal <slash> instead of the regular  
 94496 expression delimiter.
- 94497 6. A regular expression enclosed in <question-mark> characters ('?') shall address the first  
 94498 line found by searching backwards from the line preceding the current line toward the  
 94499 beginning of the edit buffer and stopping at the first line for which the line excluding the  
 94500 terminating <newline> matches the regular expression. An address consisting of a null  
 94501 regular expression delimited by <question-mark> characters ("??") shall address the  
 94502 previous line for which the line excluding the terminating <newline> matches the last  
 94503 regular expression encountered. In addition, the second <question-mark> can be omitted  
 94504 at the end of a command line. If the **wrapsan** edit option is set, the search shall wrap  
 94505 around from the beginning of the edit buffer to the end of the edit buffer and continue up  
 94506 to and including the current line, so that the entire edit buffer is searched. Within the  
 94507 regular expression, the sequence "\?" shall represent a literal <question-mark> instead  
 94508 of the RE delimiter.
- 94509 7. A <plus-sign> ('+') or a <hyphen-minus> ('-') followed by a decimal number shall  
 94510 address the current line plus or minus the number. A '+' or '-' not followed by a  
 94511 decimal number shall address the current line plus or minus 1.

94512 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.  
 94513 Address offsets are constructed as follows:

94514 1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the  
 94515 indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal  
 94516 number shall add (subtract) 1 to (from) the address.

94517 2. A decimal number shall add the indicated number of lines to the address.

94518 It shall not be an error for an intermediate address value to be less than zero or greater than the  
 94519 last line in the edit buffer. It shall be an error for the final address value to be less than zero or  
 94520 greater than the last line in the edit buffer.

94521 Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in  
 94522 [Command Descriptions in ex](#) (on page 2852). If more than the required number of addresses are  
 94523 provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than  
 94524 the required number of addresses are provided to a command, the addresses specified first shall  
 94525 be evaluated and then discarded until the maximum number of valid addresses remain.

94526 Addresses shall be separated from each other by a <comma> (', ') or a <semicolon> ('; '). If  
 94527 no address is specified before or after a <comma> or <semicolon> separator, it shall be as if the  
 94528 address of the current line was specified before or after the separator. In the case of a  
 94529 <semicolon> separator, the current line ('. ') shall be set to the first address, and only then shall  
 94530 the next address be calculated. This feature can be used to determine the starting line for  
 94531 forwards and backwards searches (see rules 5. and 6.).

94532 A <percent-sign> ('%') shall be equivalent to entering the two addresses "1, \$".

94533 Any delimiting <blank> characters between addresses, address separators, or address offsets  
 94534 shall be discarded.

### 94535 **Command Line Parsing in ex**

94536 The following symbol is used in this and following sections to describe parsing behavior:

94537 *escape* If a character is referred to as ``<backslash>-escaped'' or ``<control>-V-escaped'', it  
 94538 shall mean that the character acquired or lost a special meaning by virtue of being  
 94539 preceded, respectively, by a <backslash> or <control>-V character. Unless  
 94540 otherwise specified, the escaping character shall be discarded at that time and shall  
 94541 not be further considered for any purpose.

94542 Command-line parsing shall be done in the following steps. For each step, characters already  
 94543 evaluated shall be ignored; that is, the phrase ``leading character'' refers to the next character  
 94544 that has not yet been evaluated.

94545 1. Leading <colon> characters shall be skipped.

94546 2. Leading <blank> characters shall be skipped.

94547 3. If the leading character is a double-quote character, the characters up to and including the  
 94548 next non-<backslash>-escaped <newline> shall be discarded, and any subsequent  
 94549 characters shall be parsed as a separate command.

94550 4. Leading characters that can be interpreted as addresses shall be evaluated; see  
 94551 [Addressing in ex](#) (on page 2845).

94552 5. Leading <blank> characters shall be skipped.

94553 6. If the next character is a <vertical-line> character or a <newline>:

94554 a. If the next character is a <newline>:

- 94555 i. If *ex* is in open or visual mode, the current line shall be set to the last  
94556 address specified, if any.
- 94557 ii. Otherwise, if the last command was terminated by a <vertical-line>  
94558 character, no action shall be taken; for example, the command  
94559 " | | <newline> " shall execute two implied commands, not three.
- 94560 iii. Otherwise, step 6.b. shall apply.
- 94561 b. Otherwise, the implied command shall be the **print** command. The last #, **p**, and **l**  
94562 flags specified to any *ex* command shall be remembered and shall apply to this  
94563 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the  
94564 remembered flags to #, nothing, and **l**, respectively, plus any other flags specified  
94565 for that execution of the **number**, **print**, or **list** command.
- 94566 If *ex* is not currently performing a **global** or **v** command, and no address or count  
94567 is specified, the current line shall be incremented by 1 before the command is  
94568 executed. If incrementing the current line would result in an address past the last  
94569 line in the edit buffer, the command shall fail, and the increment shall not happen.
- 94570 c. The <newline> or <vertical-line> character shall be discarded and any subsequent  
94571 characters shall be parsed as a separate command.
- 94572 7. The command name shall be comprised of the next character (if the character is not  
94573 alphabetic), or the next character and any subsequent alphabetic characters (if the  
94574 character is alphabetic), with the following exceptions:
- 94575 a. Commands that consist of any prefix of the characters in the command name  
94576 **delete**, followed immediately by any of the characters 'l', 'p', '+', '-', or '#'  
94577 shall be interpreted as a **delete** command, followed by a <blank>, followed by the  
94578 characters that were not part of the prefix of the **delete** command. The maximum  
94579 number of characters shall be matched to the command name **delete**; for example,  
94580 "de1" shall not be treated as "de" followed by the flag **l**.
- 94581 b. Commands that consist of the character 'k', followed by a character that can be  
94582 used as the name of a mark, shall be equivalent to the mark command followed by  
94583 a <blank>, followed by the character that followed the 'k'.
- 94584 c. Commands that consist of the character 's', followed by characters that could be  
94585 interpreted as valid options to the **s** command, shall be the equivalent of the **s**  
94586 command, without any pattern or replacement values, followed by a <blank>,  
94587 followed by the characters after the 's'.
- 94588 8. The command name shall be matched against the possible command names, and a  
94589 command name that contains a prefix matching the characters specified by the user shall  
94590 be the executed command. In the case of commands where the characters specified by the  
94591 user could be ambiguous, the executed command shall be as follows:

a	append	n	next	t	t
c	change	p	print	u	undo
ch	change	pr	print	un	undo
e	edit	r	read	v	v
m	move	re	read	w	write
ma	mark	s	s		

94598 Implementation extensions with names causing similar ambiguities shall not be checked  
94599 for a match until all possible matches for commands specified by POSIX.1-2024 have been

- 94600 checked.
- 94601 9. If the command is a **!** command, or if the command is a **read** command followed by zero
- 94602 or more <blank> characters and a **!**, or if the command is a **write** command followed by
- 94603 one or more <blank> characters and a **!**, the rest of the command shall include all
- 94604 characters up to a non-<backslash>-escaped <newline>. The <newline> shall be
- 94605 discarded and any subsequent characters shall be parsed as a separate *ex* command.
- 94606 10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while
- 94607 in open or visual mode, the next part of the command shall be parsed as follows:
- 94608 a. Any **'!'** character immediately following the command shall be skipped and be
- 94609 part of the command.
- 94610 b. Any leading <blank> characters shall be skipped and be part of the command.
- 94611 c. If the next character is a **'+'**, characters up to the first non-<backslash>-escaped
- 94612 <newline> or non-<backslash>-escaped <blank> shall be skipped and be part of
- 94613 the command.
- 94614 d. The rest of the command shall be determined by the steps specified in paragraph
- 94615 12.
- 94616 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the
- 94617 command shall be parsed as follows:
- 94618 a. Any leading <blank> characters shall be skipped and be part of the command.
- 94619 b. If the next character is not an alphanumeric, double-quote, <newline>,
- 94620 <backslash>, or <vertical-line> character:
- 94621 i. The next character shall be used as a command delimiter.
- 94622 ii. If the command is a **global**, **open**, or **v** command, characters up to the first
- 94623 non-<backslash>-escaped <newline>, or first non-<backslash>-escaped
- 94624 delimiter character, shall be skipped and be part of the command.
- 94625 iii. If the command is an **s** command, characters up to the first
- 94626 non-<backslash>-escaped <newline>, or second non-<backslash>-escaped
- 94627 delimiter character, shall be skipped and be part of the command.
- 94628 c. If the command is a **global** or **v** command, characters up to the first
- 94629 non-<backslash>-escaped <newline> shall be skipped and be part of the
- 94630 command.
- 94631 d. Otherwise, the rest of the command shall be determined by the steps specified in
- 94632 paragraph 12.
- 94633 12. Otherwise:
- 94634 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,
- 94635 characters up to the first non-<control>-V-escaped <newline>, <vertical-line>, or
- 94636 double-quote character shall be skipped and be part of the command.
- 94637 b. Otherwise, characters up to the first non-<backslash>-escaped <newline>,
- 94638 <vertical-line>, or double-quote character shall be skipped and be part of the
- 94639 command.
- 94640 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.
- 94641 ended at a <vertical-line> character, any subsequent characters, up to the next
- 94642 non-<backslash>-escaped <newline> shall be used as input text to the command.



- 94643 d. If the command was ended by a double-quote character, all subsequent characters,  
94644 up to the next non-`<backslash>`-escaped `<newline>`, shall be discarded.
- 94645 e. The terminating `<newline>` or `<vertical-line>` character shall be discarded and any  
94646 subsequent characters shall be parsed as a separate *ex* command.

94647 Command arguments shall be parsed as described by the Synopsis and Description of each  
94648 individual *ex* command. This parsing shall not be `<blank>`-sensitive, except for the `!` argument,  
94649 which has to follow the command name without intervening `<blank>` characters, and where it  
94650 would otherwise be ambiguous. For example, *count* and *flag* arguments need not be  
94651 `<blank>`-separated because `"d22p"` is not ambiguous, but *file* arguments to the *ex next*  
94652 command need to be separated by one or more `<blank>` characters. Any `<blank>` in command  
94653 arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands can be  
94654 `<control>`-V-escaped, in which case the `<blank>` shall not be used as an argument delimiter. Any  
94655 `<blank>` in the command argument for any other command can be `<backslash>`-escaped, in  
94656 which case that `<blank>` shall not be used as an argument delimiter.

94657 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,  
94658 any character can be `<control>`-V-escaped. All such escaped characters shall be treated literally  
94659 and shall have no special meaning. Within command arguments for all other *ex* commands that  
94660 are not regular expressions or replacement strings, any character that would otherwise have a  
94661 special meaning can be `<backslash>`-escaped. Escaped characters shall be treated literally,  
94662 without special meaning as shell expansion characters or `'!'`, `'%'`, and `'#'` expansion  
94663 characters. See [Regular Expressions in ex](#) (on page 2875) and [Replacement Strings in ex](#) (on page  
94664 2876) for descriptions of command arguments that are regular expressions or replacement  
94665 strings.

94666 Non-`<backslash>`-escaped `'%'` characters appearing in *file* arguments to any *ex* command shall  
94667 be replaced by the current pathname; unescaped `'#'` characters shall be replaced by the  
94668 alternate pathname. It shall be an error if `'%'` or `'#'` characters appear unescaped in an  
94669 argument and their corresponding values are not set.

94670 Non-`<backslash>`-escaped `'!'` characters in the arguments to either the *ex !* command or the  
94671 open and visual mode `!` command, or in the arguments to the *ex read* command, where the first  
94672 non-`<blank>` after the command name is a `'!'` character, or in the arguments to the *ex write*  
94673 command where the command name is followed by one or more `<blank>` characters and the  
94674 first non-`<blank>` after the command name is a `'!'` character, shall be replaced with the  
94675 arguments to the last of those three commands as they appeared after all unescaped `'%'`, `'#'`,  
94676 and `'!'` characters were replaced. It shall be an error if `'!'` characters appear unescaped in one  
94677 of these commands and there has been no previous execution of one of these commands.

94678 If an error occurs during the parsing or execution of an *ex* command:

- 94679 • An informational message to this effect shall be written. Execution of the *ex* command shall  
94680 stop, and the cursor (for example, the current line and column) shall not be further  
94681 modified.
- 94682 • If the *ex* command resulted from a map expansion, all characters from that map expansion  
94683 shall be discarded, except as otherwise specified by the **map** command.
- 94684 • Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment  
94685 variable, a **.exrc** file, a **:source** command, a `-c` option, or a `+command` specified to an *ex edit*,  
94686 *ex next*, or *visual* command, no further commands from the source of the commands shall  
94687 be executed.

- 94688 • Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v**
- 94689 command, no further commands caused by the execution of the buffer or the **global** or **v**
- 94690 command shall be executed.
- 94691 • Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and
- 94692 including the next non-<backslash>-escaped <newline> shall be discarded.

### 94693 **Input Editing in ex**

94694 The following symbol is used in this and the following sections to specify command actions:

94695 *word* In the POSIX locale, a word consists of a maximal sequence of letters, digits, and

94696 underscores, delimited at both ends by characters other than letters, digits, or

94697 underscores, or by the beginning or end of a line or the edit buffer.

94698 When accepting input characters from the user, in either *ex* command mode or *ex* text input

94699 mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces

94700 volume of POSIX.1-2024.

94701 If in *ex* text input mode:

- 94702 1. If the **number** edit option is set, *ex* shall prompt for input using the line number that
- 94703 would be assigned to the line if it is entered, in the format specified for the *ex* **number**
- 94704 command.
- 94705 2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters,
- 94706 as described by the **autoindent** edit option. **autoindent** characters shall follow the line
- 94707 number, if any.

94708 If in *ex* command mode:

- 94709 1. If the **prompt** edit option is set, input shall be prompted for using a single ' : ' character;
- 94710 otherwise, there shall be no prompt.

94711 The input characters in the following sections shall have the following effects on the input line.

### 94712 **Scroll**

94713 *Synopsis:* eof

94714 See the description of the *stty eof* character in *stty*.

94715 If in *ex* command mode:

94716 If the *eof* character is the first character entered on the line, the line shall be evaluated as if

94717 it contained two characters: a <control>-D and a <newline>.

94718 Otherwise, the *eof* character shall have no special meaning.

94719 If in *ex* text input mode:

94720 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be  
 94721 modified so that a part of the next text input character is displayed on the first column in  
 94722 the line after the previous **shiftwidth** edit option column boundary, and the user shall be  
 94723 prompted again for input for the same line.

94724 Otherwise, if the cursor follows a '0', which follows an **autoindent** character, and the '0'  
 94725 was the previous text input character, the '0' and all **autoindent** characters in the line  
 94726 shall be discarded, and the user shall be prompted again for input for the same line.

94727 Otherwise, if the cursor follows a '^', which follows an **autoindent** character, and the '^'  
 94728 was the previous text input character, the '^' and all **autoindent** characters in the line  
 94729 shall be discarded, and the user shall be prompted again for input for the same line. In  
 94730 addition, the **autoindent** level for the next input line shall be derived from the same line  
 94731 from which the **autoindent** level for the current input line was derived.

94732 Otherwise, if there are no **autoindent** or text input characters in the line, the *eof* character  
 94733 shall be discarded.

94734 Otherwise, the *eof* character shall have no special meaning.

#### 94735 <newline>

94736 *Synopsis:* <newline>  
 94737 <control>-J

94738 If in *ex* command mode:

94739 Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for  
 94740 this purpose.

94741 If in *ex* text input mode:

94742 Terminate the current line. If there are no characters other than **autoindent** characters on  
 94743 the line, all characters on the line shall be discarded.

94744 Prompt for text input on a new line after the current line. If the **autoindent** edit option is  
 94745 set, an appropriate number of **autoindent** characters shall be added as a prefix to the line  
 94746 as described by the *ex* **autoindent** edit option.

#### 94747 <backslash>

94748 *Synopsis:* <backslash>

94749 Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any  
 94750 special meaning that it may have to the editor during text input mode. The <backslash>  
 94751 character shall be retained and evaluated when the command line is parsed, or retained and  
 94752 included when the input text becomes part of the edit buffer.

## 94753 &lt;control&gt;-V

94754 *Synopsis:* <control>-V

94755 Allow the entry of any subsequent character as a literal character, removing any special meaning  
 94756 that it may have to the editor during text input mode. The <control>-V character shall be  
 94757 discarded before the command line is parsed or the input text becomes part of the edit buffer.

94758 If the ``literal next'' functionality is performed by the underlying system, it is implementation-  
 94759 defined whether a character other than <control>-V performs this function.

## 94760 &lt;control&gt;-W

94761 *Synopsis:* <control>-W

94762 Discard the <control>-W, and the word previous to it in the input line, including any <blank>  
 94763 characters following the word and preceding the <control>-W. If the ``word erase'' functionality  
 94764 is performed by the underlying system, it is implementation-defined whether a character other  
 94765 than <control>-W performs this function.

94766 **Command Descriptions in ex**

94767 The following symbols are used in this section to represent command modifiers. Some of these  
 94768 modifiers can be omitted, in which case the specified defaults shall be used.

94769 *1addr* A single line address, given in any of the forms described in [Addressing in ex](#) (on  
 94770 page 2845); the default shall be the current line ( ' . ' ), unless otherwise specified.

94771 If the line address is zero, it shall be an error, unless otherwise specified in the  
 94772 following command descriptions.

94773 If the edit buffer is empty, and the address is specified with a command other than  
 94774 =, **append**, **insert**, **open**, **put**, **read**, or **visual**, or the address is not zero, it shall be  
 94775 an error.

94776 *2addr* Two addresses specifying an inclusive range of lines. If no addresses are specified,  
 94777 the default for *2addr* shall be the current line only ( " . , . " ), unless otherwise  
 94778 specified in the following command descriptions. If one address is specified, *2addr*  
 94779 shall specify that line only, unless otherwise specified in the following command  
 94780 descriptions.

94781 It shall be an error if the first address is greater than the second address.

94782 If the edit buffer is empty, and the two addresses are specified with a command  
 94783 other than the **!**, **write**, **wq**, or **xit** commands, or either address is not zero, it shall  
 94784 be an error.

94785 *count* A positive decimal number. If *count* is specified, it shall be equivalent to specifying  
 94786 an additional address to the command, unless otherwise specified by the following  
 94787 command descriptions. The additional address shall be equal to the last address  
 94788 specified to the command (either explicitly or by default) plus *count*-1.

94789 If this would result in an address greater than the last line of the edit buffer, it shall  
 94790 be corrected to equal the last line of the edit buffer.

94791 *flags* One or more of the characters '+', '-', '#', 'p', or 'l' (ell). The flag characters  
 94792 can be <blank>-separated, and in any order or combination. The characters '#',  
 94793 'p', and 'l' shall cause lines to be written in the format specified by the **print**  
 94794 command with the specified *flags*.

94795 The lines to be written are as follows:

- 94796 1. All edit buffer lines written during the execution of the *ex* **&**, **~**, **list**, **number**,  
94797 **open**, **print**, **s**, **visual**, and **z** commands shall be written as specified by *flags*.
- 94798 2. After the completion of an *ex* command with a flag as an argument, the  
94799 current line shall be written as specified by *flags*, unless the current line was  
94800 the last line written by the command.

94801 The characters '+' and '-' cause the value of the current line after the execution  
94802 of the *ex* command to be adjusted by the offset address as described in [Addressing](#)  
94803 [in ex](#) (on page 2845). This adjustment shall occur before the current line is written  
94804 as described in 2. above.

94805 The default for *flags* shall be none.

94806 *buffer* One of a number of named areas for holding text. The named buffers are specified  
94807 by the alphanumeric characters of the POSIX locale. There shall also be one  
94808 "unnamed" buffer. When no buffer is specified for editor commands that use a  
94809 buffer, the unnamed buffer shall be used. Commands that store text into buffers  
94810 shall store the text as it was before the command took effect, and shall store text  
94811 occurring earlier in the file before text occurring later in the file, regardless of how  
94812 the text region was specified. Commands that store text into buffers shall store the  
94813 text into the unnamed buffer as well as any specified buffer.

94814 In *ex* commands, buffer names are specified as the name by itself. In open or visual  
94815 mode commands the name is preceded by a double-quote ('"') character.

94816 If the specified buffer name is an uppercase character, and the buffer contents are  
94817 to be modified, the buffer shall be appended to rather than being overwritten. If  
94818 the buffer is not being modified, specifying the buffer name in lowercase and  
94819 uppercase shall have identical results.

94820 There shall also be buffers named by the numbers 1 through 9. In open and visual  
94821 mode, if a region of text including characters from more than a single line is being  
94822 modified by the *vi* **c** or **d** commands, the motion character associated with the **c** or  
94823 **d** commands specifies that the buffer text shall be in line mode, or the commands  
94824 **%**, **`**, **/**, **?**, **(**, **)**, **N**, **n**, **{**, or **}** are used to define a region of text for the **c** or **d** commands,  
94825 the contents of buffers 1 through 8 shall be moved into the buffer named by the  
94826 next numerically greater value, the contents of buffer 9 shall be discarded, and the  
94827 region of text shall be copied into buffer 1. This shall be in addition to copying the  
94828 text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can  
94829 be specified as a source buffer for open and visual mode commands; however,  
94830 specifying a numeric buffer as the write target of an open or visual mode  
94831 command shall have unspecified results.

94832 The text of each buffer shall have the characteristic of being in either line or  
94833 character mode. Appending text to a non-empty buffer shall set the mode to match  
94834 the characteristic of the text being appended. Appending text to a buffer shall  
94835 cause the creation of at least one additional line in the buffer. All text stored into  
94836 buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers  
94837 as the source of text specify individually how buffers of different modes are  
94838 handled. Each open or visual mode command that uses buffers for any purpose  
94839 specifies individually the mode of the text stored into the buffer and how buffers  
94840 of different modes are handled.

94841 *file* Command text used to derive a pathname. The default shall be the current  
 94842 pathname, as defined previously, in which case, if no current pathname has yet  
 94843 been established it shall be an error, except where specifically noted in the  
 94844 individual command descriptions that follow. If the command text contains any of  
 94845 the characters '~', '{', '[', '\*', '?', '\$', '"', backquote, single-quote, and  
 94846 <backslash>, it shall be subjected to the process of "shell expansions", as described  
 94847 below; if more than a single pathname results and the command expects only one,  
 94848 it shall be an error.

94849 The process of shell expansions in the editor shall be done as follows. The *ex* utility  
 94850 shall pass two arguments to the program named by the shell edit option; the first  
 94851 shall be `-c`, and the second shall be the string "echo" and the command text as a  
 94852 single argument. The standard output and standard error of that command shall  
 94853 replace the command text.

94854 **!** A character that can be appended to the command name to modify its operation,  
 94855 as detailed in the individual command descriptions. With the exception of the *ex*  
 94856 **read**, **write**, and **!** commands, the '!' character shall only act as a modifier if there  
 94857 are no <blank> characters between it and the command name.

94858 *remembered search direction*

94859 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in  
 94860 the edit buffer based on a remembered search direction, which is initially unset,  
 94861 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* **/** and **?** commands.

## 94862 Abbreviate

94863 *Synopsis:* `ab[breviate] [lhs rhs]`

94864 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

94865 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable  
 94866 characters and <blank> characters shall not be restricted. Additional restrictions shall be  
 94867 implementation-defined.

94868 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the character  
 94869 shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.

94870 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a  
 94871 <control>-V character is entered after a word character, a check shall be made for a set of  
 94872 characters matching *lhs*, in the text input entered during this command. If it is found, the effect  
 94873 shall be as if *rhs* was entered instead of *lhs*.

94874 The set of characters that are checked is defined as follows:

- 94875 1. If there are no characters inserted before the word and non-word or <ESC> characters  
 94876 that triggered the check, the set of characters shall consist of the word character.
- 94877 2. If the character inserted before the word and non-word or <ESC> characters that  
 94878 triggered the check is a word character, the set of characters shall consist of the characters  
 94879 inserted immediately before the triggering characters that are word characters, plus the  
 94880 triggering word character.
- 94881 3. If the character inserted before the word and non-word or <ESC> characters that  
 94882 triggered the check is not a word character, the set of characters shall consist of the  
 94883 characters that were inserted before the triggering characters that are neither <blank>  
 94884 characters nor word characters, plus the triggering word character.

94885 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**  
94886 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the  
94887 effect of the command shall be as if the replacement had not occurred.

94888 *Current line*: Unchanged.

94889 *Current column*: Unchanged.

## 94890 **Append**

94891 *Synopsis*: `[1addr] a[ppend] [!]`

94892 Enter *ex* text input mode; the input text shall be placed after the specified line. If line zero is  
94893 specified, the text shall be placed at the beginning of the edit buffer.

94894 This command shall be affected by the **number** and **autoindent** edit options; following the  
94895 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
94896 duration of this command only.

94897 *Current line*: Set to the last input line; if no lines were input, set to the specified line, or to the first  
94898 line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

94899 *Current column*: Set to non-<blank>.

## 94900 **Arguments**

94901 *Synopsis*: `ar[gs]`

94902 Write the current argument list, with the current argument-list entry, if any, between '[' and  
94903 ']' characters.

94904 *Current line*: Unchanged.

94905 *Current column*: Unchanged.

## 94906 **Change**

94907 *Synopsis*: `[2addr] c[hange] [!][count]`

94908 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall  
94909 be copied into the unnamed buffer, which shall become a line mode buffer.

94910 This command shall be affected by the **number** and **autoindent** edit options; following the  
94911 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
94912 duration of this command only.

94913 *Current line*: Set to the last input line; if no lines were input, set to the line before the first  
94914 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to  
94915 zero if the edit buffer is empty.

94916 *Current column*: Set to non-<blank>.

94917 **Change Directory**

94918 *Synopsis:*    chd[ir][!][*directory*]  
 94919            cd[!][*directory*]

94920 Change the current working directory to *directory*.

94921 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null  
 94922 and non-empty value, *directory* shall default to the value named in the *HOME* environment  
 94923 variable. If the *HOME* environment variable is empty or is undefined, the default value of  
 94924 *directory* is implementation-defined.

94925 If no '!' is appended to the command name, and the edit buffer has been modified since the  
 94926 last complete write, and the current pathname does not begin with a '/', it shall be an error.

94927 *Current line:* Unchanged.

94928 *Current column:* Unchanged.

94929 **Copy**

94930 *Synopsis:*    [*2addr*] co[py] *laddr* [*flags*]  
 94931            [*2addr*] t *laddr* [*flags*]

94932 Copy the specified lines after the specified destination line; line zero specifies that the lines shall  
 94933 be placed at the beginning of the edit buffer.

94934 *Current line:* Set to the last line copied.

94935 *Current column:* Set to non-<blank>.

94936 **Delete**

94937 *Synopsis:*    [*2addr*] d[ele]te[ ] [*buffer*] [*count*] [*flags*]

94938 Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a  
 94939 line-mode buffer.

94940 Flags can immediately follow the command name; see [Command Line Parsing in ex](#) (on page  
 94941 2846).

94942 *Current line:* Set to the line following the deleted lines, or to the last line in the edit buffer if that  
 94943 line is past the end of the edit buffer, or to zero if the edit buffer is empty.

94944 *Current column:* Set to non-<blank>.

94945 **Edit**

94946 *Synopsis:*    e[dit][!][+*command*] [*file*]  
 94947            ex[!][+*command*] [*file*]

94948 If no '!' is appended to the command name, and the edit buffer has been modified since the  
 94949 last complete write, it shall be an error.

94950 If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*,  
 94951 and set the current pathname to *file*. If *file* is not specified, replace the current contents of the  
 94952 edit buffer with the current contents of the file named by the current pathname. If for any reason  
 94953 the current contents of the file cannot be accessed, the edit buffer shall be empty.

94954 The +*command* option shall be <blank>-delimited; <blank> characters within the +*command* can  
 94955 be escaped by preceding them with a <backslash> character. The +*command* shall be interpreted



94956 as an *ex* command immediately after the contents of the edit buffer have been replaced and the  
94957 current line and column have been set.

94958 If the edit buffer is empty:

94959 *Current line*: Set to 0.

94960 *Current column*: Set to 1.

94961 Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:

94962 *Current line*: Set to the last line of the edit buffer.

94963 *Current column*: Set to non-<blank>.

94964 Otherwise, if *file* is omitted or results in the current pathname:

94965 *Current line*: Set to the first line of the edit buffer.

94966 *Current column*: Set to non-<blank>.

94967 Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if  
94968 the file was previously edited, the line and column may be set as follows:

94969 *Current line*: Set to the last value held when that file was last edited. If this value is not a valid  
94970 line in the new edit buffer, set to the first line of the edit buffer.

94971 *Current column*: If the current line was set to the last value held when the file was last edited, set  
94972 to the last value held when the file was last edited. Otherwise, or if the last value is not a valid  
94973 column in the new edit buffer, set to non-<blank>.

94974 Otherwise:

94975 *Current line*: Set to the first line of the edit buffer.

94976 *Current column*: Set to non-<blank>.

## 94977 **File**

94978 *Synopsis*: `f[file] [file]`

94979 If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and  
94980 the current pathname shall be set to *file*.

94981 Write an informational message. If the file has a current pathname, it shall be included in this  
94982 message; otherwise, the message shall indicate that there is no current pathname. If the edit  
94983 buffer contains lines, the current line number and the number of lines in the edit buffer shall be  
94984 included in this message; otherwise, the message shall indicate that the edit buffer is empty. If  
94985 the edit buffer has been modified since the last complete write, this fact shall be included in this  
94986 message. If the **readonly** edit option is set, this fact shall be included in this message. The  
94987 message may contain other unspecified information.

94988 *Current line*: Unchanged.

94989 *Current column*: Unchanged.

94990 **Global**

94991 *Synopsis:* `[2addr] g[lobal] /pattern/ [commands]`  
 94992 `[2addr] v /pattern/ [commands]`

94993 The optional `!` character after the **global** command shall be the same as executing the **v**  
 94994 command.

94995 If *pattern* is empty (for example, `"/"/`) or not specified, the last regular expression used in the  
 94996 editor command shall be used as the *pattern*. The *pattern* can be delimited by `<slash>` characters  
 94997 (shown in the Synopsis), as well as any non-alphanumeric or non-`<blank>` other than  
 94998 `<backslash>`, `<vertical-line>`, `<newline>`, or double-quote. Within the pattern, in certain  
 94999 circumstances the delimiter can be used as a literal character; see [Regular Expressions in ex](#) (on  
 95000 page 2875).

95001 If no lines are specified, the lines shall default to the entire file.

95002 The **global** and **v** commands are logically two-pass operations. First, mark the lines within the  
 95003 specified lines for which the line excluding the terminating `<newline>` matches (**global**) or does  
 95004 not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by  
 95005 *commands*, with the current line (`'.'`) set to each marked line. If an error occurs during this  
 95006 process, or the contents of the edit buffer are replaced (for example, by the *ex* **edit** command) an  
 95007 error message shall be written and no more commands resulting from the execution of this  
 95008 command shall be processed.

95009 Multiple *ex* commands can be specified by entering multiple commands on a single line using a  
 95010 `<vertical-line>` to delimit them, or one per line, by escaping each `<newline>` with a `<backslash>`.

95011 If no commands are specified:

- 95012 1. If in *ex* command mode, it shall be as if the **print** command were specified.
- 95013 2. Otherwise, no command shall be executed.

95014 For the **append**, **change**, and **insert** commands, the input text shall be included as part of the  
 95015 command, and the terminating `<period>` can be omitted if the command ends the list of  
 95016 commands. The **open** and **visual** commands can be specified as one of the commands, in which  
 95017 case each marked line shall cause the editor to enter open or visual mode. If open or visual mode  
 95018 is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open  
 95019 or visual mode reentered, until the list of marked lines is exhausted.

95020 The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted  
 95021 by commands executed for lines occurring earlier in the file than the marked lines. In this case,  
 95022 no commands shall be executed for the deleted lines.

95023 If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

95024 The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v**  
 95025 command.

95026 *Current line:* If no commands executed, set to the last marked line. Otherwise, as specified for the  
 95027 executed *ex* commands.

95028 *Current column:* If no commands are executed, set to non-`<blank>`; otherwise, as specified for the  
 95029 individual *ex* commands.

95030 **Insert**95031 *Synopsis:* `[1addr] i[nsert] [!]`95032 Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero  
95033 or 1, the text shall be placed at the beginning of the edit buffer.95034 This command shall be affected by the **number** and **autoindent** edit options; following the  
95035 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the  
95036 duration of this command only.95037 *Current line:* Set to the last input line; if no lines were input, set to the line before the specified  
95038 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero  
95039 if the edit buffer is empty.95040 *Current column:* Set to non-<blank>.95041 **Join**95042 *Synopsis:* `[2addr] j[oin] [!] [count] [flags]`95043 If *count* is specified:95044 If no address was specified, the **join** command shall behave as if *2addr* were the current  
95045 line and the current line plus *count* (*.,. + count*).95046 If one address was specified, the **join** command shall behave as if *2addr* were the specified  
95047 address and the specified address plus *count* (*addr,addr + count*).95048 If two addresses were specified, the **join** command shall behave as if an additional  
95049 address, equal to the last address plus *count* -1 (*addr1,addr2,addr2 + count -1*), was  
95050 specified.95051 If this would result in a second address greater than the last line of the edit buffer, it shall  
95052 be corrected to be equal to the last line of the edit buffer.95053 If no *count* is specified:95054 If no address was specified, the **join** command shall behave as if *2addr* were the current  
95055 line and the next line (*.,. +1*).95056 If one address was specified, the **join** command shall behave as if *2addr* were the specified  
95057 address and the next line (*addr,addr +1*).95058 Join the text from the specified lines together into a single line, which shall replace the specified  
95059 lines.95060 If a '!' character is appended to the command name, the **join** shall be without modification of  
95061 any line, independent of the current locale.95062 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for  
95063 each subsequent line, proceed as follows:

- 95064
1. Discard leading <space> characters from the line to be joined.
  - 95065 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
  - 95066 3. If the current line ends in a <blank>, or the first character of the line to be joined is a ' ) '  
95067 character, join the lines without further modification.

- 95068 4. If the last character of the current line is a ' . ', join the lines with two <space> characters  
 95069 between them.
- 95070 5. Otherwise, join the lines with a single <space> between them.

95071 *Current line*: Set to the first line specified.

95072 *Current column*: Set to non-<blank>.

### 95073 List

95074 *Synopsis*: `[2addr] l[ist] [count] [flags]`

95075 This command shall be equivalent to the *ex* command:

95076 `[2addr] p[rint] [count] l[flags]`

95077 See [Print](#) (on page 2864).

### 95078 Map

95079 *Synopsis*: `map[!][lhs rhs]`

95080 If *lhs* and *rhs* are not specified:

- 95081 1. If '!' is specified, write the current list of text input mode maps.
- 95082 2. Otherwise, write the current list of command mode maps.
- 95083 3. Do nothing more.

95084 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable  
 95085 characters and <blank> characters shall not be restricted. Additional restrictions shall be  
 95086 implementation-defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V,  
 95087 in which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V  
 95088 shall be discarded.

95089 If the character '!' is appended to the **map** command name, the mapping shall be effective  
 95090 during open or visual text input mode rather than **open** or **visual** command mode. This allows  
 95091 *lhs* to have two different **map** definitions at the same time: one for command mode and one for  
 95092 text input mode.

95093 For command mode mappings:

95094 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as  
 95095 part of the arguments to the command), the action shall be as if the corresponding *rhs* had  
 95096 been entered.

95097 If any character in the command, other than the first, is escaped using a <control>-V  
 95098 character, that character shall not be part of a match to an *lhs*.

95099 It is unspecified whether implementations shall support **map** commands where the *lhs* is  
 95100 more than a single character in length, where the first character of the *lhs* is printable.

95101 If *lhs* contains more than one character and the first character is '#', followed by a  
 95102 sequence of digits corresponding to a numbered function key, then when this function key  
 95103 is typed it shall be mapped to *rhs*. Characters other than digits following a '#' character  
 95104 also represent the function key named by the characters in the *lhs* following the '#' and  
 95105 may be mapped to *rhs*. It is unspecified how function keys are named or what function  
 95106 keys are supported.

- 95107 For text input mode mappings:
- 95108 When the *lhs* is entered as any part of text entered in open or visual text input modes, the  
95109 action shall be as if the corresponding *rhs* had been entered.
- 95110 If any character in the input text is escaped using a <control>-V character, that character  
95111 shall not be part of a match to an *lhs*.
- 95112 It is unspecified whether the *lhs* text entered for subsequent **map** or **unmap** commands is  
95113 replaced with the *rhs* text for the purposes of the screen display; regardless of whether or  
95114 not the display appears as if the corresponding *rhs* text was entered, the effect of the  
95115 command shall be as if the *lhs* text was entered.
- 95116 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,  
95117 possibly matching characters before treating the already entered characters as not matching the  
95118 *lhs*.
- 95119 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the  
95120 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those  
95121 characters shall not be remapped.
- 95122 On block-mode terminals, the mapping need not occur immediately (for example, it may occur  
95123 after the terminal transmits a group of characters to the system), but it shall achieve the same  
95124 results as if it occurred immediately.
- 95125 *Current line*: Unchanged.
- 95126 *Current column*: Unchanged.
- 95127 **Mark**
- 95128 *Synopsis:* `[laddr] ma[rk] character`  
95129 `[laddr] k character`
- 95130 Implementations shall support *character* values of a single lowercase letter of the POSIX locale  
95131 and the backquote and single-quote characters; support of other characters is implementation-  
95132 defined.
- 95133 If executing the *vi* **m** command, set the specified mark to the current line and 1-based numbered  
95134 character referenced by the current column, if any; otherwise, column position 1.
- 95135 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>  
95136 non-<newline> in the line, if any; otherwise, the last non-<newline> in the line, if any;  
95137 otherwise, column position 1.
- 95138 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a  
95139 deleted line is restored by a subsequent **undo** command, any marks previously associated with  
95140 the line, which have not been reset, shall be restored as well. Any use of a mark not associated  
95141 with a current line in the edit buffer shall be an error.
- 95142 The marks ` and ' shall be set as described previously, immediately before the following events  
95143 occur in the editor:
- 95144 1. The use of '\$' as an *ex* address
  - 95145 2. The use of a positive decimal number as an *ex* address
  - 95146 3. The use of a search command as an *ex* address

- 95147 4. The use of a mark reference as an *ex* address
- 95148 5. The use of the following open and visual mode commands: <control>-, %, (, ), [, ], {, }
- 95149 6. The use of the following open and visual mode commands: ', **G**, **H**, **L**, **M**, **z** if the current
- 95150 line will change as a result of the command
- 95151 7. The use of the open and visual mode commands: /, ?, **N**, `, **n** if the current line or column
- 95152 will change as a result of the command
- 95153 8. The use of the *ex* mode commands: **z**, **undo**, **global**, **v**

95154 For rules 1., 2., 3., and 4., the ` and ' marks shall not be set if the *ex* command is parsed as

95155 specified by rule 6.a. in [Command Line Parsing in ex](#) (on page 2846).

95156 For rules 5., 6., and 7., the ` and ' marks shall not be set if the commands are used as motion

95157 commands in open and visual mode.

95158 For rules 1., 2., 3., 4., 5., 6., 7., and 8., the ` and ' marks shall not be set if the command fails.

95159 The ` and ' marks shall be set as described previously, each time the contents of the edit buffer

95160 are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex**

95161 mode and the edit buffer is not empty, before any commands or movements (including

95162 commands or movements specified by the **-c** or **-t** options or the **+command** argument) are

95163 executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the *vi*

95164 **m** command; otherwise, as if executing the *ex* **mark** command.

95165 When changing from **ex** mode to open or visual mode, if the ` and ' marks are not already set,

95166 the ` and ' marks shall be set as described previously.

95167 *Current line*: Unchanged.

95168 *Current column*: Unchanged.

## 95169 **Move**

95170 *Synopsis*: **[2addr] m[ove] laddr [flags]**

95171 Move the specified lines after the specified destination line. A destination of line zero specifies

95172 that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the

95173 destination line is within the range of lines to be moved.

95174 *Current line*: Set to the last of the moved lines.

95175 *Current column*: Set to non-<blank>.

## 95176 **Next**

95177 *Synopsis*: **n[ext] [!] [+command] [file ...]**

95178 If no '!' is appended to the command name, and the edit buffer has been modified since the

95179 last complete write, it shall be an error, unless the file is successfully written as specified by the

95180 **autowrite** option.

95181 If one or more files is specified:

- 95182 1. Set the argument list to the specified filenames.
- 95183 2. Set the current argument list reference to be the first entry in the argument list.

95184 3. Set the current pathname to the first filename specified.

95185 Otherwise:

95186 1. It shall be an error if there are no more filenames in the argument list after the filename  
95187 currently referenced.

95188 2. Set the current pathname and the current argument list reference to the filename after the  
95189 filename currently referenced in the argument list.

95190 Replace the contents of the edit buffer with the contents of the file named by the current  
95191 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be  
95192 empty.

95193 This command shall be affected by the **autowrite** and **writeany** edit options.

95194 The *+command* option shall be <blank>-delimited; <blank> characters can be escaped by  
95195 preceding them with a <backslash> character. The *+command* shall be interpreted as an *ex*  
95196 command immediately after the contents of the edit buffer have been replaced and the current  
95197 line and column have been set.

95198 *Current line*: Set as described for the **edit** command.

95199 *Current column*: Set as described for the **edit** command.

## 95200 **Number**

95201 *Synopsis*: `[2addr] nu[mber] [count] [flags]`  
95202 `[2addr] #[count] [flags]`

95203 These commands shall be equivalent to the *ex* command:

95204 `[2addr] p[rint] [count] #[flags]`

95205 See [Print](#) (on page 2864).

## 95206 **Open**

95207 *Synopsis*: `[laddr] o[pen] /pattern/ [flags]`

95208 This command need not be supported on block-mode terminals or terminals with insufficient  
95209 capabilities. If standard input, standard output, or standard error are not terminal devices, the  
95210 results are unspecified.

95211 Enter open mode.

95212 The trailing delimiter can be omitted from *pattern* at the end of the command line. If *pattern* is  
95213 empty (for example, `"/"/`) or not specified, the last regular expression used in the editor shall  
95214 be used as the pattern. The pattern can be delimited by <slash> characters (shown in the  
95215 Synopsis), as well as any alphanumeric, or non-<blank> other than <backslash>, <vertical-line>,  
95216 <newline>, or double-quote.

95217 *Current line*: Set to the specified line.

95218 *Current column*: Set to non-<blank>.

95219 **Preserve**95220 *Synopsis:*    pre[serve]

95221 Save the edit buffer in a form that can later be recovered by using the **-r** option or by using the  
 95222 *ex* **recover** command. After the file has been preserved, a mail message shall be sent to the user.  
 95223 This message shall be readable by invoking the *mailx* utility. The message shall contain the name  
 95224 of the file, the time of preservation, and an *ex* command that could be used to recover the file.  
 95225 Additional information may be included in the mail message.

95226 *Current line:* Unchanged.95227 *Current column:* Unchanged.95228 **Print**95229 *Synopsis:*    [2addr] p[rint][count][flags]

95230 Write the addressed lines. The behavior is unspecified if the number of columns on the display is  
 95231 less than the number of columns required to write any single character in the lines being written.

95232 Non-printable characters, except for the <tab>, shall be written as implementation-defined  
 95233 multi-character sequences.

95234 If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line  
 95235 number in the following format:

95236 "%6dΔΔ", &lt;line number&gt;

95237 If the l flag is specified or the **list** edit option is set:

- 95238 1. The characters listed in XBD Table 5-1 (on page 113) shall be written as the corresponding  
 95239 escape sequence.
- 95240 2. Non-printable characters not in XBD Table 5-1 (on page 113) shall be written as one three-  
 95241 digit octal number (with a preceding <backslash>) for each byte in the character (most  
 95242 significant byte first).
- 95243 3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line  
 95244 shall be written with a preceding <backslash>.

95245 Long lines shall be folded; the length at which folding occurs is unspecified, but should be  
 95246 appropriate for the output terminal, considering the number of columns of the terminal.

95247 If a line is folded, and the l flag is not specified and the **list** edit option is not set, it is unspecified  
 95248 whether a multi-column character at the folding position is separated; it shall not be discarded.

95249 *Current line:* Set to the last written line.95250 *Current column:* Unchanged if the current line is unchanged; otherwise, set to non-<blank>.95251 **Put**95252 *Synopsis:*    [laddr] pu[t][buffer]

95253 Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line  
 95254 zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a  
 95255 line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

95256 *Current line:* Set to the last line entered into the edit buffer.95257 *Current column:* Set to non-<blank>.



95258 **Quit**95259 *Synopsis:* `q[uit][!]`95260 If no `!` is appended to the command name:

- 95261 1. If the edit buffer has been modified since the last complete write, it shall be an error.
- 95262 2. If there are filenames in the argument list after the filename currently referenced, and the
- 95263 last command was not a **quit**, **wq**, **xit**, or **ZZ** (see [Exit](#), on page 3561) command, it shall be
- 95264 an error.

95265 Otherwise, terminate the editing session.

95266 **Read**95267 *Synopsis:* `[laddr] r[ead][!][file]`

95268 If `!` is not the first non-`<blank>` to follow the command name, a copy of the specified file shall

95269 be appended into the edit buffer after the specified line; line zero specifies that the copy shall be

95270 placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If

95271 no *file* is named, the current pathname shall be the default. If there is no current pathname, then

95272 *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be

95273 an error. Specifying a *file* that is not of type regular shall have unspecified results.

95274 Otherwise, if *file* is preceded by `!`, the rest of the line after the `!` shall have `%`, `#`, and

95275 `!` characters expanded as described in [Command Line Parsing in ex](#) (on page 2846).

95276 The *ex* utility shall then pass two arguments to the program named by the shell edit option; the

95277 first shall be `-c` and the second shall be the expanded arguments to the **read** command as a

95278 single argument. The standard input of the program shall be set to the standard input of the *ex*

95279 program when it was invoked. The standard error and standard output of the program shall be

95280 appended into the edit buffer after the specified line.

95281 Each line in the copied file or program output (as delimited by `<newline>` characters or the end

95282 of the file or output if it is not immediately preceded by a `<newline>`), shall be a separate line in

95283 the edit buffer. Any occurrences of `<carriage-return>` and `<newline>` pairs in the output shall be

95284 treated as single `<newline>` characters.

95285 The special meaning of the `!` following the **read** command can be overridden by escaping it

95286 with a `<backslash>` character.

95287 *Current line:* If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual

95288 mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into

95289 the edit buffer.

95290 *Current column:* Set to non-`<blank>`.95291 **Recover**95292 *Synopsis:* `rec[over][!]file`

95293 If no `!` is appended to the command name, and the edit buffer has been modified since the

95294 last complete write, it shall be an error.

95295 If no *file* operand is specified, then the current pathname shall be used. If there is no current

95296 pathname or *file* operand, it shall be an error.

95297 If no recovery information has previously been saved about *file*, the **recover** command shall

95298 behave identically to the **edit** command, and an informational message to this effect shall be

95299 written.

95300 Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer  
 95301 with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the  
 95302 one most recently saved shall be recovered, and an informational message that there are  
 95303 previous versions of the file that can be recovered shall be written. The editor shall behave as if  
 95304 the contents of the edit buffer have already been modified.

95305 *Current file*: Set as described for the **edit** command.

95306 *Current column*: Set as described for the **edit** command.

## 95307 **Rewind**

95308 *Synopsis*: `rew[ind] [!]`

95309 If no `!` is appended to the command name, and the edit buffer has been modified since the  
 95310 last complete write, it shall be an error, unless the file is successfully written as specified by the  
 95311 **autowrite** option.

95312 If the argument list is empty, it shall be an error.

95313 The current argument list reference and the current pathname shall be set to the first filename in  
 95314 the argument list.

95315 Replace the contents of the edit buffer with the contents of the file named by the current  
 95316 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be  
 95317 empty.

95318 This command shall be affected by the **autowrite** and **writeany** edit options.

95319 *Current line*: Set as described for the **edit** command.

95320 *Current column*: Set as described for the **edit** command.

## 95321 **Set**

95322 *Synopsis*: `se[t] [ option[=value] ... ] [ nooption ... ] [ option? ... ] [ all ]`

95323 When no arguments are specified, write the value of the **term** edit option and those options  
 95324 whose values have been changed from the default settings; when the argument *all* is specified,  
 95325 write all of the option values.

95326 Giving an option name followed by the character `?` shall cause the current value of that  
 95327 option to be written. The `?` can be separated from the option name by zero or more `<blank>`  
 95328 characters. The `?` shall be necessary only for Boolean valued options. Boolean options can be  
 95329 given values by the form **set option** to turn them on or **set nooption** to turn them off; string and  
 95330 numeric options can be assigned by the form **set option=*value***. Any `<blank>` characters in strings  
 95331 can be included as is by preceding each `<blank>` with an escaping `<backslash>`. More than one  
 95332 option can be set or listed by a single set command by specifying multiple arguments, each  
 95333 separated from the next by one or more `<blank>` characters. Arguments can appear in any order  
 95334 and shall be processed in the specified order.

95335 See [Edit Options in ex](#) (on page 2877) for details about specific options.

95336 *Current line*: Unchanged.

95337 *Current column*: Unchanged.

95338 **Shell**95339 *Synopsis:* `sh[ell]`95340 Invoke the program named in the **shell** edit option with the single argument **-i** (interactive  
95341 mode). Editing shall be resumed when the program exits.95342 *Current line:* Unchanged.95343 *Current column:* Unchanged.95344 **Source**95345 *Synopsis:* `so[urce] file`95346 Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.95347 *Current line:* As specified for the individual *ex* commands.95348 *Current column:* As specified for the individual *ex* commands.95349 **Substitute**95350 *Synopsis:* `[2addr] s[substitute] [/pattern/repl/] [options] [count] [flags]`  
95351 `[2addr] & [options] [count] [flags]`  
95352 `[2addr] ~ [options] [count] [flags]`95353 Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See  
95354 [Regular Expressions in ex](#) (on page 2875) and [Replacement Strings in ex](#) (on page 2876).) Any  
95355 non-alphabetic, non-`<blank>` delimiter other than `<backslash>`, `'|'`, `<newline>`, or double-  
95356 quote can be used instead of `'/'`. Within the pattern, in certain circumstances the delimiter can  
95357 be used as a literal character; see [Regular Expressions in ex](#) (on page 2875). Within the  
95358 replacement, the delimiter shall not terminate the replacement if it is the second character of an  
95359 escape sequence (see XBD [Section 9.1](#), on page 179) and the escaped delimiter shall be treated as  
95360 that literal character in the replacement (losing any special meaning it would have had if it was  
95361 not used as the delimiter and was not escaped). It shall be an error if the substitution fails on  
95362 every addressed line.95363 The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If  
95364 both *pattern* and *repl* are not specified or are empty (for example, `"/"/`), the last `s` command  
95365 shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in  
95366 the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be  
95367 replaced by nothing. If the entire replacement pattern is `'%'`, the last replacement pattern to an  
95368 `s` command shall be used.95369 Entering a `<carriage-return>` in *repl* (which requires an escaping `<backslash>` in *ex* mode and an  
95370 escaping `<control>-V` in open or *vi* mode) shall split the line at that point, creating a new line in  
95371 the edit buffer. The `<carriage-return>` shall be discarded.95372 If *options* includes the letter `'g'` (**global**), all non-overlapping instances of the pattern in the line  
95373 shall be replaced.95374 If *options* includes the letter `'c'` (**confirm**), then before each substitution the line shall be written;  
95375 the written line shall reflect all previous substitutions. On the following line, `<space>` characters  
95376 shall be written beneath the characters from the line that are before the *pattern* to be replaced,  
95377 and `'^'` characters written beneath the characters included in the *pattern* to be replaced. The *ex*  
95378 utility shall then wait for a response from the user. An affirmative response shall cause the  
95379 substitution to be done, while any other input shall not make the substitution. An affirmative  
95380 response shall consist of a line with the affirmative response (as defined by the current locale) at

95381 the beginning of the line. This line shall be subject to editing in the same way as the *ex* command  
95382 line.

95383 If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the  
95384 user shall be preserved in the edit buffer after the interrupt.

95385 If the remembered search direction is not set, the *s* command shall set it to forward.

95386 In the second Synopsis, the *&* command shall repeat the previous substitution, as if the *&*  
95387 command were replaced by:

95388 *s/pattern/repl/*

95389 where *pattern* and *repl* are as specified in the previous *s*, *&*, or *~* command.

95390 In the third Synopsis, the *~* command shall repeat the previous substitution, as if the '*~*' were  
95391 replaced by:

95392 *s/pattern/repl/*

95393 where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from  
95394 the previous substitution (including *&* and *~*) command.

95395 These commands shall be affected by the *LC\_MESSAGES* environment variable.

95396 *Current line*: Set to the last line in which a substitution occurred, or, unchanged if no substitution  
95397 occurred.

95398 *Current column*: Set to non-<blank>.

## 95399 Suspend

95400 *Synopsis:* *su[spend] [!]*

95401 *st[op] [!]*

95402 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the  
95403 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell  
95404 (see the description of *set -m*).

95405 These commands shall be affected by the **autowrite** and **writeany** edit options.

95406 The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

## 95407 Tag

95408 *Synopsis:* *ta[g] [!] tagstring*

95409 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see  
95410 *ctags*) description.

95411 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in  
95412 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from  
95413 beginning to end. If no reference is found, it shall be an error and an error message to this effect  
95414 shall be written. If the reference is not found, or if an error occurs while processing a file referred  
95415 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first  
95416 occurrence of such an error.

95417 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression  
95418 used in the editor; for example, for the purposes of the *s* command.

95419 If the *tagstring* is in a file with a different name than the current pathname, set the current  
95420 pathname to the name of that file, and replace the contents of the edit buffer with the contents of

95421 that file. In this case, if no '!' is appended to the command name, and the edit buffer has been  
 95422 modified since the last complete write, it shall be an error, unless the file is successfully written  
 95423 as specified by the **autowrite** option.

95424 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

95425 *Current line*: If the tags file contained a line number, set to that line number. If the line number is  
 95426 larger than the last line in the edit buffer, an error message shall be written and the current line  
 95427 shall be set as specified for the **edit** command.

95428 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no  
 95429 matching pattern is found, an error message shall be written and the current line shall be set as  
 95430 specified for the **edit** command.

95431 *Current column*: If the tags file contained a line-number reference and that line-number was not  
 95432 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern  
 95433 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

### 95434 **Unabbreviate**

95435 *Synopsis*:    una[bbrev] lhs

95436 If *lhs* is not an entry in the current list of abbreviations (see [Abbreviate](#), on page 2854), it shall be  
 95437 an error. Otherwise, delete *lhs* from the list of abbreviations.

95438 *Current line*: Unchanged.

95439 *Current column*: Unchanged.

### 95440 **Undo**

95441 *Synopsis*:    u[ndo]

95442 Reverse the changes made by the last command that modified the contents of the edit buffer,  
 95443 including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands  
 95444 resulting from buffer executions and mapped character expansions, are considered single  
 95445 commands.

95446 If no action that can be undone preceded the **undo** command, it shall be an error.

95447 If the **undo** command restores lines that were marked, the mark shall also be restored unless it  
 95448 was reset subsequent to the deletion of the lines.

95449 *Current line*:

- 95450     1. If lines are added or changed in the file, set to the first line added or changed.
- 95451     2. Set to the line before the first line deleted, if it exists.
- 95452     3. Set to 1 if the edit buffer is not empty.
- 95453     4. Set to zero.

95454 *Current column*: Set to non-<blank>.

95455 **Unmap**95456 *Synopsis:* `unm[ap][!] lhs`95457 If '!' is appended to the command name, and if *lhs* is not an entry in the list of text input mode  
95458 map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map  
95459 definitions.95460 If no '!' is appended to the command name, and if *lhs* is not an entry in the list of command  
95461 mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode  
95462 map definitions.95463 *Current line:* Unchanged.95464 *Current column:* Unchanged.95465 **Version**95466 *Synopsis:* `ve[rsion]`95467 Write a message containing version information for the editor. The format of the message is  
95468 unspecified.95469 *Current line:* Unchanged.95470 *Current column:* Unchanged.95471 **Visual**95472 *Synopsis:* `[laddr] vi[sual][type][count][flags]`95473 If *ex* is currently in open or visual mode, the Synopsis and behavior of the visual command shall  
95474 be the same as the **edit** command, as specified by **Edit** (on page 2856).95475 Otherwise, this command need not be supported on block-mode terminals or terminals with  
95476 insufficient capabilities. If standard input, standard output, or standard error are not terminal  
95477 devices, the results are unspecified.95478 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in  
95479 **window**, on page 2883). If the '^' type character was also specified, the **window** edit option  
95480 shall be set before being used by the type character.95481 Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type*  
95482 shall cause the following effects:

95483 + Place the beginning of the specified line at the top of the display.

95484 - Place the end of the specified line at the bottom of the display.

95485 . Place the beginning of the specified line in the middle of the display.

95486 ^ If the specified line is less than or equal to the value of the **window** edit option, set the line  
95487 to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place  
95488 the beginning of this line as close to the bottom of the displayed lines as possible, while still  
95489 displaying the value of the **window** edit option number of lines.95490 *Current line:* Set to the specified line.95491 *Current column:* Set to non-<blank>.

95492 **Write**

95493 *Synopsis:* `[2addr] w[rite] [!] [>>] [file]`  
 95494 `[2addr] w[rite] [!] [file]`  
 95495 `[2addr] wq[!] [>>] [file]`

95496 If no lines are specified, the lines shall default to the entire file.

95497 The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!**  
 95498 shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the  
 95499 **quit** shall not be attempted.

95500 If the command name is not followed by one or more <blank> characters, or *file* is not preceded  
 95501 by a **'!'** character, the **write** shall be to a file.

- 95502 1. If the **>>** argument is specified, and the file already exists, the lines shall be appended to  
 95503 the file instead of replacing its contents. If the **>>** argument is specified, and the file does  
 95504 not already exist, it is unspecified whether the write shall proceed as if the **>>** argument  
 95505 had not been specified or if the write shall fail.
- 95506 2. If the **readonly** edit option is set (see [readonly](#), on page 2880), the **write** shall fail.
- 95507 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.
- 95508 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,  
 95509 the **write** command shall fail.
- 95510 5. If the current pathname is used, and the current pathname has been changed by the **file**  
 95511 or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,  
 95512 subsequent **writes** shall not fail for this reason (unless the current pathname is changed  
 95513 again).
- 95514 6. If the whole edit buffer is not being written, and the file to be written exists, the **write**  
 95515 shall fail.

95516 For rules 1., 2., 3., and 5., the **write** can be forced by appending the character **'!'** to the  
 95517 command name.

95518 For rules 2., 3., and 5., the **write** can be forced by setting the **writeany** edit option.

95519 Additional, implementation-defined tests may cause the **write** to fail.

95520 If the edit buffer is empty, a file without any contents shall be written.

95521 An informational message shall be written noting the number of lines and bytes written.

95522 Otherwise, if the command is followed by one or more <blank> characters, and the file is  
 95523 preceded by **'!'**, the rest of the line after the **'!'** shall have **'%'**, **'#'**, and **'!'** characters  
 95524 expanded as described in [Command Line Parsing in ex](#) (on page 2846).

95525 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the  
 95526 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a  
 95527 single argument. The specified lines shall be written to the standard input of the command. The  
 95528 standard error and standard output of the program, if any, shall be written as described for the  
 95529 **print** command. If the last character in that output is not a <newline>, a <newline> shall be  
 95530 written at the end of the output.

95531 The special meaning of the **'!'** following the **write** command can be overridden by escaping it  
 95532 with a <backslash> character.

95533 *Current line:* Unchanged.

95534 *Current column*: Unchanged.

### 95535 **Write and Exit**

95536 *Synopsis*: `[2addr] x[it] [!] [file]`

95537 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to  
95538 the **quit** command, or if a '!' is appended to the command name, to **quit!**.

95539 Otherwise, **xit** shall be equivalent to the **wq** command, or if a '!' is appended to the command  
95540 name, to **wq!**.

95541 *Current line*: Unchanged.

95542 *Current column*: Unchanged.

### 95543 **Yank**

95544 *Synopsis*: `[2addr] ya[nk] [ buffer] [count]`

95545 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall  
95546 become a line-mode buffer.

95547 *Current line*: Unchanged.

95548 *Current column*: Unchanged.

### 95549 **Adjust Window**

95550 *Synopsis*: `[laddr] z[!][type ...] [count] [flags]`

95551 If no line is specified, the current line shall be the default; if *type* is omitted as well, the current  
95552 line value shall first be incremented by 1. If incrementing the current line would cause it to be  
95553 greater than the last line in the edit buffer, it shall be an error.

95554 If there are <blank> characters between the *type* argument and the preceding **z** command name  
95555 or optional '!' character, it shall be an error.

95556 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in  
95557 [window](#), on page 2883). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit  
95558 option, or if ! was specified, the number of lines in the display minus 1.

95559 If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise,  
95560 *count* lines starting with the line specified by the *type* argument shall be written.

95561 The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

95562 – The specified line shall be decremented by the following value:

95563  $((\text{number of '-' characters}) \times \text{count}) - 1$

95564 If the calculation would result in a number less than 1, it shall be an error. Write lines from  
95565 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit  
95566 buffer has been written.

95567 + The specified line shall be incremented by the following value:

95568  $((\text{number of '+' characters}) - 1) \times \text{count} + 1$

95569 If the calculation would result in a number greater than the last line in the edit buffer, it  
95570 shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count*  
95571 lines or the last line in the edit buffer has been written.



95572 =, . If more than a single ' . ' or '=' is specified, it shall be an error. The following steps shall  
95573 be taken:

- 95574 1. If *count* is zero, nothing shall be written.
- 95575 2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count*  
95576 or '!' was specified, *N* shall be:
- 95577  $(count - 1) / 2$
- 95578 Otherwise, *N* shall be:
- 95579  $(count - 3) / 2$
- 95580 If *N* is a number less than 3, no lines shall be written.
- 95581 3. If '=' was specified as the type character, write a line consisting of the smaller of the  
95582 number of columns in the display divided by two, or 40 '-' characters.
- 95583 4. Write the current line.
- 95584 5. Repeat step 3.
- 95585 6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall  
95586 be defined as in step 2. If *N* is a number less than 3, no lines shall be written. If *count*  
95587 is less than 3, no lines shall be written.

95588 ^ The specified line shall be decremented by the following value:

95589  $((\text{number of '^' characters}) + 1) \times count - 1$

95590 If the calculation would result in a number less than 1, it shall be an error. Write lines from  
95591 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit  
95592 buffer has been written.

95593 *Current line*: Set to the last line written, unless the type is =, in which case, set to the specified  
95594 line.

95595 *Current column*: Set to non-<blank>.

## 95596 Escape

95597 *Synopsis*: !*command*  
95598 [2*addr*] !*command*

95599 The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as  
95600 described in [Command Line Parsing in ex](#) (on page 2846). If the expansion causes the text of the  
95601 line to change, it shall be redisplayed, preceded by a single '!' character.

95602 The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two  
95603 arguments to the program; the first shall be -c, and the second shall be the expanded arguments  
95604 to the ! command as a single argument.

95605 If no lines are specified, the standard input, standard output, and standard error of the program  
95606 shall be set to the standard input, standard output, and standard error of the *ex* program when it  
95607 was invoked. In addition, a warning message shall be written if the edit buffer has been  
95608 modified since the last complete write, and the **warn** edit option is set.

95609 If lines are specified, they shall be passed to the program as standard input, and the standard  
95610 output and standard error of the program shall replace those lines in the edit buffer. Each line in  
95611 the program output (as delimited by <newline> characters or the end of the output if it is not  
95612 immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any

95613 occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single  
 95614 <newline> characters. The specified lines shall be copied into the unnamed buffer before they  
 95615 are replaced, and the unnamed buffer shall become a line-mode buffer.

95616 If in *ex* mode, a single '!' character shall be written when the program completes.

95617 This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this  
 95618 command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this  
 95619 command shall be affected by the **autoprint** edit option.

95620 *Current line:*

- 95621 1. If no lines are specified, unchanged.
- 95622 2. Otherwise, set to the last line read in, if any lines are read in.
- 95623 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
- 95624 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
- 95625 5. Otherwise, set to zero.

95626 *Current column:* If no lines are specified, unchanged. Otherwise, set to non-<blank>.

### 95627 **Shift Left**

95628 *Synopsis:* `[2addr] <[< . . . ] [count] [flags]`

95629 Shift the specified lines to the start of the line; the number of column positions to be shifted shall  
 95630 be the number of command characters times the value of the **shiftwidth** edit option. Only  
 95631 leading <blank> characters shall be deleted or changed into other <blank> characters in shifting;  
 95632 other characters shall not be affected.

95633 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode  
 95634 buffer.

95635 This command shall be affected by the **autoprint** edit option.

95636 *Current line:* Set to the last line in the lines specified.

95637 *Current column:* Set to non-<blank>.

### 95638 **Shift Right**

95639 *Synopsis:* `[2addr] >[> . . . ] [count] [flags]`

95640 Shift the specified lines away from the start of the line; the number of column positions to be  
 95641 shifted shall be the number of command characters times the value of the **shiftwidth** edit  
 95642 option. The shift shall be accomplished by adding <blank> characters as a prefix to the line or  
 95643 changing leading <blank> characters into other <blank> characters. Empty lines shall not be  
 95644 changed.

95645 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode  
 95646 buffer.

95647 This command shall be affected by the **autoprint** edit option.

95648 *Current line:* Set to the last line in the lines specified.

95649 *Current column:* Set to non-<blank>.

95650 **<control>-D**95651 *Synopsis:*     <control>-D95652 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the  
95653 number of lines after the current line in the edit buffer. If the current line is the last line of the  
95654 edit buffer it shall be an error.95655 *Current line:* Set to the last line written.95656 *Current column:* Set to non-<blank>.95657 **Write Line Number**95658 *Synopsis:*     [*laddr*] = [*flags*]95659 If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of  
95660 the specified line.95661 *Current line:* Unchanged.95662 *Current column:* Unchanged.95663 **Execute**95664 *Synopsis:*     [*2addr*] @ *buffer*95665                [*2addr*] \* *buffer*95666 If no *buffer* is specified or is specified as '@' or '\*', the last *buffer* executed shall be used. If no  
95667 previous *buffer* has been executed, it shall be an error.95668 For each line specified by the addresses, set the current line ('.') to the specified line, and  
95669 execute the contents of the named *buffer* (as they were at the time the @ command was executed)  
95670 as *ex* commands. For each line of a line-mode *buffer*, and all but the last line of a character-mode  
95671 *buffer*, the *ex* command parser shall behave as if the line was terminated by a <newline>.95672 If an error occurs during this process, or a line specified by the addresses does not exist when  
95673 the current line would be set to it, or more than a single line was specified by the addresses, and  
95674 the contents of the edit buffer are replaced (for example, by the *ex* :edit command) an error  
95675 message shall be written, and no more commands resulting from the execution of this command  
95676 shall be processed.95677 *Current line:* As specified for the individual *ex* commands.95678 *Current column:* As specified for the individual *ex* commands.95679 **Regular Expressions in ex**95680 The *ex* utility shall support regular expressions that are a superset of the basic regular  
95681 expressions described in XBD [Section 9.3](#) (on page 181). A null regular expression ("/") shall  
95682 be equivalent to the last regular expression encountered.95683 Regular expressions can be used in addresses to specify lines and, in some commands (for  
95684 example, the **substitute** command), to specify portions of a line to be substituted.95685 The start and end of a regular expression (RE) are marked by a delimiter character (although in  
95686 some circumstances the end delimiter can be omitted). In addresses, the delimiter is either  
95687 <slash> or <question-mark>. In commands, other characters can be used as the delimiter, as  
95688 specified in the description of the command. Within the RE (as an *ex* extension to the BRE  
95689 syntax), the delimiter shall not terminate the RE if it is the second character of an escape

95690 sequence (see XBD [Section 9.1](#), on page 179) and the escaped delimiter shall be treated as that  
 95691 literal character in the RE (losing any special meaning it would have had if it was not used as the  
 95692 delimiter and was not escaped). In addition, the delimiter character shall not terminate the RE  
 95693 when it appears within a bracket expression, and shall have its normal meaning in the bracket  
 95694 expression. For example, the command "g[%]p" is equivalent to "g/[%]/p", and the  
 95695 command "s-[0-9]--g" is equivalent to "s/[0-9]//g".

95696 The following constructs can be used to enhance the basic regular expressions:

95697 \< Match the beginning of a *word*. (See the definition of *word* at the beginning of [Command](#)  
 95698 [Descriptions in ex](#) (on page 2852).)

95699 \> Match the end of a *word*.

95700 ~ Match the replacement part of the last **substitute** command. The <tilde> ('~') character can  
 95701 be escaped in a regular expression to become a normal character with no special meaning.  
 95702 The <backslash> shall be discarded.

95703 When the editor option **magic** is not set, the only characters with special meanings shall be '^'  
 95704 at the beginning of a pattern, '\$' at the end of a pattern, and <backslash>. The characters '.',  
 95705 '\*', '[', and '~' shall be treated as ordinary characters unless preceded by a <backslash>;  
 95706 when preceded by a <backslash> they shall regain their special meaning, or in the case of  
 95707 <backslash>, be handled as a single <backslash>. <backslash> characters used to escape other  
 95708 characters shall be discarded.

### 95709 Replacement Strings in ex

95710 Certain characters and strings have special meaning in replacement strings when the character,  
 95711 or the first character of the string, is unescaped.

95712 The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall  
 95713 stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not  
 95714 set) shall be replaced by the replacement part of the previous **substitute** command. The  
 95715 sequence '\n', where *n* is an integer, shall be replaced by the text matched by the  
 95716 corresponding back-reference expression. If the corresponding back-reference expression does  
 95717 not match, then the characters '\n' shall be replaced by the empty string.

95718 The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the  
 95719 replacement string (using the '\&' or "\"digit) notation. The string '\l' ('\u') shall cause  
 95720 the character that follows to be converted to lowercase (uppercase). The string '\L' ('\U') shall  
 95721 cause all characters subsequent to it to be converted to lowercase (uppercase) as they are  
 95722 inserted by the substitution until the string '\e' or '\E', or the end of the replacement string,  
 95723 is encountered.

95724 Otherwise, any character following an unescaped <backslash> shall be treated as that literal  
 95725 character, and the escaping <backslash> shall be discarded.

95726 An example of case conversion with the **s** command is as follows:

```
95727 :p
95728 The cat sat on the mat.
95729 :s/\<.at\>/\u&/gp
95730 The Cat Sat on the Mat.
95731 :s/S\ (.*)M/S\U\1\eM/p
95732 The Cat SAT ON THE Mat.
```

95733 **Edit Options in ex**

95734 The *ex* utility has a number of options that modify its behavior. These options have default  
95735 settings, which can be changed using the **set** command.

95736 Options are Boolean unless otherwise specified.

95737 **autoindent, ai**

95738 [Default *unset*]

95739 If **autoindent** is set, each line in input mode shall be indented (using first as many <tab>  
95740 characters as possible, as determined by the editor option **tabstop**, and then using <space>  
95741 characters) to align with another line, as follows:

- 95742 1. If in open or visual mode and the text input is part of a line-oriented command (see the  
95743 EXTENDED DESCRIPTION in *vi*), align to the first column.
- 95744 2. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:
  - 95745 a. If a line was previously inserted as part of this command, it shall be set to the  
95746 indentation of the last inserted line by default, or as otherwise specified for the  
95747 <control>-D character in **Input Mode Commands in vi** (on page 3561).
  - 95748 b. Otherwise, it shall be set to the indentation of the previous current line, if any;  
95749 otherwise, to the first column.
- 95750 3. For the *ex a*, *i*, and *c* commands, indentation for each line shall be set as follows:
  - 95751 a. If a line was previously inserted as part of this command, it shall be set to the  
95752 indentation of the last inserted line by default, or as otherwise specified for the *eof*  
95753 character in **Scroll** (on page 2850).
  - 95754 b. Otherwise, if the command is the *ex a* command, it shall be set to the line  
95755 appended after, if any; otherwise to the first column.
  - 95756 c. Otherwise, if the command is the *ex i* command, it shall be set to the line inserted  
95757 before, if any; otherwise to the first column.
  - 95758 d. Otherwise, if the command is the *ex c* command, it shall be set to the indentation of  
95759 the line replaced.

95760 **autoprint, ap**

95761 [Default *set*]

95762 If **autoprint** is set, the current line shall be written after each *ex* command that modifies the  
95763 contents of the current edit buffer, and after each **tag** command for which the tag search pattern  
95764 was found or tag line number was valid, unless:

- 95765 1. The command was executed while in open or visual mode.
- 95766 2. The command was executed as part of a **global** or **v** command or **@** buffer execution.
- 95767 3. The command was the form of the **read** command that reads a file into the edit buffer.
- 95768 4. The command was the **append**, **change**, or **insert** command.
- 95769 5. The command was not terminated by a <newline>.

95770 6. The current line shall be written by a flag specified to the command; for example, **delete #**  
 95771 shall write the current line as specified for the flag modifier to the **delete** command, and  
 95772 not as specified by the **autoprint** edit option.

95773 **autowrite, aw**

95774 [Default *unset*]

95775 If **autowrite** is set, and the edit buffer has been modified since it was last completely written to  
 95776 any file, the contents of the edit buffer shall be written as if the *ex* **write** command had been  
 95777 specified without arguments, before each command affected by the **autowrite** edit option is  
 95778 executed. Appending the character '!' to the command name of any of the *ex* commands  
 95779 except '!' shall prevent the write. If the write fails, it shall be an error and the command shall  
 95780 not be executed.

95781 **beautify, bf**

95782 XSI [Default *unset*]

95783 If **beautify** is set, all non-printable characters, other than <tab>, <newline>, and <form-feed>  
 95784 characters, shall be discarded from text read in from files.

95785 **directory, dir**

95786 [Default *implementation-defined*]

95787 The value of this option specifies the directory in which the editor buffer is to be placed. If this  
 95788 directory is not writable by the user, the editor shall quit.

95789 **edcompatible, ed**

95790 [Default *unset*]

95791 Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled  
 95792 by repeating the suffixes.

95793 **errorbells, eb**

95794 [Default *unset*]

95795 If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse  
 95796 video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.

95797 **exrc**

95798 [Default *unset*]

95799 If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in [Initialization in](#)  
 95800 [ex and vi](#) (on page 2842). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory  
 95801 during initialization, unless the current directory is that named by the *HOME* environment  
 95802 variable.

95803 **ignorecase, ic**95804 [Default *unset*]

95805 If **ignorecase** is set, characters that have uppercase and lowercase representations shall have  
95806 those representations considered as equivalent for purposes of regular expression comparison.

95807 The **ignorecase** edit option shall affect all remembered regular expressions; for example,  
95808 unsetting the **ignorecase** edit option shall cause a subsequent *vi n* command to search for the  
95809 last basic regular expression in a case-sensitive fashion.

95810 **list**95811 [Default *unset*]

95812 If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for  
95813 the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall  
95814 be displayed as specified for the *ex print* command with the **l** flag specified. In open or visual  
95815 text input mode, when the cursor does not rest on any character in the line, it shall rest on the  
95816 ' \$ ' marking the end of the line.

95817 **magic**95818 [Default *set*]

95819 If **magic** is set, modify the interpretation of characters in regular expressions and substitution  
95820 replacement strings (see [Regular Expressions in ex](#) (on page 2875) and [Replacement Strings in](#)  
95821 [ex](#), on page 2876).

95822 **mesg**95823 [Default *set*]

95824 If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the  
95825 terminal shall be turned on while in open or visual mode. The shell-level command *mesg n* shall  
95826 take precedence over any setting of the *ex mesg* option; that is, if **mesg y** was issued before the  
95827 editor started (or in a shell escape), such as:

95828 `:!mesg y`

95829 the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable  
95830 incoming messages if **mesg n** was issued.

95831 **number, nu**95832 [Default *unset*]

95833 If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line  
95834 numbers, in the format specified by the **print** command with the **#** flag specified. In *ex* text input  
95835 mode, each line shall be preceded by the line number it will have in the file.

95836 In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in  
95837 the format specified by the *ex print* command with the **#** flag specified. This line number shall  
95838 not be considered part of the line for the purposes of evaluating the current column; that is,  
95839 column position 1 shall be the first column position after the format specified by the **print**  
95840 command.

95841 **paragraphs, para**95842 [Default in the POSIX locale `IPLPPPQPP LIpplpipbp`]

95843 The **paragraphs** edit option shall define additional paragraph boundaries for the open and  
95844 visual mode commands. The **paragraphs** edit option can be set to a character string consisting of  
95845 zero or more character pairs. It shall be an error to set it to an odd number of characters.

95846 **prompt**95847 [Default *set*]

95848 If **prompt** is set, *ex* command mode input shall be prompted for with a <colon> (':'); when  
95849 unset, no prompt shall be written.

95850 **readonly**95851 [Default *see text*]

95852 If the **readonly** edit option is set, read-only mode shall be enabled (see [Write](#), on page 2871). The  
95853 **readonly** edit option shall be initialized to set if either of the following conditions are true:

- 95854 • The command-line option `-R` was specified.
- 95855 • Performing actions equivalent to the `access()` function called with the following arguments  
95856 indicates that the file lacks write permission:
  - 95857 1. The current pathname is used as the *path* argument.
  - 95858 2. The constant `W_OK` is used as the *amode* argument.

95859 The **readonly** edit option may be initialized to set for other, implementation-defined reasons.  
95860 The **readonly** edit option shall not be initialized to unset based on any special privileges of the  
95861 user or process. The **readonly** edit option shall be reinitialized each time that the contents of the  
95862 edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly  
95863 set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again  
95864 be reinitialized each time that the contents of the edit buffer are replaced.

95865 **redraw**95866 [Default *unset*]

95867 If **redraw** is set and the terminal is a type incapable of supporting open or visual modes, the  
95868 editor shall redraw the screen when necessary in order to update its contents. (Since this is  
95869 likely to require a large amount of output to the terminal, it is useful only at high transmission  
95870 speeds.)

95871 **remap**95872 [Default *set*]

95873 If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation  
95874 shall continue until a final product is obtained. If unset, only a one-step translation shall be  
95875 done.



95876 **report**

95877 [Default 5]

95878 The value of this **report** edit option specifies what number of lines being added, copied, deleted,  
 95879 or modified in the edit buffer will cause an informational message to be written to the user. The  
 95880 following conditions shall cause an informational message. The message shall contain the  
 95881 number of lines added, copied, deleted, or modified, but is otherwise unspecified.

- 95882 • An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the  
 95883 value of the **report** edit option number of lines, and which is not part of an *ex* **global** or *v*  
 95884 command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.
- 95885 • An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option  
 95886 plus 1 number of lines, and which is not part of an *ex* **global** or *v* command, or *ex* or *vi*  
 95887 buffer execution, shall cause an informational message to be written.
- 95888 • An *ex* **global**, *v*, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or  
 95889 deletes a total of at least the value of the **report** edit option number of lines, and which is  
 95890 not part of an *ex* **global** or *v* command, or *ex* or *vi* buffer execution, shall cause an  
 95891 informational message to be written. (For example, if 3 lines were added and 8 lines  
 95892 deleted during an *ex* **visual** command, 5 would be the number compared against the  
 95893 **report** edit option after the command completed.)

95894 **scroll, scr**

95895 [Default (number of lines in the display -1)/2]

95896 The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex*  
 95897 <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the  
 95898 initial number of lines to scroll when no previous <control>-D or <control>-U command has  
 95899 been executed.

95900 **sections**95901 [Default in the POSIX locale `NHSHH HUnhsh`]

95902 The **sections** edit option shall define additional section boundaries for the open and visual mode  
 95903 commands. The **sections** edit option can be set to a character string consisting of zero or more  
 95904 character pairs; it shall be an error to set it to an odd number of characters.

95905 **shell, sh**95906 [Default from the environment variable `SHELL`]

95907 The value of this option shall be a string. The default shall be taken from the `SHELL`  
 95908 environment variable. If the `SHELL` environment variable is null or empty, the *sh* (see *sh*) utility  
 95909 shall be the default.

**95910 shiftwidth, sw**

95911 [Default 8]

95912 The value of this option shall give the width in columns of an indentation level used during  
95913 autoindentation and by the shift commands (< and >).

**95914 showmatch, sm**

95915 [Default *unset*]

95916 The functionality described for the **showmatch** edit option need not be supported on block-  
95917 mode terminals or terminals with insufficient capabilities.

95918 If **showmatch** is set, in open or visual mode, when a ')' or '}' is typed, if the matching '(' or  
95919 '{' is currently visible on the display, the matching '(' or '{' shall be flagged moving the  
95920 cursor to its location for an unspecified amount of time.

**95921 showmode**

95922 [Default *unset*]

95923 If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be  
95924 displayed on the last line of the display. Command mode and text input mode shall be  
95925 differentiated; other unspecified modes and implementation-defined information may be  
95926 displayed.

**95927 slowopen**

95928 [Default *unset*]

95929 If **slowopen** is set during open and visual text input modes, the editor shall not update portions  
95930 of the display other than those display line columns that display the characters entered by the  
95931 user (see [Input Mode Commands in vi](#), on page 3561).

**95932 tabstop, ts**

95933 [Default 8]

95934 The value of this edit option shall specify the column boundary used by a <tab> in the display  
95935 (see [autoprint, ap](#) (on page 2877) and [Input Mode Commands in vi](#), on page 3561).

**95936 taglength, tl**

95937 [Default zero]

95938 The value of this edit option shall specify the maximum number of characters that are  
95939 considered significant in the user-specified tag name and in the tag name from the tags file. If  
95940 the value is zero, all characters in both tag names shall be significant.

- 95941           **tags**
- 95942           [Default *see text*]
- 95943           The value of this edit option shall be a string of <blank>-delimited pathnames of files used by  
95944           the **tag** command. The default value is unspecified.
- 95945           **term**
- 95946           [Default from the environment variable *TERM*]
- 95947           The value of this edit option shall be a string. The default shall be taken from the *TERM* variable  
95948           in the environment. If the *TERM* environment variable is empty or null, the default is  
95949           unspecified. The editor shall use the value of this edit option to determine the type of the display  
95950           device.
- 95951           The results are unspecified if the user changes the value of the term edit option after editor  
95952           initialization.
- 95953           **terse**
- 95954           [Default *unset*]
- 95955           If **terse** is set, error messages may be less verbose. However, except for this caveat, error  
95956           messages are unspecified. Furthermore, not all error messages need change for different settings  
95957           of this option.
- 95958           **warn**
- 95959           [Default *set*]
- 95960           If **warn** is set, and the contents of the edit buffer have been modified since they were last  
95961           completely written, the editor shall write a warning message before certain ! commands (see  
95962           [Escape](#), on page 2873).
- 95963           **window**
- 95964           [Default *see text*]
- 95965           A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in  
95966           visual mode, to specify the number of lines displayed when the screen is repainted.
- 95967           If the **-w** command-line option is not specified, the default value shall be set to the value of the  
95968           *LINES* environment variable. If the *LINES* environment variable is empty or null, the default  
95969           shall be the number of lines in the display minus 1.
- 95970           Setting the **window** edit option to zero or to a value greater than the number of lines in the  
95971           display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable)  
95972           shall cause the **window** edit option to be set to the number of lines in the display minus 1.
- 95973           The baud rate of the terminal line may change the default in an implementation-defined manner.

95974 **wrapmargin, wm**

95975 [Default 0]

95976 If the value of this edit option is zero, it shall have no effect.

95977 If not in the POSIX locale, the effect of this edit option is implementation-defined.

95978 Otherwise, it shall specify a number of columns from the ending margin of the terminal.

95979 During open and visual text input modes, for each character for which any part of the character  
 95980 is displayed in a column that is less than **wrapmargin** columns from the ending margin of the  
 95981 display line, the editor shall behave as follows:

- 95982 1. If the character triggering this event is a <blank>, it, and all immediately preceding  
 95983 <blank> characters on the current line entered during the execution of the current text  
 95984 input command, shall be discarded, and the editor shall behave as if the user had entered  
 95985 a single <newline> instead. In addition, if the next user-entered character is a <space>, it  
 95986 shall be discarded as well.
- 95987 2. Otherwise, if there are one or more <blank> characters on the current line immediately  
 95988 preceding the last group of inserted non-<blank> characters which was entered during  
 95989 the execution of the current text input command, the <blank> characters shall be replaced  
 95990 as if the user had entered a single <newline> instead.

95991 If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any  
 95992 <blank> characters at or after the cursor in the current line shall be discarded.

95993 The ending margin shall be determined by the system or overridden by the user, as described for  
 95994 **COLUMNS** in the ENVIRONMENT VARIABLES section and XBD [Chapter 8](#) (on page 167).

95995 **wrapsan, ws**95996 [Default *set*]

95997 If **wrapsan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, **N**, and **n**  
 95998 commands) shall wrap around the beginning or end of the edit buffer; when unset, searches  
 95999 shall stop at the beginning or end of the edit buffer.

96000 **writeany, wa**96001 [Default *unset*]

96002 If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be  
 96003 inhibited, as described in editor option **autowrite**.

96004 **EXIT STATUS**

96005 The following exit values shall be returned:

- 96006 0 Successful completion.
- 96007 >0 An error occurred.

96008 **CONSEQUENCES OF ERRORS**

96009 When any error is encountered and the standard input is not a terminal device file, in addition  
 96010 to the default requirements described in [Section 1.4](#) (on page 2462), *ex* shall neither write the file  
 96011 (if one has been opened) nor return to command or text input mode.

96012 Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP  
 96013 asynchronous event.

96014 Otherwise, when an error is encountered, the editor shall behave as specified in [Command Line](#)  
 96015 [Parsing in ex](#) (on page 2846).

## 96016 APPLICATION USAGE

96017 If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.

96018 The **next** command can accept more than one file, so usage such as:

```
96019 next `ls [abc]*`
```

96020 is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect  
 96021 only one file and unspecified results occur.

96022 Unlike the *system()* function, *ex* does not pass "--" between the "-c" argument and the  
 96023 command string, so that programs for which -c takes an option-argument can be used in the  
 96024 **shell** edit option. Users who want to use an **escape** command to execute a utility whose name  
 96025 starts with '-' or '+' need to provide a pathname for that utility that does not start with either  
 96026 of those characters, or precede the utility name with a <blank> character.

## 96027 EXAMPLES

96028 None.

## 96029 RATIONALE

96030 The *ex/vi* specification is based on the historical practice found in the 4 BSD and System V  
 96031 implementations of *ex* and *vi*.

96032 A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and  
 96033 rejected for inclusion. Neither option provided the level of security that users might expect.

96034 It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to  
 96035 implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor  
 96036 addressing; thus, it is not a mandatory requirement that such features should work on all  
 96037 terminals. It is the intention, however, that an *ex* implementation should provide the full set of  
 96038 capabilities on all terminals capable of supporting them.

## 96039 Options

96040 The -c replacement for +*command* was inspired by the -e option of *sed*. Historically, all such  
 96041 commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This  
 96042 meant, for example, that "+/pattern" would fail unless the **wrapsan** option was set.  
 96043 POSIX.1-2024 requires conformance to historical practice. The +*command* option is no longer  
 96044 specified by POSIX.1-2024 but may be present in some implementations. Historically, some  
 96045 implementations restricted the *ex* commands that could be listed as part of the command line  
 96046 arguments. For consistency, POSIX.1-2024 does not permit these restrictions.

96047 In historical implementations of the editor, the -R option (and the **readonly** edit option) only  
 96048 prevented overwriting of files; appending to files was still permitted, mapping loosely into the  
 96049 *bash* **noclobber** variable. Some implementations, however, have not followed this semantic, and  
 96050 **readonly** does not permit appending either. POSIX.1-2024 follows the latter practice, believing  
 96051 that it is a more obvious and intuitive meaning of **readonly**.

96052 The -s option suppresses all interactive user feedback and is useful for editing scripts in batch  
 96053 jobs. The list of specific effects is historical practice. The terminal type "incapable of supporting  
 96054 open and visual modes" has historically been named "dumb".

96055 The -t option was required because the *ctags* utility appears in POSIX.1-2024 and the option is  
 96056 available in all historical implementations of *ex*.

96057 Historically, the *ex* and *vi* utilities accepted a -x option, which did encryption based on the

96058 algorithm found in the historical *crypt* utility. The `-x` option for encryption, and the associated  
 96059 *crypt* utility, were omitted because the algorithm used was not specifiable and the export control  
 96060 laws of some nations make it difficult to export cryptographic technology. In addition, it did not  
 96061 historically provide the level of security that users might expect.

### 96062 **Standard Input**

96063 An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file  
 96064 character, `<control>-D`, is historically an *ex* command.

96065 There was no maximum line length in historical implementations of *ex*. Specifically, as it was  
 96066 parsed in chunks, the addresses had a different maximum length than the filenames. Further, the  
 96067 maximum line buffer size was declared as `BUFSIZ`, which was different lengths on different  
 96068 systems. This version selected the value of `{LINE_MAX}` to impose a reasonable restriction on  
 96069 portable usage of *ex* and to aid test suite writers in their development of realistic tests that  
 96070 exercise this limit.

### 96071 **Input Files**

96072 It was an explicit decision by the standard developers that a `<newline>` be added to any file  
 96073 lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order to make  
 96074 text files lacking a trailing `<newline>` more portable. It is recognized that this will require a user-  
 96075 specified option or extension for implementations that permit *ex* and *vi* to edit files of type other  
 96076 than text if such files are not otherwise identified by the system. It was agreed that the ability to  
 96077 edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an  
 96078 *ex* or *vi* implementation be required to handle files other than text files.

96079 The paragraph in the INPUT FILES section, “By default, ...”, is intended to close a long-  
 96080 standing security problem in *ex* and *vi*; that of the “`modeline`” or “`modelines`” edit option. This  
 96081 feature allows any line in the first or last five lines of the file containing the strings “`ex:`” or  
 96082 “`vi:`” (and, apparently, “`ei:`” or “`vx:`”) to be a line containing editor commands, and *ex*  
 96083 interprets all the text up to the next `:` or `<newline>` as a command. Consider the  
 96084 consequences, for example, of an unsuspecting user using *ex* or *vi* as the editor when replying to  
 96085 a mail message in which a line such as:

```
96086 ex:! rm -rf :
```

96087 appeared in the signature lines. The standard developers believed strongly that an editor should  
 96088 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from  
 96089 their implementations of *ex* and *vi*.

### 96090 **Asynchronous Events**

96091 The intention of the phrase “complete write” is that the entire edit buffer be written to stable  
 96092 storage. The note regarding temporary files is intended for implementations that use temporary  
 96093 files to back edit buffers unnamed by the user.

96094 Historically, `SIGQUIT` was ignored by *ex*, but was the equivalent of the `Q` command in visual  
 96095 mode; that is, it exited visual mode and entered *ex* mode. POSIX.1-2024 permits, but does not  
 96096 require, this behavior. Historically, `SIGINT` was often used by *vi* users to terminate text input  
 96097 mode (`<control>-C` is often easier to enter than `<ESC>`). Some implementations of *vi* alerted the  
 96098 terminal on this event, and some did not. POSIX.1-2024 requires that `SIGINT` behave identically  
 96099 to `<ESC>`, and that the terminal not be alerted.

96100 Historically, suspending the *ex* editor during text input mode was similar to `SIGINT`, as  
 96101 completed lines were retained, but any partial line discarded, and the editor returned to

96102 command mode. POSIX.1-2024 is silent on this issue; implementations are encouraged to follow  
96103 historical practice, where possible.

96104 Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore  
96105 impossible to suspend the editor in visual text input mode. There are two major reasons for this.  
96106 The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where  
96107 the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if  
96108 SIGTSTP was delivered to the process group in the default manner. The second was that most  
96109 implementations of the UNIX *curses* package did not handle SIGTSTP safely, and the receipt of  
96110 SIGTSTP at the wrong time would cause them to crash. POSIX.1-2024 is silent on this issue;  
96111 implementations are encouraged to treat suspension as an asynchronous event if possible.

96112 Historically, modifications to the edit buffer made before SIGINT interrupted an operation were  
96113 retained; that is, anywhere from zero to all of the lines to be modified might have been modified  
96114 by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT.  
96115 POSIX.1-2024 permits this behavior, noting that the **undo** command is required to be able to  
96116 undo these partially completed commands.

96117 The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is  
96118 unspecified because some implementations attempt to save the edit buffer in a useful state when  
96119 other signals are received.

## 96120 Standard Error

96121 For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to  
96122 invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination  
96123 condition. Diagnostic messages should not be confused with the error messages generated by  
96124 inappropriate or illegal user commands.

## 96125 Initialization in *ex* and *vi*

96126 If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the  
96127 alternate and current pathnames will be set. Informally, they are set as follows:

- 96128 1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the  
96129 current pathname will be set to the filename argument (the first filename argument in the  
96130 case of the **next** command) and the alternate pathname will be set to the previous current  
96131 pathname, if there was one.
- 96132 2. In the case of the file read/write forms of the **read** and **write** commands, if there is no  
96133 current pathname, the current pathname will be set to the filename argument.
- 96134 3. Otherwise, the alternate pathname will be set to the filename argument.

96135 For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there  
96136 was a previous current pathname, the alternate pathname. The commands **:write**, **!command**,  
96137 and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for  
96138 some reason, the alternate pathname would be set. The **read** and **write** commands set the  
96139 alternate pathname to their *file* argument, unless the current pathname is not set, in which case  
96140 they set the current pathname to their *file* arguments. The alternate pathname was not  
96141 historically set by the **:source** command. POSIX.1-2024 requires conformance to historical  
96142 practice. Implementations adding commands that take filenames as arguments are encouraged  
96143 to set the alternate pathname as described here.

96144 Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed  
96145 in the *\$HOME* directory. POSIX.1-2024 prohibits this behavior.

96146 Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local *.exrc* files if they were owned by the  
96147 real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was  
96148 a security problem because it is possible to put normal UNIX system commands inside a *.exrc*  
96149 file. POSIX.1-2024 does not specify the **sourceany** option, and historical implementations are  
96150 encouraged to delete it.

96151 The *.exrc* files must be owned by the real ID of the user, and not writable by anyone other than  
96152 the owner. The appropriate privileges exception is intended to permit users to acquire special  
96153 privileges, but continue to use the *.exrc* files in their home directories.

96154 System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is  
96155 that local *.exrc* files are read-only if the **exrc** option is set. The default for the **exrc** option was off,  
96156 so by default, local *.exrc* files were not read. The problem this was intended to solve was that  
96157 System V permitted users to give away files, so there is no possible ownership or writeability  
96158 test to ensure that the file is safe. This is still a security problem on systems where users can give  
96159 away files, but there is nothing additional that POSIX.1-2024 can do. The implementation-  
96160 defined exception is intended to permit groups to have local *.exrc* files that are shared by users,  
96161 by creating pseudo-users to own the shared files.

96162 POSIX.1-2024 does not mention system-wide *ex* and *vi* start-up files. While they exist in several  
96163 implementations of *ex* and *vi*, they are not present in any implementations considered historical  
96164 practice by POSIX.1-2024. Implementations that have such files should use them only if they are  
96165 owned by the real user ID or an appropriate user (for example, root on UNIX systems) and if  
96166 they are not writable by any user other than their owner. System-wide start-up files should be  
96167 read before the *EXINIT* variable, *\$HOME/.exrc*, or local *.exrc* files are evaluated.

96168 Historically, any *ex* command could be entered in the *EXINIT* variable or the *.exrc* file, although  
96169 ones requiring that the edit buffer already contain lines of text generally caused historical  
96170 implementations of the editor to drop **core**. POSIX.1-2024 requires that any *ex* command be  
96171 permitted in the *EXINIT* variable and *.exrc* files, for simplicity of specification and consistency,  
96172 although many of them will obviously fail under many circumstances.

96173 The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with  
96174 regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded  
96175 during the initialization phase not be lost; that is, loading the edit buffer should fail if the *.exrc*  
96176 file read in the contents of a file and did not subsequently write the edit buffer. An additional  
96177 intent of this phrase is to specify that the initial current line and column is set as specified for the  
96178 individual *ex* commands.

96179 Historically, the **-t** option behaved as if the tag search were a *+command*; that is, it was executed  
96180 from the last line of the file specified by the tag. This resulted in the search failing if the pattern  
96181 was a forward search pattern and the **wrapsan** edit option was not set. POSIX.1-2024 does not  
96182 permit this behavior, requiring that the search for the tag pattern be performed on the entire file,  
96183 and, if not found, that the current line be set to a more reasonable location in the file.

96184 Historically, the empty edit buffer presented for editing when a file was not specified by the user  
96185 was unnamed. This is permitted by POSIX.1-2024; however, implementations are encouraged to  
96186 provide users a temporary filename for this buffer because it permits them the use of *ex*  
96187 commands that use the current pathname during temporary edit sessions.

96188 Historically, the file specified using the **-t** option was not part of the current argument list. This  
96189 practice is permitted by POSIX.1-2024; however, implementations are encouraged to include its  
96190 name in the current argument list for consistency.

96191 Historically, the **-c** command was generally not executed until a file that already exists was  
96192 edited. POSIX.1-2024 requires conformance to this historical practice. Commands that could



96193 cause the `-c` command to be executed include the *ex* commands **edit**, **next**, **recover**, **rewind**, and  
 96194 **tag**, and the *vi* commands `<control>-^` and `<control>-]`. Historically, reading a file into an edit  
 96195 buffer did not cause the `-c` command to be executed (even though it might set the current  
 96196 pathname) with the exception that it did cause the `-c` command to be executed if: the editor was  
 96197 in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read  
 96198 commands had yet been attempted. For consistency and simplicity of specification,  
 96199 POSIX.1-2024 does not permit this behavior.

96200 Historically, the `-r` option was the same as a normal edit session if there was no recovery  
 96201 information available for the file. This allowed users to enter:

```
96202 vi -r *.c
```

96203 and recover whatever files were recoverable. In some implementations, recovery was attempted  
 96204 only on the first file named, and the file was not entered into the argument list; in others,  
 96205 recovery was attempted for each file named. In addition, some historical implementations  
 96206 ignored `-r` if `-t` was specified or did not support command line *file* arguments with the `-t` option.  
 96207 For consistency and simplicity of specification, POSIX.1-2024 disallows these special cases, and  
 96208 requires that recovery be attempted the first time each file is edited.

96209 Historically, *vi* initialized the ``` and `'` marks, but *ex* did not. This meant that if the first command  
 96210 in *ex* mode was **visual** or if an *ex* command was executed first (for example, *vi +10 file*), *vi* was  
 96211 entered without the marks being initialized. Because the standard developers believed the marks  
 96212 to be generally useful, and for consistency and simplicity of specification, POSIX.1-2024 requires  
 96213 that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is  
 96214 not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however,  
 96215 it has always been possible to set (and use) marks in empty edit buffers in open and visual mode  
 96216 edit sessions.

## 96217 Addressing

96218 Historically, *ex* and *vi* accepted the additional addressing forms `'\/'` and `'\?'`. They were  
 96219 equivalent to `"//"` and `"??"`, respectively. They are not required by POSIX.1-2024, mostly  
 96220 because nobody can remember whether they ever did anything different historically.

96221 Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the `%`  
 96222 address in empty files for others. For consistency, POSIX.1-2024 requires support for the former  
 96223 in the few commands where it makes sense, and disallows it otherwise. In addition, because  
 96224 POSIX.1-2024 requires that `%` be logically equivalent to `"1,$"`, it is also supported where it  
 96225 makes sense and disallowed otherwise.

96226 Historically, the `%` address could not be followed by further addresses. For consistency and  
 96227 simplicity of specification, POSIX.1-2024 requires that additional addresses be supported.

96228 All of the following are valid *addresses*:

```
96229 +++           Three lines after the current line.
96230 /re/-        One line before the next occurrence of re.
96231 -2           Two lines before the current line.
96232 3 ---- 2    Line one (note intermediate negative address).
96233 1 2 3       Line six.
```

96234 Any number of addresses can be provided to commands taking addresses; for example,  
 96235 `"1,2,3,4,5p"` prints lines 4 and 5, because two is the greatest valid number of addresses  
 96236 accepted by the **print** command. This, in combination with the `<semicolon>` delimiter, permits

96237 users to create commands based on ordered patterns in the file. For example, the command  
 96238 **3;/foo/;+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next  
 96239 two lines. Note that the address **3;** must be evaluated before being discarded because the search  
 96240 origin for the **/foo/** command depends on this.

96241 Historically, values could be added to addresses by including them after one or more <blank>  
 96242 characters; for example, **3 – 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as  
 96243 **/foo/+5**. However, only absolute values could be added; for example, **5 /foo/** was an error.  
 96244 POSIX.1-2024 requires conformance to historical practice. Address offsets are separately  
 96245 specified from addresses because they could historically be provided to visual mode search  
 96246 commands.

96247 Historically, any missing addresses defaulted to the current line. This was true for leading and  
 96248 trailing <comma>-delimited addresses, and for trailing <semicolon>-delimited addresses. For  
 96249 consistency, POSIX.1-2024 requires it for leading <semicolon> addresses as well.

96250 Historically, *ex* and *vi* accepted the '^' character as both an address and as a flag offset for  
 96251 commands. In both cases it was identical to the '-' character. POSIX.1-2024 does not require or  
 96252 prohibit this behavior.

96253 Historically, the enhancements to basic regular expressions could be used in addressing; for  
 96254 example, '~', '\<', and '\>'. POSIX.1-2024 requires conformance to historical practice; that  
 96255 is, that regular expression usage be consistent, and that regular expression enhancements be  
 96256 supported wherever regular expressions are used.

### 96257 Command Line Parsing in ex

96258 Historical *ex* command parsing was even more complex than that described here. POSIX.1-2024  
 96259 requires the subset of the command parsing that the standard developers believed was  
 96260 documented and that users could reasonably be expected to use in a portable fashion, and that  
 96261 was historically consistent between implementations. (The discarded functionality is obscure, at  
 96262 best.) Historical implementations will require changes in order to comply with POSIX.1-2024;  
 96263 however, users are not expected to notice any of these changes. Most of the complexity in *ex*  
 96264 parsing is to handle three special termination cases:

- 96265 1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by  
 96266 <newline> characters (they can contain <vertical-line> characters that are usually shell  
 96267 pipes).
- 96268 2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands,  
 96269 optionally containing <vertical-line> characters, as their first arguments.
- 96270 3. The **s** command takes a regular expression as its first argument, and uses the delimiting  
 96271 characters to delimit the command.

96272 Historically, <vertical-line> characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and  
 96273 **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the  
 96274 command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did  
 96275 not delimit the command at all. For example, the following commands are all valid:

```
96276 :edit +25 | s/abc/ABC/ file.c
96277 :s/ | /PIPE/
96278 :read !spell % | columnate
96279 :global/pattern/p | l
96280 :s/a/b/ | s/c/d | set
```

96281 Historically, empty or <blank> filled lines in **.exrc** files and **sourced** files (as well as *EXINIT*

96282 variables and *ex* command scripts) were treated as default commands; that is, **print** commands.  
 96283 POSIX.1-2024 specifically requires that they be ignored when encountered in **.exrc** and **sourced**  
 96284 files to eliminate a common source of new user error.

96285 Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were  
 96286 handled oddly when executed from *ex* mode. For example, the command `|||<carriage-return>`,  
 96287 when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command `|`  
 96288 would only display the line after the next line, instead of the next two lines. The former worked  
 96289 more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. POSIX.1-2024  
 96290 requires the *vi* behavior; that is, a single default command and line number increment for each  
 96291 command separator, and trailing <newline> characters after <vertical-line> separators are  
 96292 discarded.

96293 Historically, *ex* permitted a single extra <colon> as a leading command character; for example,  
 96294 **:g/pattern/:p** was a valid command. POSIX.1-2024 generalizes this to require that any number of  
 96295 leading <colon> characters be stripped.

96296 Historically, any prefix of the **delete** command could be followed without intervening <blank>  
 96297 characters by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*.  
 96298 POSIX.1-2024 requires conformance to historical practice.

96299 Historically, the **k** command could be followed by the mark name without intervening <blank>  
 96300 characters. POSIX.1-2024 requires conformance to historical practice.

96301 Historically, the **s** command could be immediately followed by flag and option characters; for  
 96302 example, **s/e/E/|s|sgc3p** was a valid command. However, flag characters could not stand alone;  
 96303 for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would  
 96304 succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the  
 96305 command.) Another issue was that option characters had to precede flag characters even when  
 96306 the command was fully specified; for example, the command **s/e/E/pg** would fail, while the  
 96307 command **s/e/E/gp** would succeed. POSIX.1-2024 requires conformance to historical practice.

96308 Historically, the first command name that had a prefix matching the input from the user was the  
 96309 executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command.  
 96310 Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**.  
 96311 POSIX.1-2024 requires conformance to historical practice. The restriction on command search  
 96312 order for implementations with extensions is to avoid the addition of commands such that the  
 96313 historical prefixes would fail to work portably.

96314 Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands,  
 96315 separated by <vertical-line> characters, that entered or exited visual mode or the editor. Because  
 96316 implementations of *vi* exist that do not exhibit this failure mode, POSIX.1-2024 does not permit  
 96317 it.

96318 The requirement that alphabetic command names consist of all following alphabetic characters  
 96319 up to the next non-alphabetic character means that alphabetic command names must be  
 96320 separated from their arguments by one or more non-alphabetic characters, normally a <blank>  
 96321 or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

96322 Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*,  
 96323 <newline>, <carriage-return>) erased any prompting character and displayed the next lines  
 96324 without scrolling the terminal; that is, immediately below any previously displayed lines. This  
 96325 provided a cleaner presentation of the lines in the file for the user. POSIX.1-2024 does not require  
 96326 this behavior because it may be impossible in some situations; however, implementations are  
 96327 strongly encouraged to provide this semantic if possible.

96328 Historically, it was possible to change files in the middle of a command, and have the rest of the

96329 command executed in the new file; for example:

```
96330 :edit +25 file.c | s/abc/ABC/ | 1
```

96331 was a valid command, and the substitution was attempted in the newly edited file.  
 96332 POSIX.1-2024 requires conformance to historical practice. The following commands are  
 96333 examples that exercise the *ex* parser:

```
96334 echo 'foo | bar' > file1; echo 'foo/bar' > file2;
```

```
96335 vi
```

```
96336 :edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\/SLASH/ | wq
```

96337 Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or **\***  
 96338 commands changing edit buffers during execution of their associated commands. Because this  
 96339 would almost invariably result in catastrophic failure of the editor, and implementations exist  
 96340 that do exhibit these problems, POSIX.1-2024 requires that changing the edit buffer during a  
 96341 **global** or **v** command, or during a **@** or **\*** command for which there will be more than a single  
 96342 execution, be an error. Implementations supporting multiple edit buffers simultaneously are  
 96343 strongly encouraged to apply the same semantics to switching between buffers as well.

96344 The *ex* command quoting required by POSIX.1-2024 is a superset of the quoting in historical  
 96345 implementations of the editor. For example, it was not historically possible to escape a <blank>  
 96346 in a filename; for example, **:edit foo\\\ bar** would report that too many filenames had been  
 96347 entered for the edit command, and there was no method of escaping a <blank> in the first  
 96348 argument of an **edit**, **ex**, **next**, or **visual** command at all. POSIX.1-2024 extends historical  
 96349 practice, requiring that quoting behavior be made consistent across all *ex* commands, except for  
 96350 the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V  
 96351 instead of <backslash> characters for quoting. For those four commands, POSIX.1-2024 requires  
 96352 conformance to historical practice.

96353 Backslash quoting in *ex* is non-intuitive. <backslash>-escapes are ignored unless they escape a  
 96354 special character; for example, when performing *file* argument expansion, the string "\\%" is  
 96355 equivalent to '\%', not "\<current pathname>". This can be confusing for users because  
 96356 <backslash> is usually one of the characters that causes shell expansion to be performed, and  
 96357 therefore shell quoting rules must be taken into consideration. Generally, quoting characters are  
 96358 only considered if they escape a special character, and a quoting character must be provided for  
 96359 each layer of parsing for which the character is special. As another example, only a single  
 96360 <backslash> is necessary for the '\1' sequence in substitute replacement patterns, because the  
 96361 character '1' is not special to any parsing layer above it.

96362 <control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands  
 96363 where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character  
 96364 may be escaped by a <control>-V whether it would have a special meaning or not. POSIX.1-2024  
 96365 requires conformance to historical practice.

96366 Historical implementations of the editor did not require delimiters within character classes to be  
 96367 escaped; for example, the command **:s/[/]//** on the string "xxx/yyy" would delete the '/' from  
 96368 the string. POSIX.1-2024 disallows this historical practice for consistency and because it places a  
 96369 large burden on implementations by requiring that knowledge of regular expressions be built  
 96370 into the editor parser.

96371 Historically, quoting <newline> characters in *ex* commands was handled inconsistently. In most  
 96372 cases, the <newline> character always terminated the command, regardless of any preceding  
 96373 escape character, because <backslash> characters did not escape <newline> characters for most  
 96374 *ex* commands. However, some *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted  
 96375 <newline> characters to be escaped (although in the case of **map** and **abbreviation**, <control>-V

96376 characters escaped them instead of <backslash> characters). This was true in not only the  
96377 command line, but also **.exrc** and **sourced** files. For example, the command:

```
96378 map = foo<control-V><newline>bar
```

96379 would succeed, although it was sometimes difficult to get the <control>-V and the inserted  
96380 <newline> passed to the *ex* parser. For consistency and simplicity of specification, POSIX.1-2024  
96381 requires that it be possible to escape <newline> characters in *ex* commands at all times, using  
96382 <backslash> characters for most *ex* commands, and using <control>-V characters for the **map**  
96383 and **abbreviation** commands. For example, the command **print<newline>list** is required to be  
96384 parsed as the single command **print<newline>list**. While this differs from historical practice,  
96385 POSIX.1-2024 developers believed it unlikely that any script or user depended on the historical  
96386 behavior.

96387 Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c**  
96388 commands to be discarded. POSIX.1-2024 disallows this for consistency with mapped keys, the  
96389 **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc** files.

### 96390 Input Editing in *ex*

96391 One of the common uses of the historical *ex* editor is over slow network connections. Editors that  
96392 run in canonical mode can require far less traffic to and from, and far less processing on, the host  
96393 machine, as well as more easily supporting block-mode terminals. For these reasons,  
96394 POSIX.1-2024 requires that *ex* be implemented using canonical mode input processing, as was  
96395 done historically.

96396 POSIX.1-2024 does not require the historical 4 BSD input editing characters “word erase” or  
96397 “literal next”. For this reason, it is unspecified how they are handled by *ex*, although they must  
96398 have the required effect. Implementations that resolve them after the line has been ended using a  
96399 <newline> or <control>-M character, and implementations that rely on the underlying system  
96400 terminal support for this processing, are both conforming. Implementations are strongly urged  
96401 to use the underlying system functionality, if at all possible, for compatibility with other system  
96402 text input interfaces.

96403 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor  
96404 moved to display the new end of the **autoindent** characters, but did not move the cursor to a  
96405 new line, nor did it erase the <control>-D character from the line. POSIX.1-2024 does not specify  
96406 that the cursor remain on the same line or that the rest of the line is erased; however,  
96407 implementations are strongly encouraged to provide the best possible user interface; that is, the  
96408 cursor should remain on the same line, and any <control>-D character on the line should be  
96409 erased.

96410 POSIX.1-2024 does not require the historical 4 BSD input editing character “reprint”,  
96411 traditionally <control>-R, which redisplayed the current input from the user. For this reason,  
96412 and because the functionality cannot be implemented after the line has been terminated by the  
96413 user, POSIX.1-2024 makes no requirements about this functionality. Implementations are  
96414 strongly urged to make this historical functionality available, if possible.

96415 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.  
96416 POSIX.1-2024 requires conformance to historical practice to avoid breaking historical *ex* scripts  
96417 and **.exrc** files.

96418 **eof**

96419 Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left  
 96420 unspecified so that implementations can conform in the presence of systems that do not support  
 96421 this functionality. Implementations are encouraged to modify the line and redisplay it  
 96422 immediately, if possible.

96423 The specification of the handling of the *eof* character differs from historical practice only in that  
 96424 *eof* characters are not discarded if they follow normal characters in the text input. Historically,  
 96425 they were always discarded.

### 96426 **Command Descriptions in ex**

96427 Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and  
 96428 **~**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit  
 96429 addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or  
 96430 command execution in an empty file, make sense only for commands that add new text to the  
 96431 edit buffer or write commands (because users may wish to write empty files). POSIX.1-2024  
 96432 requires this behavior for such commands and disallows it otherwise, for consistency and  
 96433 simplicity of specification.

96434 A count to an *ex* command has been historically corrected to be no greater than the last line in a  
 96435 file; for example, in a five-line file, the command **1,6print** would fail, but the command  
 96436 **1print300** would succeed. POSIX.1-2024 requires conformance to historical practice.

96437 Historically, the use of flags in *ex* commands could be obscure. General historical practice was as  
 96438 described by POSIX.1-2024, but there were some special cases. For instance, the **list**, **number**,  
 96439 and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line  
 96440 3, and 3 would be the current line after the execution of the command. The **open** and **visual**  
 96441 commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the  
 96442 **open** and **visual** commands interacted badly with the **list** edit option, and setting and then  
 96443 unsetting it during the open/visual session would cause *vi* to stop displaying lines in the  
 96444 specified format. For consistency and simplicity of specification, POSIX.1-2024 does not permit  
 96445 any of these exceptions to the general rule.

96446 POSIX.1-2024 uses the word *copy* in several places when discussing buffers. This is not intended  
 96447 to imply implementation.

96448 Historically, *ex* users could not specify numeric buffers because of the ambiguity this would  
 96449 cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a  
 96450 *count*. POSIX.1-2024 requires conformance to historical practice by default, but does not  
 96451 preclude extensions.

96452 Historically, the contents of the unnamed buffer were frequently discarded after commands that  
 96453 did not explicitly affect it; for example, when using the **edit** command to switch files. For  
 96454 consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

96455 The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting  
 96456 lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user  
 96457 switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric  
 96458 buffers would not have changed. POSIX.1-2024 requires conformance to historical practice.  
 96459 Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a  
 96460 single location in POSIX.1-2024.

96461 The metacharacters that trigger shell expansion in *file* arguments match historical practice, as  
 96462 does the method for doing shell expansion. Implementations wishing to provide users with the  
 96463 flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit

96464 option.

96465 Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to  
 96466 do so; for example, `!date > /dev/null` does not require a screen refresh because the output of  
 96467 the UNIX *date* command requires only a single line of the screen. POSIX.1-2024 requires that the  
 96468 screen be refreshed if it has been overwritten, but makes no requirements as to how an  
 96469 implementation should make that determination. Implementations may prompt and refresh the  
 96470 screen regardless.

### 96471 Abbreviate

96472 Historical practice was that characters that were entered as part of an abbreviation replacement  
 96473 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,  
 96474 and so on; that is, they were logically pushed onto the terminal input queue, and were not a  
 96475 simple replacement. POSIX.1-2024 requires conformance to historical practice. Historical  
 96476 practice was that whenever a non-word character (that had not been escaped by a `<control>-V`)  
 96477 was entered after a word character, *vi* would check for abbreviations. The check was based on  
 96478 the type of the character entered before the word character of the word/non-word pair that  
 96479 triggered the check. The word character of the word/non-word pair that triggered the check and  
 96480 all characters entered before the trigger pair that were of that type were included in the check,  
 96481 with the exception of `<blank>` characters, which always delimited the abbreviation.

96482 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can  
 96483 be no transitions from word to non-word characters (or *vice versa*) other than between the last  
 96484 and next-to-last characters in the *lhs*, and there can be no `<blank>` characters in the *lhs*. In  
 96485 addition, because of the historical quoting rules, it was impossible to enter a literal `<control>-V`  
 96486 in the *lhs*. POSIX.1-2024 requires conformance to historical practice. Historical implementations  
 96487 did not inform users when abbreviations that could never be used were entered;  
 96488 implementations are strongly encouraged to do so.

96489 For example, the following abbreviations will work:

```
96490 :ab (p REPLACE
96491 :ab p REPLACE
96492 :ab ( (p REPLACE
```

96493 The following abbreviations will not work:

```
96494 :ab ( REPLACE
96495 :ab (pp REPLACE
```

96496 Historical practice is that words on the *vi* colon command line were subject to abbreviation  
 96497 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**  
 96498 command. Because there are implementations that do not do abbreviation expansion for the first  
 96499 argument to those commands, this is permitted, but not required, by POSIX.1-2024. However,  
 96500 the following sequence:

```
96501 :ab foo bar
96502 :ab foo baz
```

96503 resulted in the addition of an abbreviation of "baz" for the string "bar" in historical *ex/vi*, and  
 96504 the sequence:

```
96505 :ab foo1 bar
96506 :ab foo2 bar
96507 :unabbreviate foo2
```

96508 deleted the abbreviation "foo1", not "foo2". These behaviors are not permitted by

96509 POSIX.1-2024 because they clearly violate the expectations of the user.

96510 It was historical practice that <control>-V, not <backslash>, characters be interpreted as escaping  
 96511 subsequent characters in the **abbreviate** command. POSIX.1-2024 requires conformance to  
 96512 historical practice; however, it should be noted that an abbreviation containing a <blank> will  
 96513 never work.

### 96514 **Append**

96515 Historically, any text following a <vertical-line> command separator after an **append**, **change**, or  
 96516 **insert** command became part of the insert text. For example, in the command:

```
96517 :g/pattern/append|stuff1
```

96518 a line containing the text "stuff1" would be appended to each line matching pattern. It was  
 96519 also historically valid to enter:

```
96520 :append|stuff1
96521 stuff2
96522 .
```

96523 and the text on the *ex* command line would be appended along with the text inserted after it.  
 96524 There was an historical bug, however, that the user had to enter two terminating lines (the ' . '  
 96525 lines) to terminate text input mode in this case. POSIX.1-2024 requires conformance to historical  
 96526 practice, but disallows the historical need for multiple terminating lines.

### 96527 **Change**

96528 See the RATIONALE for the **append** command. Historical practice for cursor positioning after  
 96529 the change command when no text is input, is as described in POSIX.1-2024. However, one  
 96530 System V implementation is known to have been modified such that the cursor is positioned on  
 96531 the first address specified, and not on the line before the first address. POSIX.1-2024 disallows  
 96532 this modification for consistency.

96533 Historically, the **change** command did not support buffer arguments, although some  
 96534 implementations allow the specification of an optional buffer. This behavior is neither required  
 96535 nor disallowed by POSIX.1-2024.

### 96536 **Change Directory**

96537 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as  
 96538 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have  
 96539 ' . ' or " . . " as their first component. Elements in the **cdpath** edit option are <colon>-separated.  
 96540 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment  
 96541 variable. This feature was not included in POSIX.1-2024 because it does not exist in any of the  
 96542 implementations considered historical practice.

### 96543 **Copy**

96544 Historical implementations of *ex* permitted copies to lines inside of the specified range; for  
 96545 example, **:2,5copy3** was a valid command. POSIX.1-2024 requires conformance to historical  
 96546 practice.



96547

**Delete**

96548

POSIX.1-2024 requires support for the historical parsing of a **delete** command followed by flags, without any intervening <blank> characters. For example:

96549

96550

**1dp** Deletes the first line and prints the line that was second.

96551

**1delep** As for **1dp**.

96552

**1d** Deletes the first line, saving it in buffer *p*.

96553

**1d p1l** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was second.

96554

96555

**Edit**

96556

Historically, any *ex* command could be entered as a *+command* argument to the **edit** command, although some (for example, **insert** and **append**) were known to confuse historical implementations. For consistency and simplicity of specification, POSIX.1-2024 requires that any command be supported as an argument to the **edit** command.

96557

96558

96559

96560

Historically, the command argument was executed with the current line set to the last line of the file, regardless of whether the **edit** command was executed from visual mode or not. POSIX.1-2024 requires conformance to historical practice.

96561

96562

96563

Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first <blank>, and there was no way to quote them. For consistency, POSIX.1-2024 requires that the usual *ex* backslash quoting be provided.

96564

96565

96566

Historically, specifying the *+command* argument to the edit command required a filename to be specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of specification, POSIX.1-2024 does not permit this usage to fail for that reason.

96567

96568

96569

Historically, only the cursor position of the last file edited was remembered by the editor. POSIX.1-2024 requires that this be supported; however, implementations are permitted to remember and restore the cursor position for any file previously edited.

96570

96571

96572

**File**

96573

Historical versions of the *ex* editor **file** command displayed a current line and number of lines in the edit buffer of 0 when the file was empty, while the *vi* <control>-G command displayed a current line and number of lines in the edit buffer of 1 in the same situation. POSIX.1-2024 does not permit this discrepancy, instead requiring that a message be displayed indicating that the file is empty.

96574

96575

96576

96577

96578

**Global**

96579

The two-pass operation of the **global** and **v** commands is not intended to imply implementation, only the required result of the operation.

96580

96581

The current line and column are set as specified for the individual *ex* commands. This requirement is cumulative; that is, the current line and column must track across all the commands executed by the **global** or **v** commands.

96582

96583

96584 **Insert**96585 See the RATIONALE for the **append** command.96586 Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer  
96587 was empty. POSIX.1-2024 requires that this command behave consistently with the **append**  
96588 command.96589 **Join**96590 The action of the **join** command in relation to the special characters is only defined for the  
96591 POSIX locale because the correct amount of white space after a period varies; in Japanese none is  
96592 required, in French only a single space, and so on.96593 **List**96594 The historical output of the **list** command was potentially ambiguous. The standard developers  
96595 believed correcting this to be more important than adhering to historical practice, and  
96596 POSIX.1-2024 requires unambiguous output.96597 **Map**96598 Historically, command mode maps only applied to command names; for example, if the  
96599 character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y'  
96600 character. POSIX.1-2024 requires this behavior. Historically, entering <control>-V as the first  
96601 character of a *vi* command was an error. Several implementations have extended the semantics  
96602 of *vi* such that <control>-V means that the subsequent command character is not mapped. This  
96603 is permitted, but not required, by POSIX.1-2024. Regardless, using <control>-V to escape the  
96604 second or later character in a sequence of characters that might match a **map** command, or any  
96605 character in text input mode, is historical practice, and stops the entered keys from matching a  
96606 map. POSIX.1-2024 requires conformance to historical practice.96607 Historically implementations permitted digits to be used as a **map** command *lhs*, but then ignored  
96608 the map. POSIX.1-2024 requires that the mapped digits not be ignored.96609 The historical implementation of the **map** command did not permit **map** commands that were  
96610 more than a single character in length if the first character was printable. This behavior is  
96611 permitted, but not required, by POSIX.1-2024.96612 Historically, mapped characters were remapped unless the **remap** edit option was not set, or the  
96613 prefix of the mapped characters matched the mapping characters; for example, in the **map**:96614 

```
:map ab abcd
```

96615 the characters "ab" were used as is and were not remapped, but the characters "cd" were  
96616 mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms.  
96617 POSIX.1-2024 requires conformance to historical practice, and that such loops be interruptible.96618 Text input maps had the same problems with expanding the *lhs* for the **ex map!** and **unmap!**  
96619 command as did the **ex abbreviate** and **unabbreviate** commands. See the RATIONALE for the **ex**  
96620 **abbreviate** command. POSIX.1-2024 requires similar modification of some historical practice for  
96621 the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate** commands.96622 Historically, **maps** that were subsets of other **maps** behaved differently depending on the order  
96623 in which they were defined. For example:96624 

```
:map! ab      short  
96625 :map! abc     long
```

96626 would always translate the characters "ab" to "short", regardless of how fast the characters  
96627 "abc" were entered. If the entry order was reversed:

```
96628 :map! abc    long
96629 :map! ab     short
```

96630 the characters "ab" would cause the editor to pause, waiting for the completing 'c' character,  
96631 and the characters might never be mapped to "short". For consistency and simplicity of  
96632 specification, POSIX.1-2024 requires that the shortest match be used at all times.

96633 The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified  
96634 because the timing capabilities of systems are often inexact and variable, and it may depend on  
96635 other factors such as the speed of the connection. The time should be long enough for the user to  
96636 be able to complete the sequence, but not long enough for the user to have to wait. Some  
96637 implementations of *vi* have added a **keytime** option, which permits users to set the number of  
96638 0,1 seconds the editor waits for the completing characters. Because mapped terminal function  
96639 and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input  
96640 mode, **maps** starting with <ESC> characters are generally exempted from this timeout period,  
96641 or, at least timed out differently.

## 96642 Mark

96643 Historically, users were able to set the ``previous context'' marks explicitly. In addition, the *ex*  
96644 commands " and " and the *vi* commands ", `', and " all referred to the same mark. In addition,  
96645 the previous context marks were not set if the command, with which the address setting the  
96646 mark was associated, failed. POSIX.1-2024 requires conformance to historical practice.  
96647 Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the  
96648 change was undone. POSIX.1-2024 requires conformance to historical practice.

96649 The description of the special events that set the ` and ' marks matches historical practice. For  
96650 example, historically the command */a/,b/* did not set the ` and ' marks, but the command  
96651 */a/,b/delete* did.

## 96652 Next

96653 Historically, any *ex* command could be entered as a *+command* argument to the **next** command,  
96654 although some (for example, **insert** and **append**) were known to confuse historical  
96655 implementations. POSIX.1-2024 requires that any command be permitted and that it behave as  
96656 specified. The **next** command can accept more than one file, so usage such as:

```
96657 next `ls [abc] `
```

96658 is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect  
96659 only one filename.

96660 Historically, the **next** command behaved differently from the **:rewind** command in that it  
96661 ignored the force flag if the **autowrite** flag was set. For consistency, POSIX.1-2024 does not  
96662 permit this behavior.

96663 Historically, the **next** command positioned the cursor as if the file had never been edited before,  
96664 regardless. POSIX.1-2024 does not permit this behavior, for consistency with the **edit** command.

96665 Implementations wanting to provide a counterpart to the **next** command that edited the  
96666 previous file have used the command **prev[ious]**, which takes no *file* argument. POSIX.1-2024  
96667 does not require this command.

96668 **Open**

96669 Historically, the **open** command would fail if the **open** edit option was not set. POSIX.1-2024  
 96670 does not mention the **open** edit option and does not require this behavior. Some historical  
 96671 implementations do not permit entering open mode from open or visual mode, only from *ex*  
 96672 mode. For consistency, POSIX.1-2024 does not permit this behavior.

96673 Historically, entering open mode from the command line (that is, *vi +open*) resulted in  
 96674 anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command  
 96675 <control>-G did not work. For consistency, POSIX.1-2024 does not permit this behavior.

96676 Historically, the **open** command only permitted ' / ' characters to be used as the search pattern  
 96677 delimiter. For consistency, POSIX.1-2024 requires that the search delimiters used by the **s**, **global**,  
 96678 and **v** commands be accepted as well.

96679 **Preserve**

96680 The **preserve** command does not historically cause the file to be considered unmodified for the  
 96681 purposes of future commands that may exit the editor. POSIX.1-2024 requires conformance to  
 96682 historical practice.

96683 Historical documentation stated that mail was not sent to the user when preserve was executed;  
 96684 however, historical implementations did send mail in this case. POSIX.1-2024 requires  
 96685 conformance to the historical implementations.

96686 **Print**

96687 The writing of NUL by the **print** command is not specified as a special case because the standard  
 96688 developers did not want to require *ex* to support NUL characters. Historically, characters were  
 96689 displayed using the ARPA standard mappings, which are as follows:

- 96690 1. Printable characters are left alone.
- 96691 2. Control characters less than \177 are represented as '^' followed by the character offset  
 96692 from the '@' character in the ASCII map; for example, \007 is represented as '^G'.
- 96693 3. \177 is represented as '^?' followed by '? '.

96694 The display of characters having their eighth bit set was less standard. Existing implementations  
 96695 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their  
 96696 eighth bit set as the two characters "M-" followed by the seven-bit display as described above.)  
 96697 The latter probably has the best claim to historical practice because it was used for the **-v** option  
 96698 of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

96699 No specific display format is required by POSIX.1-2024.

96700 Explicit dependence on the ASCII character set has been avoided where possible, hence the use  
 96701 of the phrase an "implementation-defined multi-character sequence" for the display of non-  
 96702 printable characters in preference to the historical usage of, for instance, "^I" for the <tab>.  
 96703 Implementations are encouraged to conform to historical practice in the absence of any strong  
 96704 reason to diverge.

96705 Historically, all *ex* commands beginning with the letter 'p' could be entered using capitalized  
 96706 versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command  
 96707 names. POSIX.1-2024 permits, but does not require, this historical practice because capital forms  
 96708 of the commands are used by some implementations for other purposes.

96709

**Put**

96710

96711

96712

96713

96714

96715

96716

96717

96718

96719

Historically, an *ex* **put** command, executed from open or visual mode, was the same as the open or visual mode **P** command, if the buffer was named and was cut in character mode, and the same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer was the source of the text, the entire line from which the text was taken was usually **put**, and the buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior. In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in errors as well, such as appending text that was unrelated to the (supposed) contents of the buffer. For consistency and simplicity of specification, POSIX.1-2024 does not permit these behaviors. All *ex* **put** commands are required to operate in line mode, and the contents of the buffers are not altered by changing the mode of the editor.

96720

**Read**

96721

96722

96723

96724

96725

Historically, an *ex* **read** command executed from open or visual mode, executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior. Historically, a **read** in open or visual mode from a program left the cursor at the last line read in, not the first. For consistency, POSIX.1-2024 does not permit this behavior.

96726

96727

Historical implementations of *ex* were unable to undo **read** commands that read from the output of a program. For consistency, POSIX.1-2024 does not permit this behavior.

96728

96729

96730

96731

96732

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified ``characters'', not ``bytes''. POSIX.1-2024 requires that the number of bytes be displayed, not the number of characters, because it may be difficult in multi-byte implementations to determine the number of characters read. Implementations are encouraged to clarify the message displayed to the user.

96733

96734

96735

96736

Historically, reads were not permitted on files other than type regular, except that FIFO files could be read (probably only because they did not exist when *ex* and *vi* were originally written). Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way to force the read. POSIX.1-2024 permits, but does not require, this behavior.

96737

**Recover**

96738

96739

96740

96741

Some historical implementations of the editor permitted users to recover the edit buffer contents from a previous edit session, and then exit without saving those contents (or explicitly discarding them). The intent of POSIX.1-2024 in requiring that the edit buffer be treated as already modified is to prevent this user error.

96742

**Rewind**

96743

96744

96745

Historical implementations supported the **rewind** command when the user was editing the first file in the list; that is, the file that the **rewind** command would edit. POSIX.1-2024 requires conformance to historical practice.

96746 **Substitute**

96747 Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the  
 96748 last regular expression used in any command as the pattern, the same as the **~** command. The **r**  
 96749 option is not required by POSIX.1-2024. Historically, the **c** and **g** options were toggled; for  
 96750 example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of  
 96751 specification, POSIX.1-2024 does not permit this behavior.

96752 The tilde command is often used to replace the last search RE. For example, in the sequence:

```
96753 s/red/blue/  
96754 /green  
96755 ~
```

96756 the **~** command is equivalent to:

```
96757 s/green/blue/
```

96758 Historically, *ex* accepted all of the following forms:

```
96759 s/abc/def/  
96760 s/abc/def  
96761 s/abc/  
96762 s/abc
```

96763 POSIX.1-2024 requires conformance to this historical practice.

96764 The **s** command presumes that the **^** character only occupies a single column in the display.  
 96765 Much of the *ex* and *vi* specification presumes that the **<space>** only occupies a single column in  
 96766 the display. There are no known character sets for which this is not true.

96767 Historically, the final column position for the substitute commands was based on previous  
 96768 column movements; a search for a pattern followed by a substitution would leave the column  
 96769 position unchanged, while a **0** command followed by a substitution would change the column  
 96770 position to the first non-**<blank>**. For consistency and simplicity of specification, POSIX.1-2024  
 96771 requires that the final column position always be set to the first non-**<blank>**.

96772 **Set**

96773 Historical implementations redisplayed all of the options for each occurrence of the **all** keyword.  
 96774 POSIX.1-2024 permits, but does not require, this behavior.

96775 **Tag**

96776 No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry.  
 96777 Historical practice has been to look for the path found in the **tags** file, based on the current  
 96778 directory. A useful extension found in some implementations is to look based on the directory  
 96779 containing the tags file that held the entry, as well. No requirement is made as to which reference  
 96780 for the tag in the tags file is used. This is deliberate, in order to permit extensions such as  
 96781 multiple entries in a tags file for a tag.

96782 Because users often specify many different tags files, some of which need not be relevant or exist  
 96783 at any particular time, POSIX.1-2024 requires that error messages about problem tags files be  
 96784 displayed only if the requested tag is not found, and then, only once for each time that the **tag**  
 96785 edit option is changed.

96786 The requirement that the current edit buffer be unmodified is only necessary if the file indicated  
 96787 by the tag entry is not the same as the current file (as defined by the current pathname).  
 96788 Historically, the file would be reloaded if the filename had changed, as well as if the filename

96789 was different from the current pathname. For consistency and simplicity of specification,  
96790 POSIX.1-2024 does not permit this behavior, requiring that the name be the only factor in the  
96791 decision.

96792 Historically, *vi* only searched for tags in the current file from the current cursor to the end of the  
96793 file, and therefore, if the **wrapsan** option was not set, tags occurring before the current cursor  
96794 were not found. POSIX.1-2024 considers this a bug, and implementations are required to search  
96795 for the first occurrence in the file, regardless.

#### 96796 **Undo**

96797 The **undo** description deliberately uses the word “modified”. The **undo** command is not  
96798 intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**,  
96799 or **recover**.

96800 Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes  
96801 attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and  
96802 sometimes, in the presence of maps, placing the cursor on the last line added or changed instead  
96803 of the first. POSIX.1-2024 requires a simplified behavior for consistency and simplicity of  
96804 specification.

#### 96805 **Version**

96806 The **version** command cannot be exactly specified since there is no widely-accepted definition of  
96807 what the version information should contain. Implementations are encouraged to do something  
96808 reasonably intelligent.

#### 96809 **Write**

96810 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified  
96811 “characters”, not “bytes”. POSIX.1-2024 requires that the number of bytes be displayed, not the  
96812 number of characters because it may be difficult in multi-byte implementations to determine the  
96813 number of characters written. Implementations are encouraged to clarify the message displayed  
96814 to the user.

96815 Implementation-defined tests are permitted so that implementations can make additional  
96816 checks; for example, for locks or file modification times.

96817 Historically, attempting to append to a nonexistent file caused an error. It has been left  
96818 unspecified in POSIX.1-2024 to permit implementations to let the **write** succeed, so that the  
96819 append semantics are similar to those of the historical *cs*.

96820 Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around  
96821 dealing with “empty” files was to always have a line in the edit buffer, no matter what, it wrote  
96822 them as files of a single, empty line. POSIX.1-2024 does not permit this behavior.

96823 Historically, *ex* restored standard output and standard error to their values as of when *ex* was  
96824 invoked, before writes to programs were performed. This could disturb the terminal  
96825 configuration as well as be a security issue for some terminals. POSIX.1-2024 does not permit  
96826 this, requiring that the program output be captured and displayed as if by the *ex* **print**  
96827 command.

### 96828 Adjust Window

96829 Historically, the line count was set to the value of the **scroll** option if the type character was end-  
 96830 of-file. This feature was broken on most historical implementations long ago, however, and is  
 96831 not documented anywhere. For this reason, POSIX.1-2024 is resolutely silent.

96832 Historically, the **z** command was <blank>-sensitive and **z +** and **z -** did different things than **z+**  
 96833 and **z-** because the type could not be distinguished from a flag. (The commands **z .** and **z =**  
 96834 were historically invalid.) POSIX.1-2024 requires conformance to this historical practice.

96835 Historically, the **z** command was further <blank>-sensitive in that the *count* could not be  
 96836 <blank>-delimited; for example, the commands **z= 5** and **z- 5** were also invalid. Because the  
 96837 *count* is not ambiguous with respect to either the type character or the flags, this is not permitted  
 96838 by POSIX.1-2024.

### 96839 Escape

96840 Historically, *ex* filter commands only read the standard output of the commands, letting  
 96841 standard error appear on the terminal as usual. The *vi* utility, however, read both standard  
 96842 output and standard error. POSIX.1-2024 requires the latter behavior for both *ex* and *vi*, for  
 96843 consistency.

96844 In Issue 8 the *system()* function was changed to require that the POSIX shell be invoked with  
 96845 "sh", "-c", "--", and *command* arguments to make it easier to execute programs with  
 96846 <hyphen-minus> ('-') or <plus-sign> ('+') as the first character of the program's filename. A  
 96847 similar request to have the *ex* **escape** command do the same was not accepted. Unlike *system()*  
 96848 (which always invokes a POSIX shell), *ex* invokes the program named by the **shell** edit option.  
 96849 For example, the *csh* and *tsh* shells that are frequently used as login shells do not recognize  
 96850 "--" after "-c" as an end-of-options indicator. The program need not even be one that  
 96851 recognizes any POSIX shell command line syntax. Some users invoke shell scripts to process  
 96852 lines that are being supplied to the specified utility. These utilities know that they will be given  
 96853 "-c" as a first argument and just ignore it. Any utilities used in this manner would have to be  
 96854 modified to skip over another argument (the "--") to find the desired argument.

### 96855 Shift Left and Shift Right

96856 Historically, it was possible to add shift characters to increase the effect of the command; for  
 96857 example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default  
 96858 1. POSIX.1-2024 requires conformance to historical practice.

### 96859 <control>-D

96860 Historically, the <control>-D command erased the prompt, providing the user with an unbroken  
 96861 presentation of lines from the edit buffer. This is not required by POSIX.1-2024; implementations  
 96862 are encouraged to provide it if possible. Historically, the <control>-D command took, and then  
 96863 ignored, a *count*. POSIX.1-2024 does not permit this behavior.



96864 **Write Line Number**

96865 Historically, the `ex =` command, when executed in `ex` mode in an empty edit buffer, reported 0,  
 96866 and from open or visual mode, reported 1. For consistency and simplicity of specification,  
 96867 POSIX.1-2024 does not permit this behavior.

96868 **Execute**

96869 Historically, `ex` did not correctly handle the inclusion of text input commands (that is, **append**,  
 96870 **insert**, and **change**) in executed buffers. POSIX.1-2024 does not permit this exclusion for  
 96871 consistency.

96872 Historically, the logical contents of the buffer being executed did not change if the buffer itself  
 96873 were modified by the commands being executed; that is, buffer execution did not support self-  
 96874 modifying code. POSIX.1-2024 requires conformance to historical practice.

96875 Historically, the `@` command took a range of lines, and the `@` buffer was executed once per line,  
 96876 with the current line (`'.'`) set to each specified line. POSIX.1-2024 requires conformance to  
 96877 historical practice.

96878 Some historical implementations did not notice if errors occurred during buffer execution. This,  
 96879 coupled with the ability to specify a range of lines for the `ex @` command, makes it trivial to  
 96880 cause them to drop **core**. POSIX.1-2024 requires that implementations stop buffer execution if  
 96881 any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are  
 96882 replaced (for example, the buffer executes the `ex :edit` command).

96883 **Regular Expressions in ex**

96884 Historical practice is that the characters in the replacement part of the last `s` command—that is,  
 96885 those matched by entering a `'~'` in the regular expression—were not further expanded by the  
 96886 regular expression engine. So, if the characters contained the string `"a."`, they would match  
 96887 `'a'` followed by `"."`, and not `'a'` followed by any character. POSIX.1-2024 requires  
 96888 conformance to historical practice.

96889 **Edit Options in ex**

96890 The following paragraphs describe the historical behavior of some edit options that were not, for  
 96891 whatever reason, included in POSIX.1-2024. Implementations are strongly encouraged to only  
 96892 use these names if the functionality described here is fully supported.

96893 **extended** The **extended** edit option has been used in some implementations of `vi` to provide  
 96894 extended regular expressions instead of basic regular expressions. This option was  
 96895 omitted from POSIX.1-2024 because it is not widespread historical practice.

96896 **flash** The **flash** edit option historically caused the screen to flash instead of beeping on  
 96897 error. This option was omitted from POSIX.1-2024 because it is not found in some  
 96898 historical implementations.

96899 **hardtabs** The **hardtabs** edit option historically defined the number of columns between  
 96900 hardware tab settings. This option was omitted from POSIX.1-2024 because it was  
 96901 believed to no longer be generally useful.

96902 **modeline** The **modeline** (sometimes named **modelines**) edit option historically caused `ex` or  
 96903 `vi` to read the five first and last lines of the file for editor commands. This option is  
 96904 a security problem, and vendors are strongly encouraged to delete it from  
 96905 historical implementations.

96906	<b>open</b>	The <b>open</b> edit option historically disallowed the <i>ex</i> <b>open</b> and <b>visual</b> commands. This edit option was omitted because these commands are required by POSIX.1-2024.
96907		
96908		
96909	<b>optimize</b>	The <b>optimize</b> edit option historically expedited text throughput by setting the terminal to not do automatic <carriage-return> characters when printing more than one logical line of output. This option was omitted from POSIX.1-2024 because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.
96910		
96911		
96912		
96913		
96914	<b>ruler</b>	The <b>ruler</b> edit option has been used in some implementations of <i>vi</i> to present a current row/column ruler for the user. This option was omitted from POSIX.1-2024 because it is not widespread historical practice.
96915		
96916		
96917	<b>sourceany</b>	The <b>sourceany</b> edit option historically caused <i>ex</i> or <i>vi</i> to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.
96918		
96919		
96920		
96921	<b>timeout</b>	The <b>timeout</b> edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from POSIX.1-2024 because its behavior is now standard, it is not widely useful, and it was rarely documented.
96922		
96923		
96924		
96925	<b>verbose</b>	The <b>verbose</b> edit option has been used in some implementations of <i>vi</i> to cause <i>vi</i> to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical <i>vi</i> only alerted the terminal and presented no message for such errors. The historical editor option <b>terse</b> did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from POSIX.1-2024 because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users.
96926		
96927		
96928		
96929		
96930		
96931		
96932		
96933		
96934	<b>wraplen</b>	The <b>wraplen</b> edit option has been used in some implementations of <i>vi</i> to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from POSIX.1-2024 because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.
96935		
96936		
96937		
96938		
96939		
96940	<b>autoindent, ai</b>	
96941		Historically, the command <b>0a</b> did not do any autoindentation, regardless of the current indentation of line 1. POSIX.1-2024 requires that any indentation present in line 1 be used.
96942		
96943	<b>autoprint, ap</b>	
96944		Historically, the <b>autoprint</b> edit option was not completely consistent or based solely on modifications to the edit buffer. Exceptions were the <b>read</b> command (when reading from a file, but not from a filter), the <b>append</b> , <b>change</b> , <b>insert</b> , <b>global</b> , and <b>v</b> commands, all of which were not affected by <b>autoprint</b> , and the <b>tag</b> command, which was affected by <b>autoprint</b> . POSIX.1-2024 requires conformance to historical practice.
96945		
96946		
96947		
96948		
96949		Historically, the <b>autoprint</b> option only applied to the last of multiple commands entered using <vertical-line> delimiters; for example, <b>delete</b> <newline> was affected by <b>autoprint</b> , but
96950		

96951 **delete** | **version** <newline> was not. POSIX.1-2024 requires conformance to historical practice.

96952 **autowrite, aw**

96953 Appending the '!' character to the *ex* **next** command to avoid performing an automatic write  
96954 was not supported in historical implementations. POSIX.1-2024 requires that the behavior match  
96955 the other *ex* commands for consistency.

96956 **ignorecase, ic**

96957 Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to  
96958 counter-intuitive situations when uppercase characters were used in range expressions.  
96959 Historically, the process was as follows:

- 96960 1. Take a line of text from the edit buffer.
- 96961 2. Convert uppercase to lowercase in text line.
- 96962 3. Convert uppercase to lowercase in regular expressions, except in character class  
96963 specifications.
- 96964 4. Match regular expressions against text.

96965 This would mean that, with **ignorecase** in effect, the text:

96966 The cat sat on the mat

96967 would be matched by

96968 /<sup>^</sup>the/

96969 but not by:

96970 /<sup>^</sup>[A-Z]he/

96971 For consistency with other commands implementing regular expressions, POSIX.1-2024 does not  
96972 permit this behavior.

96973 **paragraphs, para**

96974 The ISO POSIX-2:1993 standard made the default **paragraphs** and **sections** edit options  
96975 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text  
96976 formatter, and a “portable user” could use the {, }, [[, ]], (, and ) commands in open or visual  
96977 mode and have the cursor stop in unexpected places. POSIX.1-2024 specifies their values in the  
96978 POSIX locale because the unusual grouping (they only work when grouped into two characters  
96979 at a time) means that they cannot be used for general-purpose movement, regardless.

96980 **readonly**

96981 Implementations are encouraged to provide the best possible information to the user as to the  
96982 read-only status of the file, with the exception that they should not consider the current special  
96983 privileges of the process. This provides users with a safety net because they must force the  
96984 overwrite of read-only files, even when running with additional privileges.

96985 The **readonly** edit option specification largely conforms to historical practice. The only  
96986 difference is that historical implementations did not notice that the user had set the **readonly**  
96987 edit option in cases where the file was already marked read-only for some reason, and would  
96988 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were  
96989 replaced. This behavior is disallowed by POSIX.1-2024.

96990 **report**

96991 The requirement that lines copied to a buffer interact differently than deleted lines is historical  
 96992 practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be  
 96993 written, but 4 lines must be copied before a report is written.

96994 The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based  
 96995 on the total number of lines added or deleted during the command execution, and that  
 96996 commands executed by the **global** and **v** commands not present reports, is historical practice.  
 96997 POSIX.1-2024 extends historical practice by requiring that buffer execution be treated similarly.  
 96998 The reasons for this are two-fold. Historically, only the report by the last command executed  
 96999 from the buffer would be seen by the user, as each new report would overwrite the last. In  
 97000 addition, the standard developers believed that buffer execution had more in common with  
 97001 **global** and **v** commands than it did with other *ex* commands, and should behave similarly, for  
 97002 consistency and simplicity of specification.

97003 **showmatch, sm**

97004 The length of time the cursor spends on the matching character is unspecified because the  
 97005 timing capabilities of systems are often inexact and variable. The time should be long enough for  
 97006 the user to notice, but not long enough for the user to become annoyed. Some implementations  
 97007 of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals  
 97008 the cursor pauses on the matching character.

97009 **showmode**

97010 The **showmode** option has been used in some historical implementations of *ex* and *vi* to display  
 97011 the current editing mode when in open or visual mode. The editing modes have generally  
 97012 included “command” and “input”, and sometimes other modes such as “replace” and  
 97013 “change”. The string was usually displayed on the bottom line of the screen at the far right-hand  
 97014 corner. In addition, a preceding '\*' character often denoted whether the contents of the edit  
 97015 buffer had been modified. The latter display has sometimes been part of the **showmode** option,  
 97016 and sometimes based on another option. This option was not available in the 4 BSD historical  
 97017 implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is  
 97018 required by POSIX.1-2024.

97019 The **smd** shorthand for the **showmode** option was not present in all historical implementations  
 97020 of the editor. POSIX.1-2024 requires it, for consistency.

97021 Not all historical implementations of the editor displayed a mode string for command mode,  
 97022 differentiating command mode from text input mode by the absence of a mode string.  
 97023 POSIX.1-2024 permits this behavior for consistency with historical practice, but implementations  
 97024 are encouraged to provide a display string for both modes.

97025 **slowopen**

97026 Historically, the **slowopen** option was automatically set if the terminal baud rate was less than  
 97027 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen**  
 97028 option had two effects. First, when inserting characters in the middle of a line, characters after  
 97029 the cursor would not be pushed ahead, but would appear to be overwritten. Second, when  
 97030 creating a new line of text, lines after the current line would not be scrolled down, but would  
 97031 appear to be overwritten. In both cases, ending text input mode would cause the screen to be  
 97032 refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently  
 97033 intelligent caused the editor to ignore the **slowopen** option. POSIX.1-2024 permits most  
 97034 historical behavior, extending historical practice to require **slowopen** behaviors if the edit option

97035 is set by the user.

#### 97036 **tags**

97037 The default path for tags files is left unspecified as implementations may have their own **tags**  
97038 implementations that do not correspond to the historical ones. The default **tags** option value  
97039 should probably at least include the file **./tags**.

#### 97040 **term**

97041 Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial  
97042 terminal information was loaded. This is permitted by POSIX.1-2024; however, implementations  
97043 are encouraged to permit the user to modify their terminal type at any time.

#### 97044 **terse**

97045 Historically, the **terse** edit option optionally provided a shorter, less descriptive error message,  
97046 for some error messages. This is permitted, but not required, by POSIX.1-2024. Historically, most  
97047 common visual mode errors (for example, trying to move the cursor past the end of a line) did  
97048 not result in an error message, but simply alerted the terminal. Implementations wishing to  
97049 provide messages for novice users are urged to do so based on the **edit** option **verbose**, and not  
97050 **terse**.

#### 97051 **window**

97052 In historical implementations, the default for the **window** edit option was based on the baud  
97053 rate as follows:

97054 1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for  
97055 example, the line:

```
97056 set w300=12
```

97057 would set the window option to 12 if the baud rate was less than 1 200.

97058 2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.

97059 3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

97060 The **w300**, **w1200**, and **w9600** options do not appear in POSIX.1-2024 because of their  
97061 dependence on specific baud rates.

97062 In historical implementations, the size of the window displayed by various commands was  
97063 related to, but not necessarily the same as, the **window** edit option. For example, the size of the  
97064 window was set by the *ex* command **visual 10**, but it did not change the value of the **window**  
97065 edit option. However, changing the value of the **window** edit option did change the number of  
97066 lines that were displayed when the screen was repainted. POSIX.1-2024 does not permit this  
97067 behavior in the interests of consistency and simplicity of specification, and requires that all  
97068 commands that change the number of lines that are displayed do it by setting the value of the  
97069 **window** edit option.

97070 **wrapmargin, wm**

97071 Historically, the **wrapmargin** option did not affect maps inserting characters that also had  
 97072 associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used  
 97073 maps that depend on this behavior. For consistency and simplicity of specification,  
 97074 POSIX.1-2024 does not permit this behavior.

97075 Historically, **wrapmargin** was calculated using the column display width of all characters on the  
 97076 screen. For example, an implementation using "**^I**" to represent <tab> characters when the **list**  
 97077 edit option was set, where '**^**' and '**I**' each took up a single column on the screen, would  
 97078 calculate the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option  
 97079 similarly changed the effective length of the line as well. POSIX.1-2024 requires conformance to  
 97080 historical practice.

97081 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
 97082 but this has been modified in this version.

97083 **FUTURE DIRECTIONS**

97084 If this utility is directed to create a new directory entry that contains any bytes that have the  
 97085 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 97086 error. A future version of this standard may require implementations to treat this as an error.

97087 **SEE ALSO**

97088 [Section 2.9.1.4](#) (on page 2502), *ctags*, *ed*, *sed*, *sh*, *stty*, *vi*

97089 XBD [Table 5-1](#) (on page 113), [Chapter 8](#) (on page 167), [Section 9.3](#) (on page 181), [Section 12.2](#) (on  
 97090 page 215)

97091 XSH *access()*

97092 **CHANGE HISTORY**

97093 First released in Issue 2.

97094 **Issue 5**

97095 The FUTURE DIRECTIONS section is added.

97096 **Issue 6**

97097 This utility is marked as part of the User Portability Utilities option.

97098 The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.

97099 The following new requirements on POSIX implementations derive from alignment with the  
 97100 Single UNIX Specification:

- 97101 • In the **map** command description, the sequence *#digit* is added.
- 97102 • The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

97103 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This  
 97104 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,  
 97105 #55, #56, #57, #61, #62, #63, #64, #65, and #78.

97106 The **-l** option is removed.

97107 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/23 is applied, correcting a URL.

97108 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/8 is applied, making an editorial  
 97109 correction in the EXTENDED DESCRIPTION.

97110 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/9 is applied, removing text describing  
 97111 behavior on systems with bytes consisting of more than eight bits.

97112 **Issue 7**

97113 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is  
97114 ' \_ '.

97115 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

97116 Austin Group Interpretation 1003.1-2001 #121 is applied, clarifying the *ex write* command.

97117 Austin Group Interpretation 1003.1-2001 #156 is applied.

97118 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97119 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0093 [584] is applied.

97120 **Issue 8**

97121 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
97122 filenames containing any bytes that have the encoded value of a <newline> character.

97123 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97124 Austin Group Defect 1185 is applied, changing the SIGCONT entry in ASYNCHRONOUS  
97125 EVENTS and adding a SIGWINCH entry.

97126 Austin Group Defect 1251 is applied, clarifying the *set* command, changing ``addr`` to ``2addr``  
97127 in the *!* command synopsis, and adding spaces in some synopsis lines.

97128 Austin Group Defect 1281 is applied, changing the description of the *substitute* command to  
97129 clarify that it is an error if the substitution fails on every addressed line.

97130 Austin Group Defect 1298 is applied, changing the CONSEQUENCES OF ERRORS section.

97131 Austin Group Defect 1378 is applied, changing the description of the *LC\_MESSAGES*  
97132 environment variable.

97133 Austin Group Defect 1529 is applied, changing the synopsis of the *escape* command and adding  
97134 related paragraphs to the APPLICATION USAGE and RATIONALE sections.

97135 Austin Group Defect 1642 is applied, changing the description of the *redraw* edit option.

97136 Austin Group Defect 1662 is applied, clarifying requirements relating to delimiters in addresses  
97137 and in *s* commands.

97138 **NAME**

97139 expand — convert tabs to spaces

97140 **SYNOPSIS**97141 expand [-t *tablist*] [*file...*]97142 **DESCRIPTION**

97143 The *expand* utility shall write files or the standard input to the standard output with <tab>  
 97144 characters replaced with one or more <space> characters needed to pad to the next tab stop. Any  
 97145 <backspace> characters shall be copied to the output and cause the column position count for  
 97146 tab stop calculations to be decremented; the column position count shall not be decremented  
 97147 below zero.

97148 **OPTIONS**97149 The *expand* utility shall conform to XBD [Section 12.2](#) (on page 215).

97150 The following option shall be supported:

97151 **-t *tablist*** Specify the tab stops. The application shall ensure that the argument *tablist* consists  
 97152 of either a single positive decimal integer or a list of tabstops. If a single number is  
 97153 given, tabs shall be set that number of column positions apart instead of the  
 97154 default 8.

97155 If a list of tabstops is given, the application shall ensure that it consists of a list of  
 97156 two or more positive decimal integers, separated by <blank> or <comma>  
 97157 characters, in ascending order. The <tab> characters shall be set at those specific  
 97158 column positions. Each tab stop *N* shall be an integer value greater than zero, and  
 97159 the list is in strictly ascending order. This is taken to mean that, from the start of a  
 97160 line of output, tabbing to position *N* shall cause the next character output to be in  
 97161 the (*N*+1)th column position on that line.

97162 In the event of *expand* having to process a <tab> at a position beyond the last of  
 97163 those specified in a multiple tab-stop list, the <tab> shall be replaced by a single  
 97164 <space> in the output.

97165 **OPERANDS**

97166 The following operand shall be supported:

97167 *file* The pathname of a text file to be used as input.97168 **STDIN**

97169 See the INPUT FILES section.

97170 **INPUT FILES**

97171 Input files shall be text files.

97172 **ENVIRONMENT VARIABLES**97173 The following environment variables shall affect the execution of *expand*:

97174 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 97175 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 97176 variables used to determine the values of locale categories.)

97177 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 97178 internationalization variables.

97179 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 97180 characters (for example, single-byte as opposed to multi-byte characters in  
 97181 arguments and input files), the processing of <tab> and <space> characters, and  
 97182 for the determination of the width in column positions each character would



- 97183                    occupy on an output device.
- 97184            *LC\_MESSAGES*
- 97185                    Determine the locale that should be used to affect the format and contents of
- 97186                    diagnostic messages written to standard error.
- 
- 97187 XSI            *NLSPATH*    Determine the location of messages objects and message catalogs.
- 97188 **ASYNCHRONOUS EVENTS**
- 97189            Default.
- 97190 **STDOUT**
- 97191            The standard output shall be equivalent to the input files with <tab> characters converted into
- 97192            the appropriate number of <space> characters.
- 97193 **STDERR**
- 97194            The standard error shall be used only for diagnostic messages.
- 97195 **OUTPUT FILES**
- 97196            None.
- 97197 **EXTENDED DESCRIPTION**
- 97198            None.
- 97199 **EXIT STATUS**
- 97200            The following exit values shall be returned:
- 97201            0    Successful completion
- 97202            >0   An error occurred.
- 97203 **CONSEQUENCES OF ERRORS**
- 97204            The *expand* utility shall terminate with an error message and non-zero exit status upon
- 97205            encountering difficulties accessing one of the *file* operands.
- 97206 **APPLICATION USAGE**
- 97207            None.
- 97208 **EXAMPLES**
- 97209            None.
- 97210 **RATIONALE**
- 97211            The *expand* utility is useful for preprocessing text files (before sorting, looking at specific
- 97212            columns, and so on) that contain <tab> characters.
- 97213            See XBD [Section 3.75](#) (on page 42).
- 97214            The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8
- 97215            mandates that *expand* shall accept the integers (within the single argument) separated using
- 97216            either <comma> or <blank> characters.
- 97217            Earlier versions of this standard allowed the following form in the SYNOPSIS:
- 97218            `expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]`
- 97219            This form is no longer specified by POSIX.1-2024 but may be present in some implementations.
- 97220 **FUTURE DIRECTIONS**
- 97221            None.

97222 **SEE ALSO**97223 *tabs, unexpand*97224 XBD [Section 3.75](#) (on page 42), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)97225 **CHANGE HISTORY**

97226 First released in Issue 4.

97227 **Issue 6**

97228 This utility is marked as part of the User Portability Utilities option.

97229 The APPLICATION USAGE section is added.

97230 The obsolescent SYNOPSIS is removed.

97231 The *LC\_CTYPE* environment variable description is updated to align with the IEEE P1003.2b draft standard.

97233 The normative text is reworded to avoid use of the term “must” for application requirements.

97234 **Issue 7**

97235 Austin Group Interpretation 1003.1-2001 #027 is applied.

97236 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97237 The *expand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.97239 **Issue 8**97240 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97241 **NAME**97242            *expr* — evaluate arguments as an expression97243 **SYNOPSIS**97244            *expr operand...*97245 **DESCRIPTION**97246            The *expr* utility shall evaluate an expression and write the result to standard output.97247 **OPTIONS**

97248            None.

97249 **OPERANDS**97250            The single expression evaluated by *expr* shall be formed from the *operand* operands, as described in the EXTENDED DESCRIPTION section. The application shall ensure that each of the expression operator symbols:

97253            ( ) | &amp; = &gt; &gt;= &lt; &lt;= != + - \* / % :

97254            and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.97255 **STDIN**

97256            Not used.

97257 **INPUT FILES**

97258            None.

97259 **ENVIRONMENT VARIABLES**97260            The following environment variables shall affect the execution of *expr*:97261            *LANG*        Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)97264            *LC\_ALL*     If set to a non-empty string value, override the values of all the other internationalization variables.97266            *LC\_COLLATE*

97267                    Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions and by the string comparison operators.

97270            *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behavior of character classes within regular expressions.97273            *LC\_MESSAGES*

97274                    Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

97276 XSI        *NLSPATH* Determine the location of messages objects and message catalogs.97277 **ASYNCHRONOUS EVENTS**

97278            Default.

97279 **STDOUT**97280            The *expr* utility shall evaluate the expression and write the result, followed by a <newline>, to standard output.

97281

97282 **STDERR**

97283 The standard error shall be used only for diagnostic messages.

97284 **OUTPUT FILES**

97285 None.

97286 **EXTENDED DESCRIPTION**

97287 The formation of the expression to be evaluated is shown in the following table. The symbols  
 97288 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the  
 97289 expression operator symbols (all separate arguments) by recursive application of the constructs  
 97290 described in the table. The expressions are listed in order of decreasing precedence, with equal-  
 97291 precedence operators grouped between horizontal lines. All of the operators shall be left-  
 97292 associative.

Expression	Description
<i>integer</i>	An argument consisting only of an (optional) unary minus followed by digits.
<i>string</i>	A string argument; see below.
( <i>expr</i> )	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
<i>expr1</i> : <i>expr2</i>	Matching expression; see below.
<i>expr1</i> * <i>expr2</i> <i>expr1</i> / <i>expr2</i>	Multiplication of decimal integer-valued arguments. Integer division of decimal integer-valued arguments, producing an integer result.
<i>expr1</i> % <i>expr2</i>	Remainder of integer division of decimal integer-valued arguments.
<i>expr1</i> + <i>expr2</i> <i>expr1</i> - <i>expr2</i>	Addition of decimal integer-valued arguments. Subtraction of decimal integer-valued arguments.
<i>expr1</i> = <i>expr2</i> <i>expr1</i> > <i>expr2</i> <i>expr1</i> >= <i>expr2</i> <i>expr1</i> < <i>expr2</i> <i>expr1</i> <= <i>expr2</i> <i>expr1</i> != <i>expr2</i>	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false. Equal. Greater than. Greater than or equal. Less than. Less than or equal. Not equal.
<i>expr1</i> & <i>expr2</i>	Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.
<i>expr1</i>   <i>expr2</i>	Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.

97324 **Matching Expression**

97325 The ':' matching operator shall compare the string resulting from the evaluation of *expr1* with  
 97326 the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax  
 97327 shall be that defined in XBD [Section 9.3](#) (on page 181), except that all patterns are anchored to  
 97328 the beginning of the string (that is, only sequences starting at the first character of a string are  
 97329 matched by the regular expression) and, therefore, it is unspecified whether '^' is a special  
 97330 character in that context. Usually, the matching operator shall return a string representing the  
 97331 number of characters matched ('0' on failure). Alternatively, if the pattern contains at least one  
 97332 regular expression subexpression "\(...\)", the string matched by the back-reference  
 97333 expression "\1" shall be returned. If the back-reference expression "\1" does not match, then  
 97334 the null string shall be returned.

97335 **Identification as Integer or String**

97336 An argument or the value of a subexpression that consists only of an optional unary minus  
 97337 followed by digits is a candidate for treatment as an integer if it is used as the left argument to  
 97338 the | operator or as either argument to any of the following operators: & = > >= < <= != +  
 97339 - \* / %. Otherwise, the argument or subexpression value shall be treated as a string.

97340 The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

97341 **EXIT STATUS**

97342 The following exit values shall be returned:

- 97343 0 The *expression* evaluated to neither null nor zero, and the output specified in STDOUT was  
 97344 successfully written to standard output.
- 97345 1 The *expression* evaluated to null or zero, and the output specified in STDOUT was  
 97346 successfully written to standard output.
- 97347 2 Invalid *expression* error.
- 97348 >2 Another error occurred.

97349 **CONSEQUENCES OF ERRORS**

97350 Default.

97351 **APPLICATION USAGE**

97352 The *expr* utility has a rather difficult syntax:

- 97353 • Many of the operators are also shell control operators or reserved words, so they have to  
 97354 be escaped on the command line.
- 97355 • Each part of the expression is composed of separate arguments, so liberal usage of <blank>  
 97356 characters is required. For example:

97357	Invalid	Valid
97358	<i>expr</i> 1+2	<i>expr</i> 1 + 2
97359	<i>expr</i> "1 + 2"	<i>expr</i> 1 + 2
97360	<i>expr</i> 1 + (2 * 3)	<i>expr</i> 1 + \( 2 \* 3 \)

97361 In many cases, the arithmetic and string features provided as part of the shell command  
 97362 language are easier to use than their equivalents in *expr*. Newly written scripts should avoid  
 97363 *expr* in favor of the new features within the shell; see [Section 2.5](#) (on page 2478) and [Section 2.6.4](#)  
 97364 (on page 2490).

97365 After argument processing by the shell, *expr* is not required to be able to tell the difference  
 97366 between an operator and an operand except by the value. If "\$a" is '=', the command:

97367 `expr "$a" = '='`

97368 looks like:

97369 `expr = = =`

97370 as the arguments are passed to *expr* (and they all may be taken as the '=' operator). The  
97371 following works reliably:

97372 `expr "X$a" = X=`

97373 Also note that this volume of POSIX.1-2024 permits implementations to extend utilities. The *expr*  
97374 utility permits the integer arguments to be preceded with a unary minus. This means that an  
97375 integer argument could look like an option. Therefore, the conforming application must employ  
97376 the "--" construct of Guideline 10 of XBD [Section 12.2](#) (on page 215) to protect its operands if  
97377 there is any chance the first operand might be a negative integer (or any string with a leading  
97378 minus).

97379 For testing string equality the *test* utility is preferred over *expr*, as it is usually implemented as a  
97380 shell built-in. However, the functionality is not quite the same because the *expr* = and !=  
97381 operators check whether strings collate equally, whereas *test* checks whether they are identical.  
97382 Therefore, they can produce different results in locales where the collation sequence does not  
97383 have a total ordering of all characters (see XBD [Section 7.3.2](#), on page 139).

#### 97384 EXAMPLES

97385 The following command:

97386 `a=$(expr "$a" + 1)`

97387 adds 1 to the variable *a*.

97388 The following command, for "\$a" equal to either */usr/abc/file* or just *file*:

97389 `expr $a : '.*\/(.*)' \| $a`

97390 returns the last segment of a pathname (that is, *file*). Applications should avoid the character  
97391 '/' used alone as an argument; *expr* may interpret it as the division operator.

97392 The following command:

97393 `expr "//$a" : '.*\/(.*)'`

97394 is a better representation of the previous example. The addition of the "//" characters  
97395 eliminates any ambiguity about the division operator and simplifies the whole expression. Also  
97396 note that pathnames may contain characters contained in the *IFS* variable and should be quoted  
97397 to avoid having "\$a" expand into multiple arguments.

97398 The following command:

97399 `expr "X$VAR" : '.*' - 1`

97400 returns the number of characters in *VAR*.

#### 97401 RATIONALE

97402 In an early proposal, EREs were used in the matching expression syntax. This was changed to  
97403 BREs to avoid breaking historical applications.

97404 The use of a leading <circumflex> in the BRE is unspecified because many historical  
97405 implementations have treated it as a special character, despite their system documentation. For  
97406 example:

97407 `expr foo : ^foo`      `expr ^foo : ^foo`

97408 return 3 and 0, respectively, on those systems; their documentation would imply the reverse.  
97409 Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on  
97410 this undocumented feature.

#### 97411 **FUTURE DIRECTIONS**

97412 None.

#### 97413 **SEE ALSO**

97414 [Section 2.5](#) (on page 2478), [Section 2.6.4](#) (on page 2490)

97415 [XBD Section 7.3.2](#) (on page 139), [Chapter 8](#) (on page 167), [Section 9.3](#) (on page 181), [Section 12.2](#)  
97416 (on page 215)

#### 97417 **CHANGE HISTORY**

97418 First released in Issue 2.

#### 97419 **Issue 5**

97420 The FUTURE DIRECTIONS section is added.

#### 97421 **Issue 6**

97422 The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE  
97423 PASC Interpretation 1003.2 #104.

97424 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 97425 **Issue 7**

97426 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

97427 The SYNOPSIS and OPERANDS sections are revised to explicitly state that the name of each of  
97428 the operands is *operand*.

97429 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0094 [942], XCU/TC2-2008/0095  
97430 [709], XCU/TC2-2008/0096 [942], XCU/TC2-2008/0097 [963], and XCU/TC2-2008/0098 [942]  
97431 are applied.

#### 97432 **Issue 8**

97433 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97434 Austin Group Defect 1500 is applied, changing the EXIT STATUS section.

97435 Austin Group Defect 1757 is applied, changing "`[\ ( . . . \ ) ]`" to "`[\ ( . . . \ ) ]`".

97436 **NAME**

97437 false — return false value

97438 **SYNOPSIS**

97439 false

97440 **DESCRIPTION**97441 The *false* utility shall return with a non-zero exit code.97442 **OPTIONS**

97443 None.

97444 **OPERANDS**

97445 None.

97446 **STDIN**

97447 Not used.

97448 **INPUT FILES**

97449 None.

97450 **ENVIRONMENT VARIABLES**

97451 None.

97452 **ASYNCHRONOUS EVENTS**

97453 Default.

97454 **STDOUT**

97455 Not used.

97456 **STDERR**

97457 Not used.

97458 **OUTPUT FILES**

97459 None.

97460 **EXTENDED DESCRIPTION**

97461 None.

97462 **EXIT STATUS**97463 The *false* utility shall always exit with a value between 1 and 125, inclusive.97464 **CONSEQUENCES OF ERRORS**

97465 Default.

97466 **APPLICATION USAGE**

97467 None.

97468 **EXAMPLES**

97469 None.

97470 **RATIONALE**

97471 None.

97472 **FUTURE DIRECTIONS**

97473 None.

97474 **SEE ALSO**97475 *true*



97476 **CHANGE HISTORY**

97477 First released in Issue 2.

97478 **Issue 6**97479 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/24 is applied, changing the STDERR  
97480 section from ``None.'' to ``Not used.'' for alignment with [Section 1.4](#) (on page 2462).97481 **Issue 8**

97482 Austin Group Defect 1321 is applied, changing the EXIT STATUS section.

97483 **NAME**97484 `fc` — process the command history list97485 **SYNOPSIS**

```
97486 UP fc [-r] [-e editor] [first [last]]
97487 fc -l [-nr] [first [last]]
97488 fc -s [old=new] [first]
```

97489 **DESCRIPTION**

97490 The `fc` utility shall list, or shall edit and re-execute, commands previously entered to an  
 97491 interactive `sh`.

97492 The command history list shall reference commands by number. The first number in the list is  
 97493 selected arbitrarily. The relationship of a number to its command shall not change except when  
 97494 the user logs in and no other process is accessing the list, at which time the system may reset the  
 97495 numbering to start the oldest retained command at another number (usually 1). When the  
 97496 number reaches an implementation-defined upper limit, which shall be no smaller than the  
 97497 value in `HISTSIZE` or 32767 (whichever is greater), the shell may wrap the numbers, starting the  
 97498 next command with a lower number (usually 1). However, despite this optional wrapping of  
 97499 numbers, `fc` shall maintain the time-ordering sequence of the commands. For example, if four  
 97500 commands in sequence are given the numbers 32766, 32767, 1 (wrapped), and 2 as they are  
 97501 executed, command 32767 is considered the command previous to 1, even though its number is  
 97502 higher.

97503 When commands are edited (when the `-l` option is not specified), the resulting lines shall be  
 97504 entered at the end of the history list and then re-executed by `sh`. The `fc` command that caused the  
 97505 editing shall not be entered into the history list. If the editor returns a non-zero exit status, this  
 97506 shall suppress the entry into the history list and the command re-execution. Any command line  
 97507 variable assignments or redirection operators used with `fc` shall affect both the `fc` command itself  
 97508 as well as the command that results; for example:

```
97509 fc -s -- -l 2>/dev/null
```

97510 reinvokes the previous command, suppressing standard error for both `fc` and the previous  
 97511 command.

97512 **OPTIONS**

97513 The `fc` utility shall conform to XBD [Section 12.2](#) (on page 215).

97514 The following options shall be supported:

- 97515 **-e editor** Use the editor named by `editor` to edit the commands. The `editor` string is a utility  
 97516 name, subject to search via the `PATH` variable (see XBD [Chapter 8](#), on page 167).  
 97517 The value in the `FCEDIT` variable shall be used as a default when `-e` is not  
 97518 specified. If `FCEDIT` is null or unset, `ed` shall be used as the editor.
- 97519 **-l** (The letter ell.) List the commands rather than invoking an editor on them. The  
 97520 commands shall be written in the sequence indicated by the `first` and `last` operands,  
 97521 as affected by `-r`, with each command preceded by the command number.
- 97522 **-n** Suppress command numbers when listing with `-l`.
- 97523 **-r** Reverse the order of the commands listed (with `-l`) or edited (with neither `-l` nor  
 97524 `-s`).

- 97525            -s            Re-execute the command without invoking an editor.
- 97526   **OPERANDS**
- 97527            The following operands shall be supported:
- 97528            *first, last*    Select the commands to list or edit. The number of previous commands that can be accessed shall be determined by the value of the *HISTSIZE* variable. The value of *first* or *last* or both shall be one of the following:
- 97531            [+]*number*    A positive number representing a command number; command numbers can be displayed with the -l option.
- 97532
- 97533            -*number*        A negative decimal number representing the command that was executed *number* of commands previously. For example, -1 is the immediately previous command.
- 97534
- 97535
- 97536            *string*         A string indicating the most recently entered command that begins with that string. If the *old=new* operand is not also specified with -s, the string form of the *first* operand cannot contain an embedded <equals-sign>.
- 97537
- 97538
- 97539
- 97540            When the synopsis form with -s is used:
- 97541
  - If *first* is omitted, the previous command shall be used.

97542            For the synopsis forms without -s:

97543            
  - If *last* is omitted, *last* shall default to the previous command when -l is specified; otherwise, it shall default to *first*.
  - If *first* and *last* are both omitted, the previous 16 commands shall be listed or the previous single command shall be edited (based on the -l option).
  - If *first* and *last* are both present, all of the commands from *first* to *last* shall be edited (without -l) or listed (with -l). Editing multiple commands shall be accomplished by presenting to the editor all of the commands at one time, each command starting on a new line. If *first* represents a newer command than *last*, the commands shall be listed or edited in reverse sequence, equivalent to using -r. For example, the following commands on the first line are equivalent to the corresponding commands on the second:

97544

97545            fc -r 10 20        fc     30 40

97546            fc     20 10        fc -r 40 30

97547

97548

97549

97550

97551

97552

97553

97554

97555

97556            
  - When a range of commands is used, it shall not be an error to specify *first* or *last* values that are not in the history list; *fc* shall substitute the value representing the oldest or newest command in the list, as appropriate. For example, if there are only ten commands in the history list, numbered 1 to 10:

97557

97558            fc -l

97559            fc 1 99

97560

97561

97562            shall list and edit, respectively, all ten commands.

97563            *old=new*        Replace the first occurrence of string *old* in the commands to be re-executed by the string *new*.

97564

97565 **STDIN**

97566 Not used.

97567 **INPUT FILES**

97568 None.

97569 **ENVIRONMENT VARIABLES**97570 The following environment variables shall affect the execution of *fc*:

97571 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for  
 97572 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be  
 97573 used as the editor.

97574 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is  
 97575 not set, the shell may attempt to access or create a file *.sh\_history* in the directory  
 97576 referred to by the *HOME* environment variable. If the shell cannot obtain both read  
 97577 and write access to, or create, the history file, it shall use an unspecified  
 97578 mechanism that allows the history to operate properly. (References to history ``file''  
 97579 in this section shall be understood to mean this unspecified mechanism in such  
 97580 cases.) An implementation may choose to access this variable only when  
 97581 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt  
 97582 to retrieve entries from, or add entries to, the file, as the result of commands issued  
 97583 by the user, the file named by the *ENV* variable, or implementation-defined system  
 97584 start-up files. In some historical shells, the history file is initialized just after the  
 97585 *ENV* file has been processed. Therefore, it is implementation-defined whether  
 97586 changes made to *HISTFILE* after the history file has been initialized are effective.  
 97587 Implementations may choose to disable the history list mechanism for users with  
 97588 appropriate privileges who do not set *HISTFILE*; the specific circumstances under  
 97589 which this occurs are implementation-defined. If more than one instance of the  
 97590 shell is using the same history file, it is unspecified how updates to the history file  
 97591 from those shells interact. As entries are deleted from the history file, they shall be  
 97592 deleted oldest first. It is unspecified when history file entries are physically  
 97593 removed from the history file.

97594 *HISTSIZE* Determine a decimal number representing the limit to the number of previous  
 97595 commands that are accessible. If this variable is unset, an unspecified default  
 97596 greater than or equal to 128 shall be used. The maximum number of commands in  
 97597 the history list is unspecified, but shall be at least 128. An implementation may  
 97598 choose to access this variable only when initializing the history file, as described  
 97599 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*  
 97600 after the history file has been initialized are effective.

97601 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 97602 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 97603 variables used to determine the values of locale categories.)

97604 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 97605 internationalization variables.

97606 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 97607 characters (for example, single-byte as opposed to multi-byte characters in  
 97608 arguments and input files).

97609 *LC\_MESSAGES*

97610 Determine the locale that should be used to affect the format and contents of  
 97611 diagnostic messages written to standard error.

- 97612 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 97613 **ASYNCHRONOUS EVENTS**
- 97614 Default.
- 97615 **STDOUT**
- 97616 When the `-l` option is used to list commands, the format of each command in the list shall be as
- 97617 follows:
- 97618 `"%d\t%s\n", <line number>, <command>`
- 97619 If both the `-l` and `-n` options are specified, the format of each command shall be:
- 97620 `"\t%s\n", <command>`
- 97621 If the `<command>` consists of more than one line, the lines after the first shall be displayed as:
- 97622 `"\t%s\n", <continued-command>`
- 97623 **STDERR**
- 97624 The standard error shall be used only for diagnostic messages.
- 97625 **OUTPUT FILES**
- 97626 None.
- 97627 **EXTENDED DESCRIPTION**
- 97628 None.
- 97629 **EXIT STATUS**
- 97630 The following exit values shall be returned:
- 97631 0 Successful completion of the listing.
- 97632 >0 An error occurred.
- 97633 Otherwise, the exit status shall be that of the commands executed by `fc`.
- 97634 **CONSEQUENCES OF ERRORS**
- 97635 Default.
- 97636 **APPLICATION USAGE**
- 97637 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.
- 97638 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file
- 97639 descriptors as part of the `fc` command can produce unexpected results. For example, if `vi` is the
- 97640 `FCEDIT` editor, the command:
- 97641 `fc -s | more`
- 97642 does not work correctly on many systems.
- 97643 Users on windowing systems may want to have separate history files for each window by
- 97644 setting `HISTFILE` as follows:
- 97645 `HISTFILE=$HOME/.sh_hist$$`
- 97646 **EXAMPLES**
- 97647 None.
- 97648 **RATIONALE**
- 97649 This utility is based on the `fc` built-in of the KornShell.
- 97650 An early proposal specified the `-e` option as `[-e editor [old= new ]]`, which is not historical
- 97651 practice. Historical practice in `fc` of either `[-e editor]` or `[-e - [ old= new ]]` is acceptable, but not

97652 both together. To clarify this, a new option `-s` was introduced replacing the `[-e -]`. This resolves  
 97653 the conflict and makes `fc` conform to the Utility Syntax Guidelines.

97654 **HISTFILE** Some implementations of the KornShell check for the superuser and do not create  
 97655 a history file unless `HISTFILE` is set. This is done primarily to avoid creating  
 97656 unlinked files in the root file system when logging in during single-user mode.  
 97657 `HISTFILE` must be set for the superuser to have history.

97658 **HISTSIZE** Needed to limit the size of history files. It is the intent of the standard developers  
 97659 that when two shells share the same history file, commands that are entered in one  
 97660 shell shall be accessible by the other shell. Because of the difficulties of  
 97661 synchronization over a network, the exact nature of the interaction is unspecified.

97662 The initialization process for the history file can be dependent on the system start-up files, in  
 97663 that they may contain commands that effectively preempt the settings the user has for `HISTFILE`  
 97664 and `HISTSIZE`. For example, function definition commands are recorded in the history file. If  
 97665 the system administrator includes function definitions in some system start-up file called before  
 97666 the `ENV` file, the history file is initialized before the user can influence its characteristics. In some  
 97667 historical shells, the history file is initialized just after the `ENV` file has been processed. Because  
 97668 of these situations, the text requires the initialization process to be implementation-defined.

97669 Consideration was given to omitting the `fc` utility in favor of the command line editing feature in  
 97670 `sh`. For example, in `vi` editing mode, typing "`<ESC> v`" is equivalent to:

```
97671 EDITOR=vi fc
```

97672 However, the `fc` utility allows the user the flexibility to edit multiple commands simultaneously  
 97673 (such as `fc 10 20`) and to use editors other than those supported by `sh` for command line editing.

97674 In the KornShell, the alias `r` (``re-do``) is preset to `fc -e -` (equivalent to the POSIX `fc -s`). This is  
 97675 probably an easier command name to remember than `fc` (``fix command``), but it does not meet  
 97676 the Utility Syntax Guidelines. Renaming `fc` to `hist` or `redo` was considered, but since this  
 97677 description closely matches historical KornShell practice already, such a renaming was seen as  
 97678 gratuitous. Users are free to create aliases whenever odd historical names such as `fc`, `awk`, `cat`,  
 97679 `grep`, or `yacc` are standardized by POSIX.

97680 Command numbers have no ordering effects; they are like serial numbers. The `-r` option and  
 97681 `-number` operand address the sequence of command execution, regardless of serial numbers. So,  
 97682 for example, if the command number wrapped back to 1 at some arbitrary point, there would be  
 97683 no ambiguity associated with traversing the wrap point. For example, if the command history  
 97684 were:

```
97685 32766: echo 1
97686 32767: echo 2
97687 1: echo 3
```

97688 the number `-2` refers to command 32767 because it is the second previous command, regardless  
 97689 of serial number.

## 97690 FUTURE DIRECTIONS

97691 None.

## 97692 SEE ALSO

97693 [`sh`](#)

97694 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**97695 CHANGE HISTORY**

97696 First released in Issue 4.

**97697 Issue 5**

97698 The FUTURE DIRECTIONS section is added.

**97699 Issue 6**

97700 This utility is marked as part of the User Portability Utilities option.

97701 In the ENVIRONMENT VARIABLES section, the text ``user's home directory'' is updated to  
97702 ``directory referred to by the *HOME* environment variable''.

**97703 Issue 7**

97704 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**97705 Issue 8**

97706 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
97707 this utility is required to be intrinsic.

97708 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97709 **NAME**

97710 fg — run jobs in the foreground

97711 **SYNOPSIS**97712 UP fg [*job\_id*]97713 **DESCRIPTION**

97714 If job control is enabled (see the description of *set -m*), the shell is interactive, and the current  
 97715 shell execution environment (see [Section 2.13](#), on page 2522) is not a subshell environment, the *fg*  
 97716 utility shall move a background job in the current execution environment into the foreground, as  
 97717 described in [Section 2.11](#) (on page 2518); it may also do so if the shell is non-interactive or the  
 97718 current shell execution environment is a subshell environment.

97719 Using *fg* to place a job into the foreground shall remove its process ID from the list of those  
 97720 “known in the current shell execution environment”; see [Section 2.9.3.1](#) (on page 2506).

97721 **OPTIONS**

97722 None.

97723 **OPERANDS**

97724 The following operand shall be supported:

97725 *job\_id* Specify the job to be run as a foreground job. If no *job\_id* operand is given, the  
 97726 *job\_id* for the job that was most recently suspended, placed in the background, or  
 97727 run as a background job shall be used. The format of *job\_id* is described in XBD  
 97728 [Section 3.182](#) (on page 57).

97729 **STDIN**

97730 Not used.

97731 **INPUT FILES**

97732 None.

97733 **ENVIRONMENT VARIABLES**97734 The following environment variables shall affect the execution of *fg*:

97735 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 97736 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 97737 variables used to determine the values of locale categories.)

97738 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 97739 internationalization variables.

97740 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 97741 characters (for example, single-byte as opposed to multi-byte characters in  
 97742 arguments).

97743 *LC\_MESSAGES*

97744 Determine the locale that should be used to affect the format and contents of  
 97745 diagnostic messages written to standard error.

97746 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

97747 **ASYNCHRONOUS EVENTS**

97748 Default.



**97749 STDOUT**

97750 The *fg* utility shall write the command line of the job to standard output in the following format:

97751 "%s\n", <command>

**97752 STDERR**

97753 The standard error shall be used only for diagnostic messages.

**97754 OUTPUT FILES**

97755 None.

**97756 EXTENDED DESCRIPTION**

97757 None.

**97758 EXIT STATUS**

97759 If the *fg* utility succeeds, it does not return an exit status. Instead, the shell waits for the job that  
97760 *fg* moved into the foreground.

97761 If *fg* does not move a job into the foreground, the following exit value shall be returned:

97762 >0 An error occurred.

**97763 CONSEQUENCES OF ERRORS**

97764 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the  
97765 foreground.

**97766 APPLICATION USAGE**

97767 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

97768 The *fg* utility does not work as expected when it is operating in its own utility execution  
97769 environment because that environment has no applicable jobs to manipulate. See the  
97770 APPLICATION USAGE section for *bg*. For this reason, *fg* is generally implemented as a shell  
97771 regular built-in.

**97772 EXAMPLES**

97773 None.

**97774 RATIONALE**

97775 The extensions to the shell specified in this volume of POSIX.1-2024 have mostly been based on  
97776 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also  
97777 based on the KornShell. The standard developers examined the characteristics of the C shell  
97778 versions of these utilities and found that differences exist. Despite widespread use of the C shell,  
97779 the KornShell versions were selected for this volume of POSIX.1-2024 to maintain a degree of  
97780 uniformity with the rest of the KornShell features selected (such as the very popular command  
97781 line editing features).

**97782 FUTURE DIRECTIONS**

97783 None.

**97784 SEE ALSO**

97785 [Section 2.9.3.1](#) (on page 2506), [Section 2.13](#) (on page 2522), *bg*, *kill*, *jobs*, *wait*

97786 XBD [Section 3.182](#) (on page 57), [Chapter 8](#) (on page 167)

**97787 CHANGE HISTORY**

97788 First released in Issue 4.

97789 **Issue 6**

97790 This utility is marked as part of the User Portability Utilities option.

97791 The APPLICATION USAGE section is added.

97792 The JC marking is removed from the SYNOPSIS since job control is mandatory in this version.

97793 **Issue 8**

97794 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
97795 this utility is required to be intrinsic.

97796 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

97797 Austin Group Defect 1254 is applied, updating the DESCRIPTION to account for the addition of  
97798 [Section 2.11](#) (on page 2518) and changing the EXIT STATUS section.

97799 **NAME**

97800 file — determine file type

97801 **SYNOPSIS**97802 file [-dh] [-M *file*] [-m *file*] *file*...97803 file -i [-h] *file*...97804 **DESCRIPTION**97805 The *file* utility shall perform a series of tests in sequence on each specified *file* in an attempt to  
97806 classify it:

- 97807 1. If *file* does not exist, cannot be read, or its file status could not be determined, the output  
97808 shall indicate that the file was processed, but that its type could not be determined.
- 97809 2. If the file is not a regular file, its file type shall be identified. The file types directory,  
97810 FIFO, socket, block special, and character special shall be identified as such. Other  
97811 implementation-defined file types may also be identified. If *file* is a symbolic link, by  
97812 default the link shall be resolved and *file* shall test the type of file referenced by the  
97813 symbolic link. (See the **-h** and **-i** options below.)
- 97814 3. If the length of *file* is zero, it shall be identified as an empty file.
- 97815 4. The *file* utility shall examine an initial segment of *file* and shall make a guess at  
97816 identifying its contents based on position-sensitive tests. (The answer is not guaranteed to  
97817 be correct; see the **-d**, **-M**, and **-m** options below.)
- 97818 5. The *file* utility shall examine *file* and make a guess at identifying its contents based on  
97819 context-sensitive default system tests. (The answer is not guaranteed to be correct.)
- 97820 6. The file shall be identified as a data file.

97821 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall  
97822 indicate that the file was processed, but that its type could not be determined.97823 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file  
97824 referenced by the symbolic link.97825 **OPTIONS**97826 The *file* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the **-m**,  
97827 **-d**, and **-M** options shall be significant.

97828 The following options shall be supported by the implementation:

- 97829 **-d** Apply any position-sensitive default system tests and context-sensitive default  
97830 system tests to the file. This is the default if no **-M** or **-m** option is specified.
- 97831 **-h** When a symbolic link is encountered, identify the file as a symbolic link. If **-h** is  
97832 not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall  
97833 identify the file as a symbolic link, as if **-h** had been specified.
- 97834 **-i** If a file is a regular file, do not attempt to classify the type of the file further, but  
97835 identify the file as specified in the **STDOUT** section.
- 97836 **-M *file*** Specify the name of a file containing position-sensitive tests that shall be applied to  
97837 a file in order to classify it (see the **EXTENDED DESCRIPTION**). No position-  
97838 sensitive default system tests nor context-sensitive default system tests shall be  
97839 applied unless the **-d** option is also specified.

97840 **-m file** Specify the name of a file containing position-sensitive tests that shall be applied to  
 97841 a file in order to classify it (see the EXTENDED DESCRIPTION).

97842 If the **-m** option is specified without specifying the **-d** option or the **-M** option, position-  
 97843 sensitive default system tests shall be applied after the position-sensitive tests specified by the  
 97844 **-m** option. If the **-M** option is specified with the **-d** option, the **-m** option, or both, or the **-m**  
 97845 option is specified with the **-d** option, the concatenation of the position-sensitive tests specified  
 97846 by these options shall be applied in the order specified by the appearance of these options. If a  
 97847 **-M** or **-m file** option-argument is **-**, the results are unspecified.

#### 97848 OPERANDS

97849 The following operand shall be supported:

97850 *file* A pathname of a file to be tested.

#### 97851 STDIN

97852 The standard input shall be used if a *file* operand is **'-'** and the implementation treats the **'-'**  
 97853 as meaning standard input. Otherwise, the standard input shall not be used.

#### 97854 INPUT FILES

97855 The *file* can be any file type.

#### 97856 ENVIRONMENT VARIABLES

97857 The following environment variables shall affect the execution of *file*:

97858 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 97859 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 97860 variables used to determine the values of locale categories.)

97861 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 97862 internationalization variables.

97863 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 97864 characters (for example, single-byte as opposed to multi-byte characters in  
 97865 arguments and input files).

97866 *LC\_MESSAGES*

97867 Determine the locale that should be used to affect the format and contents of  
 97868 diagnostic messages written to standard error and informative messages written to  
 97869 standard output.

97870 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

#### 97871 ASYNCHRONOUS EVENTS

97872 Default.

#### 97873 STDOUT

97874 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

97875 "%s: %s\n", *<file>*, *<type>*

97876 The values for *<type>* are unspecified, except that in the POSIX locale, if *file* is identified as one  
 97877 of the types listed in the following table, *<type>* shall contain (but is not limited to) the  
 97878 corresponding string, unless the file is identified by a position-sensitive test specified by a **-M** or  
 97879 **-m** option. Each *<space>* shown in the strings shall be exactly one *<space>*.

97880

Table 3-10 File Utility Output Strings

	If <i>file</i> is:	< <i>type</i> > shall contain the string:	Notes
97881	Nonexistent	cannot open	
97882	Block special	block special	1
97883	Character special	character special	1
97884	Directory	directory	1
97885	FIFO	fifo	1
97886	Socket	socket	1
97887	Symbolic link	symbolic link to	1
97888	Regular file	regular file	1,2
97889	Empty regular file	empty	3
97890	Regular file that cannot be read	cannot open	3
97891	Executable binary	executable	3,4,6
97892	<i>ar</i> archive library (see <i>ar</i> )	archive	3,4,6
97893	Extended <i>cpio</i> format (see <i>pax</i> )	cpio archive	3,4,6
97894	Extended <i>tar</i> format (see <b>ustar</b> in <i>pax</i> )	tar archive	3,4,6
97895	Shell script	commands text	3,5,6
97896	C-language source	c program text	3,5,6
97897	FORTRAN source	fortran program text	3,5,6
97898	Regular file whose type cannot be determined	data	3
97899			

97900

**Notes:**

97901

1. This is a file type test.

97902

2. This test is applied only if the **-i** option is specified.

97903

3. This test is applied only if the **-i** option is not specified.

97904

4. This is a position-sensitive default system test.

97905

5. This is a context-sensitive default system test.

97906

6. Position-sensitive default system tests and context-sensitive default system tests are not applied if the **-M** option is specified unless the **-d** option is also specified.

97907

97908

In the POSIX locale, if *file* is identified as a symbolic link (see the **-h** option), the following alternative output format shall be used:

97909

97910

```
"%s: %s %s\n", <file>, <type>, <contents of link>"
```

97911

If the file named by the *file* operand does not exist, cannot be read, or the type of the file named by the *file* operand cannot be determined, this shall not be considered an error that affects the exit status.

97912

97913

97914

**STDERR**

97915

The standard error shall be used only for diagnostic messages.

97916

**OUTPUT FILES**

97917

None.

## 97918 EXTENDED DESCRIPTION

97919 A file specified as an option-argument to the `-m` or `-M` options shall contain one position-  
 97920 sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of  
 97921 the line shall be printed and no further tests shall be applied, with the exception that tests on  
 97922 immediately following lines beginning with a single `'>'` character shall be applied.

97923 Each line shall be composed of the following four `<tab>`-separated fields. (Implementations may  
 97924 allow any combination of one or more white-space characters other than `<newline>` to act as  
 97925 field separators.)

97926 *offset* An unsigned number (optionally preceded by a single `'>'` character) specifying  
 97927 the *offset*, in bytes, of the value in the file that is to be compared against the *value*  
 97928 field of the line. If the file is shorter than the specified offset, the test shall fail.

97929 If the *offset* begins with the character `'>'`, the test contained in the line shall not be  
 97930 applied to the file unless the test on the last line for which the *offset* did not begin  
 97931 with a `'>'` was successful. By default, the *offset* shall be interpreted as an unsigned  
 97932 decimal number. With a leading `0x` or `0X`, the *offset* shall be interpreted as a  
 97933 hexadecimal number; otherwise, with a leading `0`, the *offset* shall be interpreted as  
 97934 an octal number.

97935 *type* The type of the value in the file to be tested. The type shall consist of the type  
 97936 specification characters `d`, `s`, and `u`, specifying signed decimal, string, and  
 97937 unsigned decimal, respectively.

97938 The *type* string shall be interpreted as the bytes from the file starting at the  
 97939 specified *offset* and including the same number of bytes specified by the *value* field.  
 97940 If insufficient bytes remain in the file past the *offset* to match the *value* field, the test  
 97941 shall fail.

97942 The type specification characters `d` and `u` can be followed by an optional unsigned  
 97943 decimal integer that specifies the number of bytes represented by the type. The  
 97944 type specification characters `d` and `u` can be followed by an optional `C`, `S`, `I`, or `L`,  
 97945 indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

97946 The default number of bytes represented by the type specifiers `d`, `f`, and `u` shall  
 97947 correspond to their respective C-language types as follows. If the system claims  
 97948 conformance to the C-Language Development Utilities option, those specifiers  
 97949 shall correspond to the default sizes used in the `c17` utility. Otherwise, the default  
 97950 sizes shall be implementation-defined.

97951 For the type specifier characters `d` and `u`, the default number of bytes shall  
 97952 correspond to the size of a basic integer type of the implementation. For these  
 97953 specifier characters, the implementation shall support values of the optional  
 97954 number of bytes to be converted corresponding to the number of bytes in the C-  
 97955 language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an  
 97956 application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when  
 97957 interpreting numeric values is implementation-defined, but shall correspond to the  
 97958 order in which a constant of the corresponding type is stored in memory on the  
 97959 system.

97960 All type specifiers, except for `s`, can be followed by a mask specifier of the form  
 97961 `&number`. The mask value shall be AND'ed with the value of the input file before  
 97962 the comparison with the *value* field of the line is made. By default, the mask shall  
 97963 be interpreted as an unsigned decimal number. With a leading `0x` or `0X`, the mask  
 97964 shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading

97965		0, the mask shall be interpreted as an unsigned octal number.
97966		The strings <b>byte</b> , <b>short</b> , <b>long</b> , and <b>string</b> shall also be supported as type fields,
97967		being interpreted as dC, dS, dL, and s, respectively.
97968	<i>value</i>	The <i>value</i> to be compared with the value from the file.
97969		If the specifier from the type field is s or <b>string</b> , then interpret the value as a string.
97970		Otherwise, interpret it as a number. If the value is a string, then the test shall
97971		succeed only when a string value exactly matches the bytes from the file.
97972		If the <i>value</i> is a string, it can contain the following sequences:
97973	<i>\character</i>	The <backslash>-escape sequences as specified in XBD Table 5-1
97974		(on page 113) ('\a', '\b', '\f', '\n', '\r', '\t',
97975		'\v'). In addition, the escape sequence '\ ' (the <backslash>
97976		character followed by a <space> character) shall be recognized to
97977		represent a <space> character. The results of using any other
97978		character, other than an octal digit, following the <backslash>
97979		are unspecified.
97980	<i>\octal</i>	Octal sequences that can be used to represent characters with
97981		specific coded values. An octal sequence shall consist of a
97982		<backslash> followed by the longest sequence of one, two, or
97983		three octal-digit characters (01234567).
97984		By default, any value that is not a string shall be interpreted as a signed decimal
97985		number. Any such value, with a leading 0x or 0X, shall be interpreted as an
97986		unsigned hexadecimal number; otherwise, with a leading zero, the value shall be
97987		interpreted as an unsigned octal number.
97988		If the value is not a string, it can be preceded by a character indicating the
97989		comparison to be performed. Permissible characters and the comparisons they
97990		specify are as follows:
97991	=	The test shall succeed if the value from the file equals the <i>value</i> field.
97992	<	The test shall succeed if the value from the file is less than the <i>value</i> field.
97993	>	The test shall succeed if the value from the file is greater than the <i>value</i> field.
97994	&	The test shall succeed if all of the set bits in the <i>value</i> field are set in the value
97995		from the file.
97996	^	The test shall succeed if at least one of the set bits in the <i>value</i> field is not set in
97997		the value from the file.
97998	x	The test shall succeed if the file is large enough to contain a value of the type
97999		specified starting at the offset specified.
98000	<i>message</i>	The <i>message</i> to be printed if the test succeeds. The <i>message</i> shall be interpreted
98001		using the notation for the <i>printf</i> formatting specification; see <i>printf</i> . If the <i>value</i>
98002		field was a string, then the value from the file shall be the argument for the <i>printf</i>
98003		formatting specification; otherwise, the value from the file shall be the argument.

98004 **EXIT STATUS**

98005 The following exit values shall be returned:

98006 0 Successful completion.

98007 &gt;0 An error occurred.

98008 **CONSEQUENCES OF ERRORS**

98009 Default.

98010 **APPLICATION USAGE**98011 The *file* utility can only be required to guess at many of the file types because only exhaustive  
98012 testing can determine some types with certainty. For example, binary data on some  
98013 implementations might match the initial segment of an executable or a *tar* archive.98014 Note that the table indicates that the output contains the stated string. Systems may add text  
98015 before or after the string. For executables, as an example, the machine architecture and various  
98016 facts about how the file was link-edited may be included. Note also that on systems that  
98017 recognize shell script files starting with "#!" as executable files, these may be identified as  
98018 executable binary files rather than as shell scripts.98019 **EXAMPLES**

98020 Determine whether an argument is a binary executable file:

98021 

```
file -- "$1" | grep -q '.*executable' &&  
98022     printf "%s is executable.\n" "$1"
```

98023 **RATIONALE**98024 The *-f* option was omitted because the same effect can (and should) be obtained using the *xargs*  
98025 utility.98026 Historical versions of the *file* utility attempt to identify the following types of files: symbolic link,  
98027 directory, character special, block special, socket, *tar* archive, *cpio* archive, SCCS archive, archive  
98028 library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler  
98029 source, *nroff/troff/eqn/tbl* source *troff* output, shell script, C shell script, English text, ASCII text,  
98030 various executables, APL workspace, compiled terminfo entries, and CURSES screen images.  
98031 Only those types that are reasonably well specified in POSIX or are directly related to POSIX  
98032 utilities are listed in the table.98033 Historical systems have used a "magic file" named */etc/magic* to help identify file types. Because  
98034 it is generally useful for users and scripts to be able to identify special file types, the *-m* flag and  
98035 a portable format for user-created magic files has been specified. No requirement is made that an  
98036 implementation of *file* use this method of identifying files, only that users be permitted to add  
98037 their own classifying tests.98038 In addition, three options have been added to historical practice. The *-d* flag has been added to  
98039 permit users to cause their tests to follow any default system tests. The *-i* flag has been added to  
98040 permit users to test portably for regular files in shell scripts. The *-M* flag has been added to  
98041 permit users to ignore any default system tests.98042 The POSIX.1-2024 description of default system tests and the interaction between the *-d*, *-M*,  
98043 and *-m* options did not clearly indicate that there were two types of "default system tests". The  
98044 "position-sensitive tests" determine file types by looking for certain string or binary values at  
98045 specific offsets in the file being examined. These position-sensitive tests were implemented in  
98046 historical systems using the magic file described above. Some of these tests are now built into  
98047 the *file* utility itself on some implementations so the output can provide more detail than can be  
98048 provided by magic files. For example, a magic file can easily identify a file containing a core  
98049 image on most implementations, but cannot name the program file that dropped the core. A



98050 magic file could produce output such as:

98051 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1

98052 but by building the test into the *file* utility, you could get output such as:

98053 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'

98054 These extended built-in tests are still to be treated as position-sensitive default system tests even  
98055 if they are not listed in */etc/magic* or any other magic file.

98056 The context-sensitive default system tests were always built into the *file* utility. These tests  
98057 looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and  
98058 other computer language source files, and even plain text files. With the addition of the *-m* and  
98059 *-M* options the distinction between position-sensitive and context-sensitive default system tests  
98060 became important because the order of testing is important. The context-sensitive system default  
98061 tests should never be applied before any position-sensitive tests even if the *-d* option is specified  
98062 before a *-m* option or *-M* option due to the high probability that the context-sensitive system  
98063 default tests will incorrectly identify arbitrary text files as text files before position-sensitive tests  
98064 specified by the *-m* or *-M* option would be applied to give a more accurate identification.

98065 Leaving the meaning of *-M* – and *-m* – unspecified allows an existing prototype of these  
98066 options to continue to work in a backwards-compatible manner. (In that implementation, *-M* –  
98067 was roughly equivalent to *-d* in POSIX.1-2024.)

98068 The historical *-c* option was omitted as not particularly useful to users or portable shell scripts.  
98069 In addition, a reasonable implementation of the *file* utility would report any errors found each  
98070 time the magic file is read.

98071 The historical format of the magic file was the same as that specified by the Rationale in the  
98072 ISO POSIX-2:1993 standard for the *offset*, *value*, and *message* fields; however, it used less precise  
98073 type fields than the format specified by the current normative text. The new type field values are  
98074 a superset of the historical ones.

98075 The following is an example magic file:

```

98076 0 short      070707          cpio archive
98077 0 short      0143561        Byte-swapped cpio archive
98078 0 string     070707          ASCII cpio archive
98079 0 long       0177555        Very old archive
98080 0 short      0177545        Old archive
98081 0 short      017437         Old packed data
98082 0 string     \037\036       Packed data
98083 0 string     \377\037       Compacted data
98084 0 string     \037\235       Compressed data
98085 >2 byte&0x80 >0          Block compressed
98086 >2 byte&0x1f x           %d bits
98087 0 string     \032\001       Compiled Terminfo Entry
98088 0 short      0433           Curses screen image
98089 0 short      0434           Curses screen image
98090 0 string     <ar>           System V Release 1 archive
98091 0 string     !<arch>\n___.SYMDEF Archive random library
98092 0 string     !<arch>        Archive
98093 0 string     ARF_BEGARF     PHIGS clear text archive
98094 0 long       0x137A2950     Scalable OpenFont binary
98095 0 long       0x137A2951     Encrypted scalable OpenFont binary

```

98096 The use of a basic integer data type is intended to allow the implementation to choose a word  
98097 size commonly used by applications on that architecture.

98098 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
98099 but this has been modified in this version.

#### 98100 FUTURE DIRECTIONS

98101 If this utility is directed to display a pathname that contains any bytes that have the encoded  
98102 value of a <newline> character when <newline> is a terminator or separator in the output  
98103 format being used, implementations are encouraged to treat this as an error. A future version of  
98104 this standard may require implementations to treat this as an error.

#### 98105 SEE ALSO

98106 *ar, ls, pax, printf*

98107 XBD [Table 5-1](#) (on page 113), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 98108 CHANGE HISTORY

98109 First released in Issue 4.

#### 98110 Issue 6

98111 This utility is marked as part of the User Portability Utilities option.

98112 Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft  
98113 standard.

98114 IEEE PASC Interpretations 1003.2 #192 and #178 are applied.

98115 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/25 is applied, making major changes to  
98116 address ambiguities raised in defect reports.

98117 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/26 is applied, making it clear in the  
98118 OPTIONS section that the **-m**, **-d**, and **-M** options do not comply with Guideline 11 of the  
98119 Utility Syntax Guidelines.

98120 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/10 is applied, clarifying the specification  
98121 characters.

98122 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/11 is applied, allowing application  
98123 developers to create portable magic files that can match characters in strings, and allowing  
98124 common extensions found in existing implementations.

98125 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/12 is applied, removing text describing  
98126 behavior on systems with bytes consisting of more than eight bits.

#### 98127 Issue 7

98128 Austin Group Interpretation 1003.1-2001 #092 is applied.

98129 SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in [Table 3-10](#).

98130 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

98131 The *file* utility is moved from the User Portability Utilities option to the Base. User Portability  
98132 Utilities is now an option for interactive utilities.

98133 The EXAMPLES section is revised to correct an error with the pathname "\$1".

#### 98134 Issue 8

98135 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
98136 directed to display a pathname that contains any bytes that have the encoded value of a  
98137 <newline> character when <newline> is a terminator or separator in the output format being

98138

used.

98139

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

98140

Austin Group Defect 1141 is applied, changing ``core file'' to ``file containing a core image''.

98141 **NAME**

98142 find — find files

98143 **SYNOPSIS**98144 find [-H|-L] *path...* [*operand\_expression...*]98145 **DESCRIPTION**

98146 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,  
 98147 evaluating a Boolean expression composed of the primaries described in the OPERANDS section  
 98148 for each file encountered. Each *path* operand shall be evaluated unaltered as it was provided,  
 98149 including all trailing <slash> characters; all pathnames for other files encountered in the  
 98150 hierarchy shall consist of the concatenation of the current *path* operand, a <slash> if the current  
 98151 *path* operand did not end in one, and the filename relative to the *path* operand. The relative  
 98152 portion shall contain no dot or dot-dot components, no trailing <slash> characters, and only  
 98153 single <slash> characters between pathname components.

98154 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail  
 98155 due to path length limitations (unless a *path* operand specified by the application exceeds  
 98156 {PATH\_MAX} requirements).

98157 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 98158 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a  
 98159 diagnostic message to standard error and shall either recover its position in the hierarchy or  
 98160 terminate. In either case, the final exit status shall be non-zero.

98161 If a file is removed from or added to the directory hierarchy being searched it is unspecified  
 98162 whether or not *find* includes that file in its search.

98163 **OPTIONS**98164 The *find* utility shall conform to XBD [Section 12.2](#) (on page 215).

98165 The following options shall be supported by the implementation:

98166 **-H** Cause the file information and file type evaluated for each symbolic link  
 98167 encountered as a *path* operand on the command line to be those of the file  
 98168 referenced by the link, and not the link itself. If the referenced file does not exist,  
 98169 the file information and type shall be for the link itself. File information and type  
 98170 for symbolic links encountered during the traversal of a file hierarchy shall be that  
 98171 of the link itself.

98172 **-L** Cause the file information and file type evaluated for each symbolic link  
 98173 encountered as a *path* operand on the command line or encountered during the  
 98174 traversal of a file hierarchy to be those of the file referenced by the link, and not the  
 98175 link itself. If the referenced file does not exist, the file information and type shall be  
 98176 for the link itself.

98177 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
 98178 an error. The last option specified shall determine the behavior of the utility. If neither the **-H**  
 98179 nor the **-L** option is specified, then the file information and type for symbolic links encountered  
 98180 as a *path* operand on the command line or encountered during the traversal of a file hierarchy  
 98181 shall be that of the link itself.

98182 **OPERANDS**

98183 The following operands shall be supported:

98184 The first operand and subsequent operands up to but not including the first operand that starts  
 98185 with a '-', or is a '!' or a '(', shall be interpreted as *path* operands. If the first operand starts  
 98186 with a '-', or is a '!' or a '(', the behavior is unspecified. Each *path* operand is a pathname of

- 98187 a starting point in the file hierarchy.
- 98188 The first operand that starts with a '-', or is a '!' or a '(' , and all subsequent arguments shall  
 98189 be interpreted as an *expression* made up of the following primaries and operators. In the  
 98190 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal  
 98191 integer optionally preceded by a <plus-sign> ('+') or <hyphen-minus> ('-'), as follows:
- 98192 **+n** More than *n*.
- 98193 **n** Exactly *n*.
- 98194 **-n** Less than *n*.
- 98195 The following primaries shall be supported:
- 98196 **-name pattern**  
 98197 The primary shall evaluate as true if the basename of the current pathname  
 98198 matches *pattern* using the pattern matching notation described in [Section 2.14](#) (on  
 98199 page 2523). The additional rules in [Section 2.14.3](#) (on page 2525) do not apply as  
 98200 this is a matching operation, not an expansion.
- 98201 **-iname pattern**  
 98202 The **-iname** primary shall be equivalent to **-name**, except that the match shall be  
 98203 case insensitive. See [XBD Section 4.1](#) (on page 95).
- 98204 **-path pattern**  
 98205 The primary shall evaluate as true if the current pathname matches *pattern* using  
 98206 the pattern matching notation described in [Section 2.14](#) (on page 2523). The  
 98207 additional rules in [Section 2.14.3](#) (on page 2525) do not apply as this is a matching  
 98208 operation, not an expansion.
- 98209 **-nouser** The primary shall evaluate as true if the file belongs to a user ID for which the  
 98210 *getpwuid()* function defined in the System Interfaces volume of POSIX.1-2024 (or  
 98211 equivalent) returns NULL.
- 98212 **-nogroup** The primary shall evaluate as true if the file belongs to a group ID for which the  
 98213 *getgrgid()* function defined in the System Interfaces volume of POSIX.1-2024 (or  
 98214 equivalent) returns NULL.
- 98215 **-mount** The primary shall always evaluate as true; it shall cause *find* to act only on files that  
 98216 have the same device ID (*st\_dev*, see [XSH fstatat\(\)](#)) as the *path* operand below  
 98217 which they are encountered and cause *find* not to descend below directories that  
 98218 have a different device ID than that *path* operand. If any **-mount** primary is  
 98219 specified, it shall apply to the entire expression even if the **-mount** primary would  
 98220 not normally be evaluated.
- 98221 **-xdev** The primary shall always evaluate as true; it shall cause *find* not to descend below  
 98222 directories that have a different device ID (*st\_dev*, see [XSH fstatat\(\)](#)) than the *path*  
 98223 operand below which they are encountered; that is, when a directory with a  
 98224 different device ID is encountered, *find* shall act on the directory itself (unless  
 98225 **-mount** is specified) but shall not act on any files below the directory. If any **-xdev**  
 98226 primary is specified, it shall apply to the entire expression even if the **-xdev**  
 98227 primary would not normally be evaluated.
- 98228 **-prune** The primary shall always evaluate as true; it shall cause *find* not to descend the  
 98229 current pathname if it is a directory. If the **-depth** primary is specified, the **-prune**  
 98230 primary shall have no effect.

98231 **-perm [-]mode**  
 98232 The *mode* argument is used to represent file mode bits. It shall be processed in an  
 98233 identical manner to the *symbolic\_mode* operand described in *chmod*, except that:

- 98234 1. The changes to file mode bits shall be applied to a template instead of to any  
 98235 files. The template shall initially have all file mode bits cleared.
- 98236 2. The *op* symbol '-' cannot be the first character of *mode*; this avoids  
 98237 ambiguity with the optional leading <hyphen-minus>. Since the initial  
 98238 mode is all bits off, there are not any symbolic modes that need to use '-'  
 98239 as the first character.

98240 If the <hyphen-minus> is omitted, the primary shall evaluate as true when the file  
 98241 permission bits exactly match the value of the resulting template.

98242 Otherwise, if *mode* is prefixed by a <hyphen-minus>, the primary shall evaluate as  
 98243 true if at least all the bits in the resulting template are set in the file permission bits.

98244 **-perm [-]onum**  
 98245 If the <hyphen-minus> is omitted, the primary shall evaluate as true when the file  
 98246 mode bits exactly match the value of the octal number *onum* (see the description of  
 98247 the octal *mode* in *chmod*). Otherwise, if *onum* is prefixed by a <hyphen-minus>, the  
 98248 primary shall evaluate as true if at least all of the bits specified in *onum* are set. In  
 98249 both cases, the behavior is unspecified when *onum* exceeds 07777.

98250 **-type c** The primary shall evaluate as true if the type of the file is *c*, where *c* is 'b', 'c',  
 98251 'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory,  
 98252 symbolic link, FIFO, regular file, or socket, respectively.

98253 **-links n** The primary shall evaluate as true if the file has *n* links.

98254 **-user uname** The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is  
 98255 a decimal integer and the *getpwnam()* (or equivalent) function does not return a  
 98256 valid user name, *uname* shall be interpreted as a user ID.

98257 **-group gname**  
 98258 The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*  
 98259 is a decimal integer and the *getgrnam()* (or equivalent) function does not return a  
 98260 valid group name, *gname* shall be interpreted as a group ID.

98261 **-size n[c]** The primary shall evaluate as true if the file size in bytes, divided by 512 and  
 98262 rounded up to the next integer, is *n*. If *n* is followed by the character 'c', the size  
 98263 shall be in bytes.

98264 **-atime n** The primary shall evaluate as true if the file access time subtracted from the  
 98265 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

98266 **-ctime n** The primary shall evaluate as true if the time of last change of file status  
 98267 information subtracted from the initialization time, divided by 86 400 (with any  
 98268 remainder discarded), is *n*.

98269 **-mtime n** The primary shall evaluate as true if the file modification time subtracted from the  
 98270 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

98271 **-exec utility\_name [argument ...];**  
 98272 **-exec utility\_name [argument ...] {} +**  
 98273 The end of the primary expression shall be punctuated by a <semicolon> or by a  
 98274 <plus-sign>. Only a <plus-sign> that immediately follows an argument  
 98275 containing only the two characters "{}" shall punctuate the end of the primary

- 98276 expression. Other uses of the <plus-sign> shall not be treated as special.
- 98277 If the primary expression is punctuated by a <semicolon>, the utility *utility\_name*  
 98278 shall be invoked once for each pathname and the primary shall evaluate as true if  
 98279 the utility returns a zero value as exit status. A *utility\_name* or *argument* containing  
 98280 only the two characters "{}" shall be replaced by the current pathname. If a  
 98281 *utility\_name* or *argument* string contains the two characters "{}", but not just the  
 98282 two characters "{}", it is implementation-defined whether *find* replaces those two  
 98283 characters or uses the string without change.
- 98284 If the primary expression is punctuated by a <plus-sign>, the primary shall always  
 98285 evaluate as true, and the pathnames for which the primary is evaluated shall be  
 98286 aggregated into sets. The utility *utility\_name* shall be invoked once for each set of  
 98287 aggregated pathnames. Each invocation shall begin after the last pathname in the  
 98288 set is aggregated, and shall be completed before the *find* utility exits and before the  
 98289 first pathname in the next set (if any) is aggregated for this primary, but it is  
 98290 otherwise unspecified whether the invocation occurs before, during, or after the  
 98291 evaluations of other primaries. If any invocation returns a non-zero value as exit  
 98292 status, the *find* utility shall return a non-zero exit status. An argument containing  
 98293 only the two characters "{}" shall be replaced by the set of aggregated  
 98294 pathnames, with each pathname passed as a separate argument to the invoked  
 98295 utility in the same order that it was aggregated. The size of any set of two or more  
 98296 pathnames shall be limited such that execution of the utility does not cause the  
 98297 system's {ARG\_MAX} limit to be exceeded. If more than one argument containing  
 98298 the two characters "{}" is present, the behavior is unspecified.
- 98299 The current directory for the invocation of *utility\_name* shall be the same as the  
 98300 current directory when the *find* utility was started. If the *utility\_name* names any of  
 98301 the special built-in utilities (see Section 2.15, on page 2526), the results are  
 98302 undefined.
- 98303 **-ok** *utility\_name* [*argument* ...];  
 98304 The **-ok** primary shall be equivalent to **-exec**, except that the use of a <plus-sign>  
 98305 to punctuate the end of the primary expression need not be supported, and *find*  
 98306 shall request affirmation of the invocation of *utility\_name* using the current file as  
 98307 an argument by writing to standard error as described in the STDERR section. If  
 98308 the response on standard input is affirmative, the utility shall be invoked.  
 98309 Otherwise, the command shall not be invoked and the value of the **-ok** operand  
 98310 shall be false.
- 98311 **-print** The primary shall always evaluate as true; it shall cause the current pathname to  
 98312 be written to standard output, followed by a <newline>.
- 98313 **-print0** The primary shall always evaluate as true; it shall cause the current pathname to  
 98314 be written to standard output, followed by a null byte.
- 98315 **-newer** *file* The primary shall evaluate as true if the modification time of the current file is  
 98316 more recent than the modification time of the file named by the pathname *file*. If  
 98317 *file* names a symbolic link, the modification time used shall be that of the file  
 98318 referenced by the symbolic link if either the **-H** or **-L** option is specified; if neither  
 98319 **-H** nor **-L** is specified, it is unspecified whether the modification time is that of the  
 98320 symbolic link itself or of the file referenced by the symbolic link. In either case, if  
 98321 the referenced file does not exist, the modification time used shall be that of the  
 98322 link itself. If *file* is a relative pathname, it shall be resolved relative to the current  
 98323 working directory that was inherited by *find* when it was invoked.

98324        **-depth**     The primary shall always evaluate as true; it shall cause descent of the directory  
 98325                    hierarchy to be done so that all entries in a directory are acted on before the  
 98326                    directory itself. If a **-depth** primary is not specified, all entries in a directory shall  
 98327                    be acted on after the directory itself. If any **-depth** primary is specified, it shall  
 98328                    apply to the entire expression even if the **-depth** primary would not normally be  
 98329                    evaluated.

98330        The primaries can be combined using the following operators (in order of decreasing  
 98331                    precedence):

98332        (*expression* ) True if *expression* is true.

98333        !*expression*   Negation of a primary; the unary NOT operator.

98334        *expression* [-**a**] *expression*

98335                    Conjunction of primaries; the AND operator is implied by the juxtaposition of two  
 98336                    primaries or made explicit by the optional **-a** operator. The second expression shall  
 98337                    not be evaluated if the first expression is false.

98338        *expression* -**o** *expression*

98339                    Alternation of primaries; the OR operator. The second expression shall not be  
 98340                    evaluated if the first expression is true.

98341        If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given  
 98342        expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression  
 98343        shall be effectively replaced by:

98344        ( *given\_expression* ) -print

98345        The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only  
 98346        once.

98347        When the file type evaluated for the current file is a symbolic link, the results of evaluating the  
 98348        **-perm** primary are implementation-defined.

#### 98349   **STDIN**

98350        If the **-ok** primary is used, the response shall be read from the standard input. An entire line  
 98351        shall be read as the response. Otherwise, the standard input shall not be used.

#### 98352   **INPUT FILES**

98353        None.

#### 98354   **ENVIRONMENT VARIABLES**

98355        The following environment variables shall affect the execution of *find*:

98356        **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 98357                    (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 98358                    variables used to determine the values of locale categories.)

98359        **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 98360                    internationalization variables.

98361        **LC\_COLLATE**

98362                    Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 98363                    character collating elements used in the pattern matching notation for the **-name**,  
 98364                    **-iname**, and **-path** primaries and in the extended regular expression defined for  
 98365                    the **yesexpr** locale keyword in the *LC\_MESSAGES* category.



- 98366 **LC\_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of  
 98367 text data as characters (for example, single-byte as opposed to multi-byte  
 98368 characters in arguments), the behavior of character classes within the pattern  
 98369 matching notation used for the **-name**, **-iname**, and **-path** primaries, and the  
 98370 behavior of character classes within regular expressions used in the extended  
 98371 regular expression defined for the **yesexpr** locale keyword in the **LC\_MESSAGES**  
 98372 category.
- 98373 **LC\_MESSAGES**  
 98374 Determine the locale used to process affirmative responses, and the locale used to  
 98375 affect the format and contents of diagnostic messages and prompts written to  
 98376 standard error.
- 98377 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 98378 **PATH** Determine the location of the *utility\_name* for the **-exec** and **-ok** primaries, as  
 98379 described in XBD Chapter 8 (on page 167).
- 98380 **ASYNCHRONOUS EVENTS**  
 98381 Default.
- 98382 **STDOUT**  
 98383 The **-print** primary shall cause the current pathname to be written to standard output. The  
 98384 format shall be:  
 98385 "%s\n", <path>  
 98386 The **-print0** primary shall cause the current pathname to be written to standard output,  
 98387 followed by a null byte.
- 98388 **STDERR**  
 98389 The **-ok** primary shall write a prompt to standard error containing at least the *utility\_name* to be  
 98390 invoked and the current pathname. In the POSIX locale, the last non-<blank> in the prompt shall  
 98391 be ' ? '. The exact format used is unspecified.  
 98392 Otherwise, the standard error shall be used only for diagnostic messages.
- 98393 **OUTPUT FILES**  
 98394 None.
- 98395 **EXTENDED DESCRIPTION**  
 98396 None.
- 98397 **EXIT STATUS**  
 98398 The following exit values shall be returned:  
 98399 0 All *path* operands were traversed successfully, the output (if any) specified in **STDOUT** was  
 98400 successfully written to standard output, and all commands (if any) executed using the **-exec**  
 98401 primary punctuated by a <plus-sign> exited with exit status 0.  
 98402 >0 A command executed using the **-exec** primary punctuated by a <plus-sign> exited with  
 98403 non-zero status, or an error occurred.
- 98404 **CONSEQUENCES OF ERRORS**  
 98405 Default.

## 98406 APPLICATION USAGE

98407 When used in operands, pattern matching notation, <semicolon>, <left-parenthesis>, and  
 98408 <right-parenthesis> characters are special to the shell and must be quoted (see [Section 2.2](#), on  
 98409 page 2472).

98410 When restricting the search to files on one file system, it can sometimes be desirable for the  
 98411 crossing points themselves to be acted on and sometimes for them not to be acted on. (Crossing  
 98412 points are mount points and, if the **-L** option is specified, symbolic links to directories on other  
 98413 file systems.) The **-xdev** primary acts on them and the **-mount** primary does not. However,  
 98414 **-mount** also does not act on symbolic links to non-directory files on other file systems (if **-L**  
 98415 is specified). If there is a need for an application to exclude crossing points but include symbolic  
 98416 links to non-directory files on other file systems, this can be achieved by using two *find*  
 98417 commands as follows:

```
98418 find -L dir -mount -type d -print
98419 find -L dir -xdev ! -type d -print
```

98420 (in a subshell whose output is piped to *sort*, if the order matters).

98421 If both **-mount** and **-xdev** are specified, *find* obeys both primaries but the end result is the same  
 98422 as if **-xdev** were not specified.

98423 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary  
 98424 using the octal number argument form. Since this bit is not defined by this volume of  
 98425 POSIX.1-2024, applications must not assume that it actually refers to the traditional sticky bit.

## 98426 EXAMPLES

98427 1. The following commands are equivalent:

```
98428 find .
98429 find . -print
```

98430 They both write out the entire directory hierarchy from the current directory.

98431 With this output format, if any pathnames include <newline> characters, it is not possible  
 98432 to tell where each pathname begins and ends. This problem can be avoided by omitting  
 98433 such pathnames:

```
98434 LC_ALL=POSIX find . -name $'*\n*' -prune -o -print
```

98435 or by using a sentinel in the pathname that *find* would never otherwise produce, such as:

```
98436 find .//. -print
```

98437 or by using **-print0** instead of **-print** and processing the output with a utility that can  
 98438 accept null-terminated pathnames as input, such as *xargs* with the **-0** option or *read* with  
 98439 **-d ""**, for example:

```
98440 find . -print0 | while IFS= read -rd "" file
98441 do
98442     # process "$file"
98443 done
```

98444 It should be noted that using *find* with **-print0** to pipe input to *xargs* **-r0** is less safe than  
 98445 using *find* with **-exec** because if *find* **-print0** is terminated after it has written a partial  
 98446 pathname, the partial pathname may be processed as if it was a complete pathname.

- 98447 2. The following command:
- ```
98448 find / \( -name tmp -o -name '*.xx' \) ! -type d -mtime +7 \
98449 -exec rm {} +
```
- 98450 removes all files named **tmp** or ending in **.xx** that have not been modified for more than  
98451 seven (that is, eight or more) 24-hour periods.
- 98452 3. The following command:
- ```
98453 find . -perm -o+w,+s
```
- 98454 prints (**-print** is assumed) the names of all files in or below the current directory, with all  
98455 of the file permission bits **S\_ISUID**, **S\_ISGID**, and **S\_IWOTH** set, regardless of the value of  
98456 the file creation mask. (Note that the file creation mask is only specified for the file  
98457 permission bits, and not **S\_ISUID**, **S\_ISGID** or **S\_ISVTX**.)
- 98458 4. The following command:
- ```
98459 find . -perm -+w
```
- 98460 prints (**-print** is assumed) the names of all files in or below the current directory, with  
98461 **S\_IWUSR** set if the file creation mask does not have **S\_IWUSR** set (otherwise the  
98462 **S\_IWUSR** bit is ignored), **S\_IWGRP** set if the file creation mask does not have **S\_IWGRP**  
98463 set (otherwise **S\_IWGRP** is ignored), and **S\_IWOTH** set if the file creation mask does not  
98464 have **S\_IWOTH** set (otherwise **S\_IWOTH** is ignored).
- 98465 5. The following command:
- ```
98466 find . -name SCCS -prune -o -print
```
- 98467 recursively prints pathnames of all files in the current directory and below, but skips  
98468 directories named **SCCS** and files in them.
- 98469 6. The following command:
- ```
98470 find . -print -name SCCS -prune
```
- 98471 behaves as in the previous example, but prints the names of the **SCCS** directories.
- 98472 7. The following command is roughly equivalent to the **-nt** extension to *test*:
- ```
98473 if [ -n "$(find file1 -prune -newer file2)" ]; then
98474     printf %s\\n "file1 is newer than file2"
98475 fi
```
- 98476 8. The descriptions of **-atime**, **-ctime**, and **-mtime** use the terminology *n* “86 400 second  
98477 periods (days)”. For example, a file accessed at 23:59 is selected by:
- ```
98478 find . -atime -1 -print
```
- 98479 at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight  
98480 boundary between days has no effect on the 24-hour calculation.
- 98481 9. The following command:
- ```
98482 find . ! -name . -prune -name '*.old' -exec \
98483     sh -c 'mv "$@" ../old/' sh {} +
```
- 98484 performs the same task as:
- ```
98485 mv /*.old ../old /*.old ../old/
```
- 98486 while avoiding an “Argument list too long” error if there are a large number of files

98487 ending with **.old** and without running *mv* if there are no such files (and avoiding “No  
98488 such file or directory” errors if **.old** does not exist or no files match **/\*.old** or **/\*.old**).

98489 The alternative:

```
98490 find . ! -name . -prune -name '*.old' -exec mv {} ../old/ \;
```

98491 is less efficient if there are many files to move because it executes one *mv* command per  
98492 file.

98493 10. On systems configured to mount removable media on directories under **/media**, the  
98494 following command searches the file hierarchy for files of size larger than 100 000 KiB  
98495 without searching any mounted removable media:

```
98496 find / -path /media -prune -o -size +200000 -print
```

98497 11. Except for the root directory, and **"/"** on implementations where **"/"** does not refer to  
98498 the root directory, no pattern given to **-name** will match a **<slash>**, because trailing  
98499 **<slash>** characters are ignored when computing the basename of the file under  
98500 evaluation. Given two empty directories named **foo** and **bar**, the following command:

```
98501 find foo/// bar/// -name foo -o -name 'bar?*' 
```

98502 prints only the line **foo///**.

#### 98503 RATIONALE

98504 The **-a** operator was retained as an optional operator for compatibility with historical shell  
98505 scripts, even though it is redundant with expression concatenation.

98506 The descriptions of the **'-'** modifier on the *mode* and *onum* arguments to the **-perm** primary  
98507 agree with historical practice on BSD and System V implementations. System V and BSD  
98508 documentation both describe it in terms of checking additional bits; in fact, it uses the same bits,  
98509 but checks for having at least all of the matching bits set instead of having exactly the matching  
98510 bits set.

98511 The exact format of the interactive prompts is unspecified. Only the general nature of the  
98512 contents of prompts are specified because:

- 98513 • Implementations may desire more descriptive prompts than those used on historical  
98514 implementations.
- 98515 • Since the historical prompt strings do not terminate with **<newline>** characters, there is no  
98516 portable way for another program to interact with the prompts of this utility via pipes.

98517 Therefore, an application using this prompting option relies on the system to provide the most  
98518 suitable dialog directly with the user, based on the general guidelines specified.

98519 The **-size** operand refers to the size of a file, rather than the number of blocks it may occupy in  
98520 the file system. The intent is that the *st\_size* field defined in the System Interfaces volume of  
98521 POSIX.1-2024 should be used, not the *st\_blocks* found in historical implementations. There are at  
98522 least two reasons for this:

- 98523 1. In both System V and BSD, *find* only uses *st\_size* in size calculations for the operands  
98524 specified by this volume of POSIX.1-2024. (BSD uses *st\_blocks* only when processing the  
98525 **-ls** primary.)
- 98526 2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls*  
98527 utility for the output from the **-l** option. (In both System V and BSD, *ls* uses *st\_size* for the  
98528 **-l** option size field and uses *st\_blocks* for the *ls -s* calculations. This volume of  
98529 POSIX.1-2024 does not specify *ls -s*.)

98530 The descriptions of **-atime**, **-ctime**, and **-mtime** were changed from the SVID description of *n*  
 98531 ``days'' to *n* being the result of the integer division of the time difference in seconds by 86 400.  
 98532 The description is also different in terms of the exact timeframe for the *n* case (*versus* the *+n* or  
 98533 *-n*), but it matches all known historical implementations. It refers to one 86 400 second period in  
 98534 the past, not any time from the beginning of that period to the current time. For example, **-atime**  
 98535 2 is true if the file was accessed any time in the period from 72 hours to 48 hours ago.

98536 Historical implementations do not modify "{ }" when it appears as a substring of an **-exec** or  
 98537 **-ok utility\_name** or argument string. There have been numerous user requests for this extension,  
 98538 so this volume of POSIX.1-2024 allows the desired behavior. At least one recent implementation  
 98539 does support this feature, but encountered several problems in managing memory allocation  
 98540 and dealing with multiple occurrences of "{ }" in a string while it was being developed, so it is  
 98541 not yet required behavior.

98542 Assuming the presence of **-print** was added to correct a historical pitfall that plagues novice  
 98543 users, it is entirely upwards-compatible from the historical System V *find* utility. In its simplest  
 98544 form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers  
 98545 agreed that adding **-print** as a default expression was the correct decision and have added the  
 98546 fast *find* functionality within a new utility called *locate*.

98547 Historically, the **-L** option was implemented using the primary **-follow**. The **-H** and **-L** options  
 98548 were added for two reasons. First, they offer a finer granularity of control and consistency with  
 98549 other programs that walk file hierarchies. Second, the **-follow** primary always evaluated to true.  
 98550 As they were historically really global variables that took effect before the traversal began, some  
 98551 valid expressions had unexpected results. An example is the expression **-print -o -follow**.  
 98552 Because **-print** always evaluates to true, the standard order of evaluation implies that **-follow**  
 98553 would never be evaluated. This was never the case. Historical practice for the **-follow** primary,  
 98554 however, is not consistent. Some implementations always follow symbolic links on the  
 98555 command line whether **-follow** is specified or not. Others follow symbolic links on the  
 98556 command line only if **-follow** is specified. Both behaviors are provided by the **-H** and **-L**  
 98557 options, but scripts using the current **-follow** primary would be broken if the **-follow** option is  
 98558 specified to work either way.

98559 Since the **-L** option resolves all symbolic links and the **-type l** primary is true for symbolic links  
 98560 that still exist after symbolic links have been resolved, the command:

```
98561 find -L . -type l
```

98562 prints a list of symbolic links reachable from the current directory that do not resolve to  
 98563 accessible files.

98564 A feature of SVR4's *find* utility was the **-exec** primary's + terminator. This allowed filenames  
 98565 containing special characters (especially <newline> characters) to be grouped together without  
 98566 the problems that occur if such filenames are piped to *xargs*.

98567 The "**-exec ... {} +**" syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210.  
 98568 It should be noted that this is an incompatible change to IEEE Std 1003.2-1992. For example, the  
 98569 following command printed all files with a '-' after their name if they are regular files, and a  
 98570 '+' otherwise:

```
98571 find / -type f -exec echo {} - ';' -o -exec echo {} + ';' -
```

98572 The change invalidates usage like this. Even though the previous standard stated that this usage  
 98573 would work, in practice many did not support it and the standard developers felt it better to  
 98574 now state that this was not allowable.

98575 Historically, many *find* implementations supported **-mount** and **-xdev** as synonymous

98576 primaries and earlier versions of this standard only required support for `-xdev`. However, the  
 98577 behavior of `find` with `-xdev` differed from that of the `nftw()` function with `FTW_MOUNT` as  
 98578 regards whether the mount point itself was included or excluded. Therefore the standard now  
 98579 requires support for both primaries with slightly differing behaviors: `-mount` behaves in the  
 98580 manner of `nftw()` with the traditional `FTW_MOUNT` flag, and `-xdev` in the manner of `nftw()`  
 98581 with a new `FTW_XDEV` flag.

#### 98582 FUTURE DIRECTIONS

98583 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 98584 value of a <newline> character when <newline> is a terminator or separator in the output  
 98585 format being used, implementations are encouraged to treat this as an error. A future version of  
 98586 this standard may require implementations to treat this as an error.

#### 98587 SEE ALSO

98588 [Section 2.2](#) (on page 2472), [Section 2.14](#) (on page 2523), [Section 2.15](#) (on page 2526), `chmod`, `mv`,  
 98589 `pax`, `sh`, `test`

98590 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

98591 XSH `fstatat()`, `getgrgid()`, `getpwuid()`

#### 98592 CHANGE HISTORY

98593 First released in Issue 2.

#### 98594 Issue 5

98595 The FUTURE DIRECTIONS section is added.

#### 98596 Issue 6

98597 The following new requirements on POSIX implementations derive from alignment with the  
 98598 Single UNIX Specification:

- 98599 • The `-perm [-]onum` primary is supported.

98600 The `find` utility is aligned with the IEEE P1003.2b draft standard, to include processing of  
 98601 symbolic links and changes to the description of the `atime`, `ctime`, and `mtime` operands.

98602 IEEE PASC Interpretation 1003.2 #210 is applied, extending the `-exec` operand.

98603 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/13 is applied, updating the RATIONALE  
 98604 section to be consistent with the normative text.

#### 98605 Issue 7

98606 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
 98607 `LC_MESSAGES` environment variable.

98608 Austin Group Interpretation 1003.1-2001 #127 is applied, rephrasing the description of the `-exec`  
 98609 primary to be “immediately follows”.

98610 Austin Group Interpretation 1003.1-2001 #185 is applied, clarifying the requirements for the `-H`  
 98611 and `-L` options.

98612 Austin Group Interpretation 1003.1-2001 #186 is applied, clarifying the requirements for the  
 98613 evaluation of `path` operands.

98614 Austin Group Interpretation 1003.1-2001 #195 is applied, clarifying the interpretation of the first  
 98615 operand.

98616 SD5-XCU-ERN-48 is applied, clarifying the `-L` option in the case that the referenced file does not  
 98617 exist.

98618 SD5-XCU-ERN-89 is applied, updating the OPERANDS section.

- 98619 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 98620 SD5-XCU-ERN-117 is applied, clarifying the **-perm** operand.
- 98621 SD5-XCU-ERN-122 is applied, adding a new EXAMPLE.
- 98622 The description of the **-name** primary is revised and the **-path** primary is added (with a new  
98623 example).
- 98624 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0086 [365], XCU/TC1-2008/0087  
98625 [310], XCU/TC1-2008/0088 [309,310,430], XCU/TC1-2008/0089 [235], and XCU/TC1-2008/0090  
98626 [445] are applied.
- 98627 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0099 [584], XCU/TC2-2008/0100  
98628 [584], and XCU/TC2-2008/0101 [584] are applied.
- 98629 **Issue 8**
- 98630 Austin Group Defect 243 is applied, adding the **-print0** primary.
- 98631 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
98632 directed to display a pathname that contains any bytes that have the encoded value of a  
98633 <newline> character when <newline> is a terminator or separator in the output format being  
98634 used.
- 98635 Austin Group Defect 1031 is applied, adding the **-iname** primary.
- 98636 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 98637 Austin Group Defect 1133 is applied, adding the **-mount** primary.
- 98638 Austin Group Defects 1259 and 1777 are applied, changing the EXAMPLES section.
- 98639 Austin Group Defect 1392 is applied, changing the effect of the file creation mask on the *mode*  
98640 argument for the **-perm** primary to be consistent with *chmod*.
- 98641 Austin Group Defect 1501 is applied, changing the EXIT STATUS section.
- 98642 Austin Group Defect 1553 is applied, changing the ENVIRONMENT VARIABLES section.
- 98643 Austin Group Defect 1554 is applied, changing the RATIONALE section.
- 98644 Austin Group Defect 1606 is applied, clarifying that if *find* detects an infinite loop and recovers  
98645 its position, the final exit status is non-zero.
- 98646 Austin Group Defect 1776 is applied, clarifying how symbolic links are handled by the **-newer**  
98647 *file* primary.

98648 **NAME**

98649 fold — filter for folding lines

98650 **SYNOPSIS**98651 fold [-bs] [-w *width*] [*file...*]98652 **DESCRIPTION**

98653 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a  
 98654 maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be  
 98655 broken by the insertion of a <newline> such that each output line (referred to later in this section  
 98656 as a *segment*) is the maximum width possible that does not exceed the specified number of  
 98657 column positions (or bytes). A line shall not be broken in the middle of a character. The behavior  
 98658 is undefined if *width* is less than the number of columns any single character in the input would  
 98659 occupy.

98660 If the <carriage-return>, <backspace>, or <tab> characters are encountered in the input, and the  
 98661 **-b** option is not specified, they shall be treated specially:

98662 <backspace> The current count of line width shall be decremented by one, although the count  
 98663 never shall become negative. The *fold* utility shall not insert a <newline>  
 98664 immediately before or after any <backspace>, unless the following character has a  
 98665 width greater than 1 and would cause the line width to exceed *width*.

98666 <carriage-return>  
 98667 The current count of line width shall be set to zero. The *fold* utility shall not insert a  
 98668 <newline> immediately before or after any <carriage-return>.

98669 <tab> Each <tab> encountered shall advance the column position pointer to the next tab  
 98670 stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

98671 **OPTIONS**98672 The *fold* utility shall conform to XBD [Section 12.2](#) (on page 215).

98673 The following options shall be supported:

98674 **-b** Count *width* in bytes rather than column positions.

98675 **-s** If a segment of a line contains a <blank> within the first *width* column positions (or  
 98676 bytes), break the line after the last such <blank> meeting the width constraints. If  
 98677 there is no <blank> meeting the requirements, the **-s** option shall have no effect for  
 98678 that output segment of the input line.

98679 **-w *width*** Specify the maximum line length, in column positions (or bytes if **-b** is specified).  
 98680 The results are unspecified if *width* is not a positive decimal number. The default  
 98681 value shall be 80.

98682 **OPERANDS**

98683 The following operand shall be supported:

98684 *file* A pathname of a text file to be folded. If no *file* operands are specified, the standard  
 98685 input shall be used.

98686 **STDIN**

98687 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
 98688 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
 98689 the standard input shall not be used. See the INPUT FILES section.



98690 **INPUT FILES**

98691 If the **-b** option is specified, the input files shall be text files except that the lines are not limited  
98692 to {LINE\_MAX} bytes in length. If the **-b** option is not specified, the input files shall be text files.

98693 **ENVIRONMENT VARIABLES**

98694 The following environment variables shall affect the execution of *fold*:

98695 **LANG** Provide a default value for the internationalization variables that are unset or null.  
98696 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
98697 variables used to determine the values of locale categories.)

98698 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
98699 internationalization variables.

98700 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
98701 characters (for example, single-byte as opposed to multi-byte characters in  
98702 arguments and input files), and for the determination of the width in column  
98703 positions each character would occupy on a constant-width font output device.

98704 **LC\_MESSAGES**

98705 Determine the locale that should be used to affect the format and contents of  
98706 diagnostic messages written to standard error.

98707 XSI **NLSPATH** Determine the location of messages objects and message catalogs.

98708 **ASYNCHRONOUS EVENTS**

98709 Default.

98710 **STDOUT**

98711 The standard output shall be a file containing a sequence of characters whose order shall be  
98712 preserved from the input files, possibly with inserted <newline> characters.

98713 **STDERR**

98714 The standard error shall be used only for diagnostic messages.

98715 **OUTPUT FILES**

98716 None.

98717 **EXTENDED DESCRIPTION**

98718 None.

98719 **EXIT STATUS**

98720 The following exit values shall be returned:

98721 0 All input files were processed successfully.

98722 >0 An error occurred.

98723 **CONSEQUENCES OF ERRORS**

98724 Default.

**98725 APPLICATION USAGE**

98726 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.  
98727 The *cut* utility should be used when the number of lines (or records) needs to remain constant.  
98728 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

98729 The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines  
98730 wider than the printer is able to print (usually 80 or 132 column positions).

**98731 EXAMPLES**

98732 An example invocation that submits a file of possibly long lines to the printer (under the  
98733 assumption that the user knows the line width of the printer to be assigned by *lp*):

```
98734 fold -w 132 bigfile | lp
```

**98735 RATIONALE**

98736 Although terminal input in canonical processing mode requires the erase character (frequently  
98737 set to <backspace>) to erase the previous character (not byte or column position), terminal  
98738 output is not buffered and is extremely difficult, if not impossible, to parse correctly; the  
98739 interpretation depends entirely on the physical device that actually displays/prints/stores the  
98740 output. In all known internationalized implementations, the utilities producing output for  
98741 mixed column-width output assume that a <backspace> character backs up one column position  
98742 and outputs enough <backspace> characters to return to the start of the character when  
98743 <backspace> is used to provide local line motions to support underlining and boldening  
98744 operations. Since *fold* without the **-b** option is dealing with these same constraints, <backspace>  
98745 is always treated as backing up one column position rather than backing up one character.

98746 Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column  
98747 position when written out. This is no longer always true. Since the most common usage of *fold* is  
98748 believed to be folding long lines for output to limited-length output devices, this capability was  
98749 preserved as the default case. The **-b** option was added so that applications could *fold* files with  
98750 arbitrary length lines into text files that could then be processed by the standard utilities. Note  
98751 that although the width for the **-b** option is in bytes, a line is never split in the middle of a  
98752 character. (It is unspecified what happens if a width is specified that is too small to hold a single  
98753 character found in the input followed by a <newline>.)

98754 The tab stops are hardcoded to be every eighth column to meet historical practice. No new  
98755 method of specifying other tab stops was invented.

**98756 FUTURE DIRECTIONS**

98757 None.

**98758 SEE ALSO**

98759 *cut*

98760 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**98761 CHANGE HISTORY**

98762 First released in Issue 4.

**98763 Issue 6**

98764 The normative text is reworded to avoid use of the term “must” for application requirements.

**98765 Issue 7**

98766 Austin Group Interpretation 1003.1-2001 #092 is applied.

- 98767 Austin Group Interpretation 1003.1-2001 #204 is applied, updating the DESCRIPTION to clarify  
98768 when a <newline> can be inserted before or after a <backspace>.
- 98769 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 98770 **Issue 8**  
98771 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

98772 **NAME**

98773 fuser — list process IDs of all processes that are using one or more named files

98774 **SYNOPSIS**98775 XSI `fuser [-cfu] file...`98776 **DESCRIPTION**

98777 For each *file* operand, in order, *fuser* shall write one line of output, some of it to standard output,  
 98778 and the rest to standard error, giving information about processes running on the local system  
 98779 that are using the file. A process shall be considered to be using a file if it has at least one open  
 98780 file descriptor associated with the file or if the file is a directory that is the current working  
 98781 directory or the root directory for the process, and may be considered to be using a file for other  
 98782 implementation-dependent reasons. If *file* names a block special device that contains a mounted  
 98783 file system, and the `-f` option is not specified, any processes using any file on that mounted file  
 98784 system and any processes that are using the device file itself shall be listed.

98785 Any output for processes running on remote systems that are using a named file is unspecified.

98786 A user may need appropriate privileges to invoke the *fuser* utility.

98787 When standard output and standard error are directed to the same file, the output for each *file*  
 98788 operand shall be interleaved so that it is written to the file in the following order:

- 98789 • On standard error, a pathname for the file, immediately followed by a <colon> and zero or  
 98790 more <blank> characters. The pathname shall be either the file operand (unaltered) or the  
 98791 pathname that would result from a successful call to the *realpath()* function, defined in the  
 98792 System Interfaces volume of POSIX.1-2024, with the *file* operand as its *file\_name* argument.

- 98793 • For each process using the file:

- 98794 — On standard output, the process ID in the format:

98795 " %1d", <process ID>

- 98796 — On standard error, information about the file's use by the process, in the following  
 98797 format:

98798 "%s", <use chars>

98799 if the `-u` option is not specified, or in the following format:

98800 "%s(%s)", <use chars>, <user name>

98801 if the `-u` option is specified, where <use chars> is a string of zero or more characters  
 98802 indicating the use of the file and <user name> is the user name corresponding to the  
 98803 real user ID of the process or, if the user name cannot be resolved from the real user  
 98804 ID of the process, the real user ID of the process in decimal. The value of <use chars>  
 98805 shall include the character 'c' if the process is using the file as its current directory  
 98806 and the character 'r' if the process is using the file as its root directory;  
 98807 implementations may include other alphabetic characters to indicate other uses of  
 98808 the file.

- 98809 • On standard error, a <newline> character.

98810 When standard output and standard error are not directed to the same file, the data written to  
 98811 each shall be as described above but the ordering of writes to standard output relative to writes  
 98812 to standard error is unspecified. For example, *fuser* might first write the information for all file  
 98813 operands to standard error and then write all of the process IDs to standard output.

98814 **OPTIONS**98815 The *fuser* utility shall conform to XBD [Section 12.2](#) (on page 215).

98816 The following options shall be supported:

98817 **-c** If a *file* operand names a directory that is the mount point of a mounted file system,  
98818 all processes using any file on that file system shall be listed as if they were using  
98819 the named directory. The behavior for any *file* operand that names an existing file  
98820 that is not the mount point of a mounted file system is unspecified.98821 **-f** The report shall be only for the named files.98822 **-u** The user name, in parentheses, associated with each process ID written to standard  
98823 output shall be written to standard error.98824 **OPERANDS**

98825 The following operand shall be supported:

98826 *file* A pathname of a file for which the processes using the file are to be reported.98827 **STDIN**

98828 Not used.

98829 **INPUT FILES**

98830 The user database.

98831 **ENVIRONMENT VARIABLES**98832 The following environment variables shall affect the execution of *fuser*:98833 *LANG* Provide a default value for the internationalization variables that are unset or null.  
98834 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
98835 variables used to determine the values of locale categories.)98836 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
98837 internationalization variables.98838 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
98839 characters (for example, single-byte as opposed to multi-byte characters in  
98840 arguments).98841 *LC\_MESSAGES*98842 Determine the locale that should be used to affect the format and contents of  
98843 diagnostic messages written to standard error.98844 *NLSPATH* Determine the location of messages objects and message catalogs.98845 **ASYNCHRONOUS EVENTS**

98846 Default.

98847 **STDOUT**

98848 See DESCRIPTION.

98849 **STDERR**98850 The *fuser* utility shall write diagnostic messages to standard error.98851 The *fuser* utility also shall write information to standard error as specified in the DESCRIPTION  
98852 section.

98853 **OUTPUT FILES**

98854 None.

98855 **EXTENDED DESCRIPTION**

98856 None.

98857 **EXIT STATUS**

98858 The following exit values shall be returned:

98859 0 Successful completion.

98860 &gt;0 An error occurred.

98861 **CONSEQUENCES OF ERRORS**

98862 Default.

98863 **APPLICATION USAGE**98864 Things can change while *fuser* is running; the snapshot it gives is only true for an instant, and  
98865 might not be accurate by the time it is displayed.98866 **EXAMPLES**

98867 The command:

98868 `fuser -fu .`98869 writes to standard output the process IDs of processes that are using the current directory and  
98870 writes to standard error an indication of how those processes are using the directory and the  
98871 user names associated with the processes that are using the current directory.98872 `fuser -c <mount point>`98873 writes to standard output the process IDs of processes that are using any file in the file system  
98874 which is mounted on *<mount point>* and writes to standard error an indication of how those  
98875 processes are using the files.98876 `fuser <mount point>`98877 writes to standard output the process IDs of processes that are using the file which is named by  
98878 *<mount point>* and writes to standard error an indication of how those processes are using the  
98879 file.98880 `fuser <mounted block device>`98881 writes to standard output the process IDs of processes that are using any file on the mounted file  
98882 system contained by *<mounted block device>* and of processes that are using the device file  
98883 *<mounted block device>* itself, and writes to standard error an indication of how those processes  
98884 are using the files.98885 `fuser -f <mounted block device>`98886 writes to standard output the process IDs of processes that are using the file *<mounted block  
98887 device>* itself and writes to standard error an indication of how those processes are using the file.98888 **RATIONALE**98889 The definition of the *fuser* utility follows existing practice.98890 **FUTURE DIRECTIONS**98891 If this utility is directed to display a pathname that contains any bytes that have the encoded  
98892 value of a *<newline>* character when *<newline>* is a terminator or separator in the output  
98893 format being used, implementations are encouraged to treat this as an error. A future version of  
98894 this standard may require implementations to treat this as an error.

98895 **SEE ALSO**

98896 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

98897 **CHANGE HISTORY**

98898 First released in Issue 5.

98899 **Issue 7**

98900 SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

98901 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

98902 **Issue 8**

98903 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
98904 directed to display a pathname that contains any bytes that have the encoded value of a  
98905 <newline> character when <newline> is a terminator or separator in the output format being  
98906 used.

98907 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

98908 Austin Group Defect 1746 is applied, clarifying the output written to standard output and  
98909 standard error.

98910 **NAME**

98911 gencat — generate a formatted message catalog

98912 **SYNOPSIS**98913 gencat *catfile* *msgfile*...98914 **DESCRIPTION**

98915 The *gencat* utility shall merge the message text source file *msgfile* into a formatted message  
 98916 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its  
 98917 messages shall be included in the new *catfile*. If set and message numbers collide, the new  
 98918 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

98919 **OPTIONS**

98920 None.

98921 **OPERANDS**

98922 The following operands shall be supported:

98923 *catfile* A pathname of the formatted message catalog. If '-' is specified, standard output  
 98924 shall be used. The format of the message catalog produced is unspecified.

98925 *msgfile* A pathname of a message text source file. If '-' is specified for an instance of  
 98926 *msgfile*, standard input shall be used. The format of message text source files is  
 98927 defined in the EXTENDED DESCRIPTION section.

98928 **STDIN**98929 The standard input shall not be used unless a *msgfile* operand is specified as '-'.98930 **INPUT FILES**

98931 The input files shall be text files.

98932 **ENVIRONMENT VARIABLES**98933 The following environment variables shall affect the execution of *gencat*:

98934 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 98935 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 98936 variables used to determine the values of locale categories.)

98937 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 98938 internationalization variables.

98939 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 98940 characters (for example, single-byte as opposed to multi-byte characters in  
 98941 arguments and input files).

98942 *LC\_MESSAGES*

98943 Determine the locale that should be used to affect the format and contents of  
 98944 diagnostic messages written to standard error.

98945 XSI *NLSPATH* Determine the location of messages objects and message catalogs.98946 **ASYNCHRONOUS EVENTS**

98947 Default.

98948 **STDOUT**98949 The standard output shall not be used unless the *catfile* operand is specified as '-'.



98950 **STDERR**

98951 The standard error shall be used only for diagnostic messages.

98952 **OUTPUT FILES**

98953 None.

98954 **EXTENDED DESCRIPTION**98955 The content of a message text file shall be in the format defined as follows. Note that the fields of  
98956 a message text source line are separated by a single <blank> character. Any other <blank>  
98957 characters are considered to be part of the subsequent field.98958 **\$set** *n comment*98959 This line specifies the set identifier of the following messages until the next **\$set** or  
98960 end-of-file appears. The *n* denotes the set identifier, which is defined as a number  
98961 in the range [1, {NL\_SETMAX}] (see the <limits.h> header defined in the Base  
98962 Definitions volume of POSIX.1-2024). The application shall ensure that set  
98963 identifiers are presented in ascending order within a single source file, but need  
98964 not be contiguous. Any string following the set identifier shall be treated as a  
98965 comment. If no **\$set** directive is specified in a message text source file, all messages  
98966 shall be located in an implementation-defined default message set NL\_SETD (see  
98967 the <nl\_types.h> header defined in the Base Definitions volume of POSIX.1-2024).98968 **\$delset** *n comment*98969 This line deletes message set *n* from an existing message catalog. The *n* denotes the  
98970 set number [1, {NL\_SETMAX}]. Any string following the set number shall be  
98971 treated as a comment.98972 **\$ comment** A line beginning with '\$' followed by a <blank> shall be treated as a comment.98973 *m message-text*98974 The *m* denotes the message identifier, which is defined as a number in the range [1,  
98975 {NL\_MSGMAX}] (see the <limits.h> header). The *message-text* shall be stored in the  
98976 message catalog with the set identifier specified by the last **\$set** directive, and with  
98977 message identifier *m*. If the *message-text* is empty, and a <blank> field separator is  
98978 present, an empty string shall be stored in the message catalog. If a message source  
98979 line has a message number, but neither a field separator nor *message-text*, the  
98980 existing message with that number (if any) shall be deleted from the catalog. The  
98981 application shall ensure that message identifiers are in ascending order within a  
98982 single set, but need not be contiguous. The application shall ensure that the length  
98983 of *message-text* is in the range [0, {NL\_TEXTMAX}] (see the <limits.h> header).98984 **\$quote** *c* This line specifies an optional quote character *c*, which can be used to surround  
98985 *message-text* so that trailing <space> characters or null (empty) messages are visible  
98986 in a message source line. By default, or if an empty **\$quote** directive is supplied, no  
98987 quoting of *message-text* shall be recognized.98988 Empty lines in a message text source file shall be ignored. The effects of lines starting with any  
98989 character other than those defined above are implementation-defined.98990 Text strings can contain the special characters and escape sequences defined in the following  
98991 table:

|       | Description       | Symbol | Sequence |
|-------|-------------------|--------|----------|
| 98992 | <newline>         | NL(LF) | \n       |
| 98993 | Horizontal-tab    | HT     | \t       |
| 98994 | <vertical-tab>    | VT     | \v       |
| 98995 | <backspace>       | BS     | \b       |
| 98996 | <carriage-return> | CR     | \r       |
| 98997 | <form-feed>       | FF     | \f       |
| 98998 | Backslash         | \      | \\       |
| 98999 | Bit pattern       | ddd    | \ddd     |
| 99000 |                   |        |          |

99001 The escape sequence "\ddd" consists of <backslash> followed by one, two, or three octal digits,  
 99002 which shall be taken to specify the value of the desired character. If the character following a  
 99003 <backslash> is not one of those specified, the <backslash> shall be ignored.

99004 A <backslash> followed by a <newline> is also used to continue a string on the following line.  
 99005 Thus, the following two lines describe a single message string:

```
99006 1 This line continues \  

  99007 to the next line
```

99008 which shall be equivalent to:

```
99009 1 This line continues to the next line
```

#### 99010 EXIT STATUS

99011 The following exit values shall be returned:

99012 0 Successful completion.

99013 >0 An error occurred.

#### 99014 CONSEQUENCES OF ERRORS

99015 Default.

#### 99016 APPLICATION USAGE

99017 Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot  
 99018 be guaranteed between different types of machine. Thus, just as C programs need to be  
 99019 recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

#### 99020 EXAMPLES

99021 None.

#### 99022 RATIONALE

99023 None.

#### 99024 FUTURE DIRECTIONS

99025 None.

#### 99026 SEE ALSO

99027 *iconv*

99028 XBD Chapter 8 (on page 167), [<limits.h>](#), [<nl\\_types.h>](#)

#### 99029 CHANGE HISTORY

99030 First released in Issue 3.

#### 99031 Issue 6

99032 The normative text is reworded to avoid use of the term “must” for application requirements.

99033 **Issue 7**

99034 The *gencat* utility is moved from the XSI option to the Base.

99035 **Issue 8**

99036 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

99037 Austin Group Defect 1463 is applied, changing *n* to *c* in the definition of **\$quote**.

99038 Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

99039 **NAME**99040 get — get a version of an SCCS file (**DEVELOPMENT**)99041 **SYNOPSIS**99042 XSI `get [-begkmlLpst] [-c cutoff] [-i list] [-r SID] [-x list] file...`99043 **DESCRIPTION**99044 The *get* utility shall generate a text file from each named SCCS *file* according to the specifications  
99045 given by its options.99046 The generated text shall normally be written into a file called the **g-file** whose name is derived  
99047 from the SCCS filename by simply removing the leading "s.". 99048 **OPTIONS**99049 The *get* utility shall conform to XBD [Section 12.2](#) (on page 215).

99050 The following options shall be supported:

99051 **-r** *SID* Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file  
99052 to be retrieved. The table shows, for the most useful cases, what version of an  
99053 SCCS file is retrieved (as well as the SID of the version to be eventually created by  
99054 *delta* if the **-e** option is also used), as a function of the SID specified.99055 **-c** *cutoff* Indicate the *cutoff* date-time, in the form:99056 `YY[MM[DD[HH[MM[SS]]]]]`99057 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999  
99058 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.99059 **Note:** It is expected that in a future version of this standard the default century inferred  
99060 from a 2-digit year will change. (This would apply to all commands accepting a  
99061 2-digit year as input.)99062 No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
99063 date-time shall be included in the generated text file. Units omitted from the date-  
99064 time default to their maximum possible values; for example, **-c** 7502 is equivalent  
99065 to **-c** 750228235959.99066 Any number of non-numeric characters may separate the various 2-digit pieces of  
99067 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:  
99068 **-c** "77/2/2 9:22:25".99069 **-e** Indicate that the *get* is for the purpose of editing or making a change (delta) to the  
99070 SCCS file via a subsequent use of *delta*. The **-e** option used in a *get* for a particular  
99071 version (SID) of the SCCS file shall prevent further *get* commands from editing on  
99072 the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.  
99073 Concurrent use of *get* **-e** for different SIDs is always allowed.99074 If the **g-file** generated by *get* with a **-e** option is accidentally ruined in the process  
99075 of editing, it may be regenerated by re-executing the *get* command with the **-k**  
99076 option in place of the **-e** option.99077 SCCS file protection specified via the ceiling, floor, and authorized user list stored  
99078 in the SCCS file shall be enforced when the **-e** option is used.99079 **-b** Use with the **-e** option to indicate that the new delta should have an SID in a new  
99080 branch as shown in the table below. This option shall be ignored if the **b** flag is not  
99081 present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that  
99082 has no successors on the SCCS file tree.)

- 99083                   **Note:**       A branch delta may always be created from a non-leaf delta.
- 99084           **-i list**       Indicate a *list* of deltas to be included (forced to be applied) in the creation of the  
99085                   generated file. The *list* has the following syntax:
- 99086                   <list> ::= <range> | <list> , <range>  
99087                   <range> ::= SID | SID - SID
- 99088                   SID, the SCCS Identification of a delta, may be in any form shown in the "SID  
99089                   Specified" column of the table in the EXTENDED DESCRIPTION section, except  
99090                   that the result of supplying a partial SID is unspecified. A diagnostic message shall  
99091                   be written if the first SID in the range is not an ancestor of the second SID in the  
99092                   range.
- 99093           **-x list**       Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of  
99094                   the generated file. See the **-i** option for the *list* format.
- 99095           **-k**           Suppress replacement of identification keywords (see below) in the retrieved text  
99096                   by their value. The **-k** option shall be implied by the **-e** option.
- 99097           **-l**           Write a delta summary into an **l-file**.
- 99098           **-L**           Write a delta summary to standard output. All informative output that normally is  
99099                   written to standard output shall be written to standard error instead, unless the **-s**  
99100                   option is used, in which case it shall be suppressed.
- 99101           **-p**           Write the text retrieved from the SCCS file to the standard output. No **g-file** shall  
99102                   be created. All informative output that normally goes to the standard output shall  
99103                   go to standard error instead, unless the **-s** option is used, in which case it shall  
99104                   disappear.
- 99105           **-s**           Suppress all informative output normally written to standard output. However,  
99106                   fatal error messages (which shall always be written to the standard error) shall  
99107                   remain unaffected.
- 99108           **-m**           Precede each text line retrieved from the SCCS file by the SID of the delta that  
99109                   inserted the text line in the SCCS file. The format shall be:
- 99110                   "%s\t%s", <SID>, <text line>
- 99111           **-n**           Precede each generated text line with the %M% identification keyword value (see  
99112                   below). The format shall be:
- 99113                   "%s\t%s", <%M% value>, <text line>
- 99114                   When both the **-m** and **-n** options are used, the <text line> shall be replaced by the  
99115                   **-m** option-generated format.
- 99116           **-g**           Suppress the actual retrieval of text from the SCCS file. It is primarily used to  
99117                   generate an **l-file**, or to verify the existence of a particular SID.
- 99118           **-t**           Use to access the most recently created (top) delta in a given release (for example,  
99119                   **-r 1**), or release and level (for example, **-r 1.2**).

99120 **OPERANDS**

99121 The following operands shall be supported:

99122 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*  
 99123 utility shall behave as though each file in the directory were specified as a named  
 99124 file, except that non-SCCS files (last component of the pathname does not begin  
 99125 with **s**.) and unreadable files shall be silently ignored.

99126 If exactly one *file* operand appears, and it is **'-'**, the standard input shall be read;  
 99127 each line of the standard input is taken to be the name of an SCCS file to be  
 99128 processed. Non-SCCS files and unreadable files shall be silently ignored.

99129 **STDIN**

99130 The standard input shall be a text file used only if the *file* operand is specified as **'-'**. Each line  
 99131 of the text file shall be interpreted as an SCCS pathname.

99132 **INPUT FILES**

99133 The SCCS files shall be files of an unspecified format.

99134 **ENVIRONMENT VARIABLES**99135 The following environment variables shall affect the execution of *get*:

99136 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 99137 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 99138 variables used to determine the values of locale categories.)

99139 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 99140 internationalization variables.

99141 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 99142 characters (for example, single-byte as opposed to multi-byte characters in  
 99143 arguments and input files).

99144 *LC\_MESSAGES*

99145 Determine the locale that should be used to affect the format and contents of  
 99146 diagnostic messages written to standard error, and informative messages written  
 99147 to standard output (or standard error, if the **-p** option is used).

99148 *NLSPATH* Determine the location of messages objects and message catalogs.

99149 *TZ* Determine the timezone in which the times and dates written in the SCCS file are  
 99150 evaluated. If the *TZ* variable is unset or NULL, an unspecified system default  
 99151 timezone is used.

99152 **ASYNCHRONOUS EVENTS**

99153 Default.

99154 **STDOUT**

99155 For each file processed, *get* shall write to standard output the SID being accessed and the  
 99156 number of lines retrieved from the SCCS file, in the following format:

99157 "%s\n%d lines\n", &lt;SID&gt;, &lt;number of lines&gt;

99158 If the **-e** option is used, the SID of the delta to be made shall appear after the SID accessed and  
 99159 before the number of lines generated, in the POSIX locale:

99160 "%s\nnew delta %s\n%d lines\n", <SID accessed>,  
 99161 <SID to be made>, <number of lines>

99162 If there is more than one named file or if a directory or standard input is named, each pathname

99163 shall be written before each of the lines shown in one of the preceding formats:

99164 `"\n%s:\n", <pathname>`

99165 If the `-L` option is used, a delta summary shall be written following the format specified below  
99166 for **l-files**.

99167 If the `-i` option is used, included deltas shall be listed following the notation, in the POSIX  
99168 locale:

99169 `"Included:\n"`

99170 If the `-x` option is used, excluded deltas shall be listed following the notation, in the POSIX  
99171 locale:

99172 `"Excluded:\n"`

99173 If the `-p` or `-L` options are specified, the standard output shall consist of the text retrieved from  
99174 the SCCS file.

#### 99175 **STDERR**

99176 The standard error shall be used only for diagnostic messages, except if the `-p` or `-L` options are  
99177 specified, it shall include all informative messages normally sent to standard output.

#### 99178 **OUTPUT FILES**

99179 Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-**  
99180 **file**, **p-file**, and **z-file**. The letter before the <hyphen-minus> is called the *tag*. An auxiliary  
99181 filename shall be formed from the SCCS filename: the application shall ensure that the last  
99182 component of all SCCS filenames is of the form *s.module-name*; the auxiliary files shall be named  
99183 by replacing the leading *s* with the tag. The **g-file** shall be an exception to this scheme: the **g-file**  
99184 is named by removing the *s*. prefix. For example, for *s.xyz.c*, the auxiliary filenames would be  
99185 *xyz.c*, *l.xyz.c*, *p.xyz.c*, and *z.xyz.c*, respectively.

99186 The **g-file**, which contains the generated text, shall be created in the current directory (unless the  
99187 `-p` option is used). A **g-file** shall be created in all cases, whether or not any lines of text were  
99188 generated by the *get*. It shall be owned by the real user. If the `-k` option is used or implied, the  
99189 **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be  
99190 read-only. Only the real user need have write permission in the current directory.

99191 The **l-file** shall contain a table showing which deltas were applied in generating the retrieved  
99192 text. The **l-file** shall be created in the current directory if the `-l` option is used; it shall be read-  
99193 only and it is owned by the real user. Only the real user need have write permission in the  
99194 current directory.

99195 Lines in the **l-file** shall have the following format:

99196 `"%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,  
99197 <SID>, <date-time>, <login>`

99198 where the entries are:

99199 `<code1>` A <space> if the delta was applied; ' \* ' otherwise.

99200 `<code2>` A <space> if the delta was applied or was not applied and ignored; ' \* ' if the delta  
99201 was not applied and was not ignored.

99202 `<code3>` A character indicating a special reason why the delta was or was not applied:

99203 **I** Included.

99204 X Excluded.

99205 C Cut off (by a `-c` option).

99206 `<date-time>` Date and time (using the format of the `date` utility's `%Y/%m/%d %T` conversion  
99207 specification format) of creation.

99208 `<login>` Login name of person who created `delta`.

99209 The comments and MR data shall follow on subsequent lines, indented one `<tab>`. A blank line  
99210 shall terminate each entry.

99211 The **p-file** shall be used to pass information resulting from a `get` with a `-e` option along to `delta`.  
99212 Its contents shall also be used to prevent a subsequent execution of `get` with a `-e` option for the  
99213 same SID until `delta` is executed or the joint edit flag, `j`, is set in the SCCS file. The **p-file** shall be  
99214 created in the directory containing the SCCS file and the application shall ensure that the  
99215 effective user has write permission in that directory. It shall be writable by owner only, and  
99216 owned by the effective user. Each line in the **p-file** shall have the following format:

99217 `"%sΔ%sΔ%sΔ%s%s%s\n", <g-file SID>,  
99218 <SID of new delta>, <login-name of real user>,  
99219 <date-time>, <i-value>, <x-value>`

99220 where `<i-value>` uses the format " " if no `-i` option was specified, and shall use the format:

99221 `"Δ-i%s", <-i option option-argument>`

99222 if a `-i` option was specified and `<x-value>` uses the format " " if no `-x` option was specified, and  
99223 shall use the format:

99224 `"Δ-x%s", <-x option option-argument>`

99225 if a `-x` option was specified. There can be an arbitrary number of lines in the **p-file** at any time;  
99226 no two lines shall have the same new delta SID.

99227 The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall  
99228 be the binary process ID of the command (that is, `get`) that created it. The **z-file** shall be created  
99229 in the directory containing the SCCS file for the duration of `get`. The same protection restrictions  
99230 as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.



## 99231 EXTENDED DESCRIPTION

| Determination of SCCS Identification String |                    |                  |                                             |                            |                |
|---------------------------------------------|--------------------|------------------|---------------------------------------------|----------------------------|----------------|
| SID* Specified                              | -b Keyletter Used† | Other Conditions | SID Retrieved                               | SID of Delta to be Created |                |
| 99235                                       | none‡              | no               | R defaults to mR                            | mR.mL                      | mR.(mL+1)      |
| 99236                                       | none‡              | yes              | R defaults to mR                            | mR.mL                      | mR.mL.(mB+1).1 |
| 99237                                       | R                  | no               | R > mR                                      | mR.mL                      | R.1***         |
| 99238                                       | R                  | no               | R = mR                                      | mR.mL                      | mR.(mL+1)      |
| 99239                                       | R                  | yes              | R > mR                                      | mR.mL                      | mR.mL.(mB+1).1 |
| 99240                                       | R                  | yes              | R = mR                                      | mR.mL                      | mR.mL.(mB+1).1 |
| 99241                                       | R                  | -                | R < mR and R does not exist                 | hR.mL**                    | hR.mL.(mB+1).1 |
| 99242                                       |                    |                  |                                             |                            |                |
| 99243                                       | R                  | -                | Trunk successor in release > R and R exists | R.mL                       | R.mL.(mB+1).1  |
| 99244                                       |                    |                  |                                             |                            |                |
| 99245                                       | R.L                | no               | No trunk successor                          | R.L                        | R.(L+1)        |
| 99246                                       | R.L                | yes              | No trunk successor                          | R.L                        | R.L.(mB+1).1   |
| 99247                                       | R.L                | -                | Trunk successor in release ≥ R              | R.L                        | R.L.(mB+1).1   |
| 99248                                       |                    |                  |                                             |                            |                |
| 99249                                       | R.L.B              | no               | No branch successor                         | R.L.B.mS                   | R.L.B.(mS+1)   |
| 99250                                       | R.L.B              | yes              | No branch successor                         | R.L.B.mS                   | R.L.(mB+1).1   |
| 99251                                       | R.L.B.S            | no               | No branch successor                         | R.L.B.S                    | R.L.B.(S+1)    |
| 99252                                       | R.L.B.S            | yes              | No branch successor                         | R.L.B.S                    | R.L.(mB+1).1   |
| 99253                                       | R.L.B.S            | -                | Branch successor                            | R.L.B.S                    | R.L.(mB+1).1   |

99254 \* R, L, B, and S are the release, level, branch, and sequence components of the SID,  
 99255 respectively; m means maximum. Thus, for example, R.mL means “the maximum level  
 99256 number within release R”; R.L.(mB+1).1 means “the first sequence number on the new  
 99257 branch (that is, maximum branch number plus one) of level L within release R”. Note  
 99258 that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified  
 99259 components shall exist.

99260 \*\* hR is the highest existing release that is lower than the specified, nonexistent, release R.

99261 \*\*\* This is used to force creation of the first delta in a new release.

99262 † The -b option is effective only if the b flag is present in the file. An entry of ‘-’ means  
 99263 “irrelevant”.

99264 ‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is  
 99265 present in the file, then the SID obtained from the d flag is interpreted as if it had been  
 99266 specified on the command line. Thus, one of the other cases in this table applies.

99267 **System Date and Time**

99268 When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into  
 99269 account. If any of these times are apparently in the future, the behavior is unspecified.

99270 **Identification Keywords**

99271 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing  
 99272 identification keywords with their value wherever they occur. The following keywords may be  
 99273 used in the text stored in an SCCS file:

99274 **%M%** Module name: either the value of the **m** flag in the file, or if absent, the name of the  
 99275 SCCS file with the leading **s.** removed.

99276 **%I%** SCCS identification (SID) (**%R%.%L%** or **%R%.%L%.%B%.%S%**) of the retrieved  
 99277 text.

99278 **%R%** Release.

99279 **%L%** Level.

99280 **%B%** Branch.

99281 **%S%** Sequence.

99282 **%D%** Current date (*YY/MM/DD*).

99283 **%H%** Current date (*MM/DD/YY*).

99284 **%T%** Current time (*HH:MM:SS*).

99285 **%E%** Date newest applied delta was created (*YY/MM/DD*).

99286 **%G%** Date newest applied delta was created (*MM/DD/YY*).

99287 **%U%** Time newest applied delta was created (*HH:MM:SS*).

99288 **%Y%** Module type: value of the **t** flag in the SCCS file.

99289 **%F%** SCCS filename.

99290 **%P%** SCCS absolute pathname.

99291 **%Q%** The value of the **q** flag in the file.

99292 **%C%** Current line number. This keyword is intended for identifying messages output by  
 99293 the program, such as "this should not have happened" type errors. It is not  
 99294 intended to be used on every line to provide sequence numbers.

99295 **%Z%** The four-character string "**@ (#)**" recognizable by *what*.

99296 **%W%** A shorthand notation for constructing *what* strings:

99297 `%W%=%Z%%M%<tab>%I%`

99298 **%A%** Another shorthand notation for constructing *what* strings:

99299 `%A%=%Z%%Y%%M%%I%%Z%`

99300 **EXIT STATUS**

99301 The following exit values shall be returned:

99302 0 Successful completion.

99303 >0 An error occurred.

99304 **CONSEQUENCES OF ERRORS**

99305 Default.

99306 **APPLICATION USAGE**

99307 Problems can arise if the system date and time have been modified (for example, put forward  
99308 and then back again, or unsynchronized clocks across a network) and can also arise when  
99309 different values of the *TZ* environment variable are used.

99310 Problems of a similar nature can also arise for the operation of the *delta* utility, which compares  
99311 the previous file body against the working file as part of its normal operation.

99312 **EXAMPLES**

99313 None.

99314 **RATIONALE**

99315 None.

99316 **FUTURE DIRECTIONS**

99317 If this utility is directed to display a pathname that contains any bytes that have the encoded  
99318 value of a <newline> character when <newline> is a terminator or separator in the output  
99319 format being used, implementations are encouraged to treat this as an error. A future version of  
99320 this standard may require implementations to treat this as an error.

99321 If this utility is directed to create a new directory entry that contains any bytes that have the  
99322 encoded value of a <newline> character, implementations are encouraged to treat this as an  
99323 error. A future version of this standard may require implementations to treat this as an error.

99324 **SEE ALSO**

99325 *admin, delta, prs, what*

99326 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

99327 **CHANGE HISTORY**

99328 First released in Issue 2.

99329 **Issue 5**

99330 A correction is made to the first format string in STDOUT.

99331 The interpretation of the *YY* component of the *-c cutoff* argument is noted.

99332 **Issue 6**

99333 The obsolescent SYNOPSIS is removed, removing the *-lp* option.

99334 The normative text is reworded to avoid use of the term “must” for application requirements.

99335 The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

99336 The Open Group Corrigendum U048/1 is applied.

99337 The Open Group Interpretation PIN4C.00014 is applied.

99338 The Open Group Base Resolution bwg2001-007 is applied as follows:

- 99339 • The EXTENDED DESCRIPTION section is updated to make partial SID handling
- 99340 unspecified, reflecting common usage, and to clarify SID ranges.
- 99341 • New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections
- 99342 regarding how the system date and time may be taken into account.

99343 • The *TZ* environment variable is added to the ENVIRONMENT VARIABLES section.

99344 **Issue 7**

99345 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

99346 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0104 [584] is applied.

99347 **Issue 8**

99348 Austin Group Defect 251 is applied, encouraging implementations to behave as follows:

99349 a. Report an error if a utility is directed to display a pathname that contains any bytes that  
99350 have the encoded value of a <newline> character when <newline> is a terminator or  
99351 separator in the output format being used.

99352 b. Disallow the creation of filenames containing any bytes that have the encoded value of a  
99353 <newline> character.

99354 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

99355 **NAME**99356 `getconf` — get configuration values99357 **SYNOPSIS**99358 `getconf [-v specification] system_var`99359 `getconf [-v specification] path_var pathname`99360 **DESCRIPTION**99361 In the first synopsis form, the `getconf` utility shall write to the standard output the value of the  
99362 variable specified by the `system_var` operand.99363 In the second synopsis form, the `getconf` utility shall write to the standard output the value of the  
99364 variable specified by the `path_var` operand for the path specified by the `pathname` operand.99365 The value of each configuration variable shall be determined as if it were obtained by calling the  
99366 function from which it is defined to be available by this volume of POSIX.1-2024 or by the  
99367 System Interfaces volume of POSIX.1-2024 (see the OPERANDS section). The value shall reflect  
99368 conditions in the current operating environment.99369 **OPTIONS**99370 The `getconf` utility shall conform to XBD [Section 12.2](#) (on page 215).

99371 The following option shall be supported:

99372 `-v specification`99373 Indicate a specific specification and version for which configuration variables shall  
99374 be determined. If this option is not specified, the values returned correspond to an  
99375 implementation default conforming compilation environment.

99376 If the command:

99377 `getconf _POSIX_V8_ILP32_OFF32`99378 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of  
99379 the form:99380 `getconf -v POSIX_V8_ILP32_OFF32 ...`99381 determine values for configuration variables corresponding to the  
99382 POSIX\_V8\_ILP32\_OFF32 compilation environment specified in [c17](#), the  
99383 EXTENDED DESCRIPTION.

99384 If the command:

99385 `getconf _POSIX_V8_ILP32_OFFBIG`99386 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of  
99387 the form:99388 `getconf -v POSIX_V8_ILP32_OFFBIG ...`99389 determine values for configuration variables corresponding to the  
99390 POSIX\_V8\_ILP32\_OFFBIG compilation environment specified in [c17](#), the  
99391 EXTENDED DESCRIPTION.

99392 If the command:

99393 `getconf _POSIX_V8_LP64_OFF64`99394 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of  
99395 the form:

99396 `getconf -v POSIX_V8_LP64_OFF64 ...`  
 99397 determine values for configuration variables corresponding to the  
 99398 `POSIX_V8_LP64_OFF64` compilation environment specified in *c17*, the  
 99399 EXTENDED DESCRIPTION.

99400 If the command:

99401 `getconf _POSIX_V8_LPBIG_OFFBIG`

99402 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of  
 99403 the form:

99404 `getconf -v POSIX_V8_LPBIG_OFFBIG ...`

99405 determine values for configuration variables corresponding to the  
 99406 `POSIX_V8_LPBIG_OFFBIG` compilation environment specified in *c17*, the  
 99407 EXTENDED DESCRIPTION.

## 99408 OPERANDS

99409 The following operands shall be supported:

99410 *path\_var* A name of a configuration variable. All of the variables in the Variable column of  
 99411 the table in the DESCRIPTION of the *fpathconf()* function defined in the System  
 99412 Interfaces volume of POSIX.1-2024, without the enclosing braces, shall be  
 99413 supported. The implementation may add other local variables.

99414 *pathname* A pathname for which the variable specified by *path\_var* is to be determined.

99415 *system\_var* A name of a configuration variable. All of the following variables shall be  
 99416 supported:

- 99417 • The names, without the enclosing braces, in the Variable column of the table
- 99418 in the DESCRIPTION of the *sysconf()* function in the System Interfaces
- 99419 volume of POSIX.1-2024, except for the entries corresponding to
- 99420 `_SC_CLK_TCK`, `_SC_GETGR_R_SIZE_MAX`, `_SC_GETPW_R_SIZE_MAX`,
- 99421 `_SC_NPROCESSORS_CONF`, `_SC_NPROCESSORS_ONLN`, and `_SC_NSIG`.

99422 For compatibility with earlier versions, the following variable names shall  
 99423 also be supported:

99424 `POSIX2_C_BIND`  
 99425 `POSIX2_C_DEV`  
 99426 `POSIX2_CHAR_TERM`  
 99427 `POSIX2_FORT_RUN`  
 99428 `POSIX2_LOCALEDEF`  
 99429 `POSIX2_SW_DEV`  
 99430 `POSIX2_UPE`  
 99431 `POSIX2_VERSION`

99432 and shall be equivalent to the same name prefixed with an `<underscore>`.  
 99433 This requirement may be removed in a future version.

- 99434 • The names `NPROCESSORS_CONF` and `NPROCESSORS_ONLN`. The values
- 99435 of these configuration variables shall be determined as if they were obtained
- 99436 by calling the function *sysconf()* with the argument
- 99437 `_SC_NPROCESSORS_CONF` or `_SC_NPROCESSORS_ONLN`, respectively.

99438 • The names of the symbolic constants used as the *name* argument of the  
 99439 *confstr()* function in the System Interfaces volume of POSIX.1-2024, without  
 99440 the *\_CS\_* prefix.

99441 • The names of the symbolic constants listed under the headings “Maximum  
 99442 Values” and “Minimum Values” in the description of the **<limits.h>** header  
 99443 in the Base Definitions volume of POSIX.1-2024, without the enclosing  
 99444 braces.

99445 For compatibility with earlier versions, the following variable names shall  
 99446 also be supported:

99447 POSIX2\_BC\_BASE\_MAX  
 99448 POSIX2\_BC\_DIM\_MAX  
 99449 POSIX2\_BC\_SCALE\_MAX  
 99450 POSIX2\_BC\_STRING\_MAX  
 99451 POSIX2\_COLL\_WEIGHTS\_MAX  
 99452 POSIX2\_EXPR\_NEST\_MAX  
 99453 POSIX2\_LINE\_MAX  
 99454 POSIX2\_RE\_DUP\_MAX

99455 and shall be equivalent to the same name prefixed with an *<underscore>*.  
 99456 This requirement may be removed in a future version.

99457 The implementation may add other local values.

#### 99458 **STDIN**

99459 Not used.

#### 99460 **INPUT FILES**

99461 None.

#### 99462 **ENVIRONMENT VARIABLES**

99463 The following environment variables shall affect the execution of *getconf*:

99464 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 99465 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 99466 variables used to determine the values of locale categories.)

99467 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 99468 internationalization variables.

99469 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 99470 characters (for example, single-byte as opposed to multi-byte characters in  
 99471 arguments).

99472 *LC\_MESSAGES*

99473 Determine the locale that should be used to affect the format and contents of  
 99474 diagnostic messages written to standard error.

99475 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

#### 99476 **ASYNCHRONOUS EVENTS**

99477 Default.

**99478 STDOUT**

99479 If the specified variable is defined on the system and its value is described to be available from  
 99480 the *confstr()* function defined in the System Interfaces volume of POSIX.1-2024, its value shall be  
 99481 written in the following format:

99482 `"%s\n", <value>`

99483 Otherwise, if the specified variable is defined on the system, its value shall be written in the  
 99484 following format:

99485 `"%d\n", <value>`

99486 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the  
 99487 following format:

99488 `"undefined\n"`

99489 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

**99490 STDERR**

99491 The standard error shall be used only for diagnostic messages.

**99492 OUTPUT FILES**

99493 None.

**99494 EXTENDED DESCRIPTION**

99495 None.

**99496 EXIT STATUS**

99497 The following exit values shall be returned:

99498 0 The specified variable is valid and information about its current state was written  
 99499 successfully.

99500 >0 An error occurred.

**99501 CONSEQUENCES OF ERRORS**

99502 Default.

**99503 APPLICATION USAGE**

99504 None.

**99505 EXAMPLES**

99506 The following example illustrates the value of {NGROUPS\_MAX}:

99507 `getconf NGROUPS_MAX`

99508 The following example illustrates the value of {NAME\_MAX} for a specific directory:

99509 `getconf NAME_MAX /usr`

99510 The following example shows how to deal more carefully with results that might be unspecified:

```
99511 if value=$(getconf PATH_MAX /usr); then
99512     if [ "$value" = "undefined" ]; then
99513         echo PATH_MAX in /usr is indeterminate.
99514     else
99515         echo PATH_MAX in /usr is $value.
99516     fi
99517 else
99518     echo Error in getconf.
99519 fi
```



99520 **RATIONALE**

99521 The original need for this utility, and for the *confstr()* function, was to provide a way of finding  
 99522 the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be  
 99523 modified by the user to include directories that could contain utilities replacing the standard  
 99524 utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable  
 99525 value that contains the correct search path for the standard utilities. It was later suggested that  
 99526 access to the other variables described in this volume of POSIX.1-2024 could also be useful to  
 99527 applications.

99528 This functionality of *getconf* would not be adequately subsumed by another command such as:

99529 `grep var /etc/conf`

99530 because such a strategy would provide correct values for neither those variables that can vary at  
 99531 runtime, nor those that can vary depending on the path.

99532 Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid,  
 99533 but not defined on the system. The output string "undefined" is now used to specify this case  
 99534 with exit code 0 because so many things depend on an exit code of zero when an invoked utility  
 99535 is successful.

99536 **FUTURE DIRECTIONS**

99537 None.

99538 **SEE ALSO**

99539 *c17*

99540 XBD Chapter 8 (on page 167), Section 12.2 (on page 215), <limits.h>

99541 XSH *confstr()*, *fpathconf()*, *sysconf()*, *system()*

99542 **CHANGE HISTORY**

99543 First released in Issue 4.

99544 **Issue 5**

99545 In the OPERANDS section:

- 99546 • {NL\_MAX} is changed to {NL\_NMAX}.
- 99547 • Entries beginning NL\_ are deleted from the list of standard configuration variables.
- 99548 • The list of variables previously marked UX is merged with the list marked EX.
- 99549 • Operands are added to support new Option Groups.
- 99550 • Operands are added so that *getconf* can determine supported programming environments.

99551 **Issue 6**

99552 The Open Group Corrigendum U029/4 is applied, correcting the example command in the last  
 99553 paragraph of the OPTIONS section.

99554 The following new requirements on POSIX implementations derive from alignment with the  
 99555 Single UNIX Specification:

- 99556 • Operands are added to determine supported programming environments.

99557 This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard. Specifically,  
 99558 new macros for *c99* programming environments are introduced.

99559 XSI marked *system\_var* (XBS5\_\*) values are marked LEGACY.

99560 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/27 is applied, correcting the descriptions

99561 of *path\_var* and *system\_var* in the OPERANDS section.

99562 **Issue 7**

99563 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

99564 The EXAMPLES section is corrected.

99565 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0091 [125] is applied.

99566 **Issue 8**

99567 Austin Group Defect 339 is applied, adding the *system\_var* names NPROCESSORS\_CONF and  
99568 NPROCESSORS\_ONLN.

99569 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

99570 Austin Group Defect 1330 is applied, removing obsolescent interfaces and changing ``\_V7\_'' to  
99571 ``\_V8\_''.

99572 **NAME**

99573 getopts — parse utility options

99574 **SYNOPSIS**99575 `getopts optstring name [param...]`99576 **DESCRIPTION**

99577 The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall  
 99578 support the Utility Syntax Guidelines 3 to 10, inclusive, described in XBD Section 12.2 (on page  
 99579 215).

99580 When the shell is first invoked, the shell variable *OPTIND* shall be initialized to 1. Each time  
 99581 *getopts* is invoked, it shall place the value of the next option found in the parameter list in the  
 99582 shell variable specified by the *name* operand and the shell variable *OPTIND* shall be set as  
 99583 follows:

- 99584 • When *getopts* successfully parses an option that takes an option-argument (that is, a  
 99585 character followed by <colon> in *optstring*, and exit status is 0), the value of *OPTIND* shall  
 99586 be the integer index of the next element of the parameter list (if any; see OPERANDS  
 99587 below) to be searched for an option character. Index 1 identifies the first element of the  
 99588 parameter list.
- 99589 • When *getopts* reports end of options (that is, when exit status is 1), the value of *OPTIND*  
 99590 shall be the integer index of the next element of the parameter list (if any).
- 99591 • In all other cases, the value of *OPTIND* is unspecified, but shall encode the information  
 99592 needed for the next invocation of *getopts* to resume parsing options after the option just  
 99593 parsed.

99594 When the option requires an option-argument, the *getopts* utility shall place it in the shell  
 99595 variable *OPTARG*. If no option was found, or if the option that was found does not have an  
 99596 option-argument, *OPTARG* shall be unset.

99597 If an option character not contained in the *optstring* operand is found where an option character  
 99598 is expected, the shell variable specified by *name* shall be set to the <question-mark> ('?')  
 99599 character. In this case, if the first character in *optstring* is a <colon> (':'), the shell variable  
 99600 *OPTARG* shall be set to the option character found, but no output shall be written to standard  
 99601 error; otherwise, the shell variable *OPTARG* shall be unset and a diagnostic message shall be  
 99602 written to standard error. This condition shall be considered to be an error detected in the way  
 99603 arguments were presented to the invoking application, but shall not be an error in *getopts*  
 99604 processing.

99605 If an option-argument is missing:

- 99606 • If the first character of *optstring* is a <colon>, the shell variable specified by *name* shall be  
 99607 set to the <colon> character and the shell variable *OPTARG* shall be set to the option  
 99608 character found.
- 99609 • Otherwise, the shell variable specified by *name* shall be set to the <question-mark>  
 99610 character, the shell variable *OPTARG* shall be unset, and a diagnostic message shall be  
 99611 written to standard error. This condition shall be considered to be an error detected in the  
 99612 way arguments were presented to the invoking application, but shall not be an error in  
 99613 *getopts* processing; a diagnostic message shall be written as stated, but the exit status shall  
 99614 be zero.

99615 When the end of options is encountered, the *getopts* utility shall exit with a return value of one;  
 99616 the shell variable *OPTIND* shall be set to the index of the argument containing the first operand  
 99617 in the parameter list, or the value 1 plus the number of elements in the parameter list if there are

99618 no operands in the parameter list; the *name* variable shall be set to the <question-mark>  
 99619 character. Any of the following shall identify the end of options: the first "--" element of the  
 99620 parameter list that is not an option-argument, finding an element of the parameter list that is not  
 99621 an option-argument and does not begin with a '-', or encountering an error.

99622 The shell variables *OPTIND* and *OPTARG* shall not be exported by default. An error in setting  
 99623 any of these variables (such as if *name* has previously been marked *readonly*) shall be considered  
 99624 an error of *getopts* processing, and shall result in a return value greater than one.

99625 The *getopts* utility can affect *OPTIND*, *OPTARG*, and the shell variable specified by the *name*  
 99626 operand, within the current shell execution environment; see [Section 2.13](#) (on page 2522).

99627 If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the  
 99628 current positional parameters or new *param* values. Any other attempt to invoke *getopts* multiple  
 99629 times in a single shell execution environment with parameters (positional parameters or *param*  
 99630 operands) that are not the same in all invocations, or with an *OPTIND* value modified by the  
 99631 application to be a value other than 1, produces unspecified results.

#### 99632 OPTIONS

99633 None.

#### 99634 OPERANDS

99635 The following operands shall be supported:

99636 *optstring* A string containing the option characters recognized by the utility invoking *getopts*.  
 99637 If a character is followed by a <colon>, the option shall be expected to have an  
 99638 argument, which should be supplied as a separate argument. Applications should  
 99639 specify an option character and its option-argument as separate arguments, but  
 99640 *getopts* shall interpret the characters following an option character requiring  
 99641 arguments as an argument whether or not this is done. An explicit null option-  
 99642 argument need not be recognized if it is not supplied as a separate argument when  
 99643 *getopts* is invoked. (See also the *getopt()* function defined in the System Interfaces  
 99644 volume of POSIX.1-2024.) The characters <question-mark> and <colon> shall not  
 99645 be used as option characters by an application. The use of other option characters  
 99646 that are not alphanumeric produces unspecified results. Whether or not the option-  
 99647 argument is supplied as a separate argument from the option character, the value  
 99648 in *OPTARG* shall only be the characters of the option-argument. The first character  
 99649 in *optstring* determines how *getopts* behaves if an option character is not known or  
 99650 an option-argument is missing.

99651 *name* The name of a shell variable that shall be set by the *getopts* utility to the option  
 99652 character that was found.

99653 By default, the list of parameters parsed by the *getopts* utility shall be the positional parameters  
 99654 currently set in the invoking shell environment ("\${@}"). If *param* operands are given, they shall  
 99655 be parsed instead of the positional parameters. Note that the next element of the parameter list  
 99656 need not exist; in this case, *OPTIND* will be set to \$#+1 or the number of *param* operands plus 1.

#### 99657 STDIN

99658 Not used.

#### 99659 INPUT FILES

99660 None.

99661 **ENVIRONMENT VARIABLES**99662 The following environment variables shall affect the execution of *getopts*:

99663 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 99664 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 99665 variables used to determine the values of locale categories.)

99666 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 99667 internationalization variables.

99668 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 99669 characters (for example, single-byte as opposed to multi-byte characters in  
 99670 arguments and input files).

99671 *LC\_MESSAGES*

99672 Determine the locale that should be used to affect the format and contents of  
 99673 diagnostic messages written to standard error.

99674 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

99675 *OPTIND* This variable shall be used by the *getopts* utility as the index of the next argument  
 99676 to be processed.

99677 **ASYNCHRONOUS EVENTS**

99678 Default.

99679 **STDOUT**

99680 Not used.

99681 **STDERR**

99682 Whenever an error is detected and the first character in the *optstring* operand is not a <colon>  
 99683 (' : '), a diagnostic message shall be written to standard error with the following information in  
 99684 an unspecified format:

- 99685 • The invoking program name shall be identified in the message. The invoking program  
 99686 name shall be the value of the shell special parameter 0 (see Section 2.5.2, on page 2479) at  
 99687 the time the *getopts* utility is invoked. A name equivalent to:

99688 `basename "$0"`

99689 may be used.

- 99690 • If an option is found that was not specified in *optstring*, this error is identified and the  
 99691 invalid option character shall be identified in the message.

- 99692 • If an option requiring an option-argument is found, but an option-argument is not found,  
 99693 this error shall be identified and the invalid option character shall be identified in the  
 99694 message.

99695 **OUTPUT FILES**

99696 None.

99697 **EXTENDED DESCRIPTION**

99698 None.

99699 **EXIT STATUS**

99700 The following exit values shall be returned:

99701 0 An option, specified or unspecified by *optstring*, was found.

99702           1 The end of options was encountered.

99703           >1 An error occurred.

## 99704 CONSEQUENCES OF ERRORS

99705           Default.

## 99706 APPLICATION USAGE

99707           This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

99708           Since *getopts* affects the current shell execution environment, it is generally provided as a shell  
99709           regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
99710           of the following:

```
99711           (getopts abc value "$@")
99712           nohup getopts ...
99713           find . -exec getopts ... \;
```

99714           it does not affect the shell variables in the caller's environment.

99715           Note that shell functions share *OPTIND* with the calling shell even though the positional  
99716           parameters are changed. If the calling shell and any of its functions uses *getopts* to parse  
99717           arguments, the results are unspecified.

## 99718 EXAMPLES

99719           The following example script parses and displays its arguments:

```
99720           aflag=
99721           bflag=
99722           while getopts ab: name
99723           do
99724               case $name in
99725               a)     aflag=1;;
99726               b)     bflag=1
99727                     bval="$OPTARG";;
99728               ?)     printf "Usage: %s: [-a] [-b value] args\n" $0
99729                     exit 2;;
99730               esac
99731           done
99732           if [ -n "$aflag" ]; then
99733               printf "Option -a specified\n"
99734           fi
99735           if [ -n "$bflag" ]; then
99736               printf 'Option -b "%s" specified\n' "$bval"
99737           fi
99738           shift $((OPTIND - 1))
99739           printf "Remaining arguments are: %s\n" "$*"
```

## 99740 RATIONALE

99741           The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles  
99742           option-arguments containing <blank> characters.

99743           The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it  
99744           does not affect the execution of *getopts*; it is one of the few "output-only" variables used by the  
99745           standard utilities.

99746           The <colon> is not allowed as an option character because that is not historical behavior, and it  
99747           violates the Utility Syntax Guidelines. The <colon> is now specified to behave as in the

99748 KornShell version of the *getopts* utility; when used as the first character in the *optstring* operand,  
 99749 it disables diagnostics concerning missing option-arguments and unexpected option characters.  
 99750 This replaces the use of the *OPTERR* variable that was specified in an early proposal.

99751 Although a leading *<plus-sign>* in *optstring* is required to have no effect on the behavior of  
 99752 *getopt()*, this standard intentionally allows implementations of the *getopts* utility to use a leading  
 99753 *<plus-sign>* as an extension that alters behavior. In fact, a *<plus-sign>* anywhere in the *optstring*  
 99754 in the *getopts* utility produces unspecified behavior.

99755 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function  
 99756 are not fully specified because implementations with superior (“friendlier”) formats objected to  
 99757 the formats used by some historical implementations. The standard developers considered it  
 99758 important that the information in the messages used be uniform between *getopts* and *getopt()*.  
 99759 Exact duplication of the messages might not be possible, particularly if a utility is built on  
 99760 another system that has a different *getopt()* function, but the messages must have specific  
 99761 information included so that the program name, invalid option character, and type of error can  
 99762 be distinguished by a user.

99763 Only a rare application program intercepts a *getopts* standard error message and wants to parse  
 99764 it. Therefore, implementations are free to choose the most usable messages they can devise. The  
 99765 following formats are used by many historical implementations:

99766 "%s: illegal option -- %c\n", *<program name>*, *<option character>*

99767 "%s: option requires an argument -- %c\n", *<program name>*, \  
 99768 *<option character>*

99769 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,  
 99770 frequently not even indicating the option character found in error.

#### 99771 FUTURE DIRECTIONS

99772 None.

#### 99773 SEE ALSO

99774 [Section 2.5.2](#) (on page 2479)

99775 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

99776 [XSH \*getopt\(\)\*](#)

#### 99777 CHANGE HISTORY

99778 First released in Issue 4.

#### 99779 Issue 6

99780 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 99781 Issue 7

99782 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0092 [159] is applied.

#### 99783 Issue 8

99784 Austin Group Defect 191 is applied, adding a paragraph about leading *<plus-sign>* to the  
 99785 RATIONALE section.

99786 Austin Group Defect 367 is applied, requiring that *getopts* distinguishes between encountering  
 99787 the end of options and an error occurring, setting its exit status to one and greater than one,  
 99788 respectively.

99789 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
 99790 this utility is required to be intrinsic.

- 99791 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 99792 Austin Group Defect 1442 is applied, changing the EXAMPLES section.
- 99793 Austin Group Defect 1784 is applied, clarifying several aspects of *getopts* behavior and changing
- 99794 the value of *OPTIND* to be unspecified in some circumstances.



99795 **NAME**99796 `gettext, ngettext` — retrieve text string from messages object99797 **SYNOPSIS**99798 `gettext [-e|-E] [-d textdomain] [textdomain] msgid`99799 `gettext [-e|-E] [-n] -s [-d textdomain] msgid...`99800 `ngettext [-e|-E] [-d textdomain] [textdomain] msgid msgid_plural n`99801 **DESCRIPTION**99802 The *gettext* and *ngettext* utilities shall write to standard output the message string(s) that would  
99803 result from the following calls to functions defined in the System Interfaces volume of  
99804 POSIX.1-2024:

```

99805 if (textdomainname == NULL || textdomainname[0] == '\0')
99806     message_string = msgid;
99807 else {
99808     setlocale(LC_ALL, "");
99809     if (textdomaindir != NULL)
99810         bindtextdomain(textdomainname, textdomaindir);
99811     if (msgid_plural == NULL)
99812         message_string = dgettext(textdomainname, msgid);
99813     else
99814         message_string = dngettext(textdomainname, msgid, msgid_plural, n);
99815 }

```

99816 where:

- 99817 • The *textdomaindir* variable is a string containing the value of the *TEXTDOMAINDIR*  
99818 environment variable, if set and not empty, or is NULL otherwise.
- 99819 • The *textdomainname* variable is a string containing the text domain name obtained from, in  
99820 decreasing order of precedence:
  - 99821 — The optional operand *textdomain*, if present
  - 99822 — The *-d textdomain* option, if specified
  - 99823 — The *TEXTDOMAIN* environment variable, if set and not empty
- 99824 If the text domain name cannot be obtained from these sources, the *textdomainname*  
99825 variable is NULL.
- 99826 • If the *-s* option of *gettext* is not specified and for the *ngettext* utility, the *msgid* variable is a  
99827 string containing:
  - 99828 — The value of the *msgid* operand, if the *-E* option is specified
  - 99829 — The value of the *msgid* operand with C-language escape sequences processed (see  
99830 below), if the *-e* option is specified
  - 99831 — The value of the *msgid* operand with C-language escape sequences optionally  
99832 processed (see below), otherwise
- 99833 • If the *-s* option of *gettext* is specified, the *msgid* variable is a string containing:
  - 99834 — The value of each *msgid* operand in turn, if the *-E* option is specified or neither the *-e*  
99835 nor the *-E* option is specified

- 99836 — The value of each *msgid* operand in turn with C-language escape sequences  
99837 processed (see below), if the `-e` option is specified
- 99838 • For the *gettext* utility, the *msgid\_plural* variable is NULL. For the *ngettext* utility, the  
99839 *msgid\_plural* variable is a string containing:
  - 99840 — The value of the *msgid\_plural* operand, if the `-E` option is specified
  - 99841 — The value of the *msgid\_plural* operand with C-language escape sequences processed  
99842 (see below), if the `-e` option is specified
  - 99843 — The value of the *msgid\_plural* operand with C-language escape sequences optionally  
99844 processed (see below), otherwise
- 99845 • For the *gettext* utility, the *n* variable is 1 (one). For the *ngettext* utility the *n* variable is the *n*  
99846 operand, parsed as an integer as if by using the *strtoul()* function with a *base* argument of  
99847 10.

99848 When C-language escape sequences are processed, they shall be processed as specified for  
99849 character string literals in the ISO C standard, except that *universal-character-name* escape  
99850 sequences need not be supported. Implementations may also support a `<backslash> 'c'` escape  
99851 sequence; if supported, the `'\c'` and all characters following it shall be removed and, if the `-s`  
99852 option is specified, the behavior shall be as if the `-n` option is also specified.

99853 For the *ngettext* utility, and for the *gettext* utility if the `-s` option is not specified, the resulting  
99854 message string shall be written to standard output. If the `-s` option of *gettext* is specified, the  
99855 resulting message string for each *msgid* shall be written to standard output with consecutive  
99856 message strings separated by a single `<space>` character and, if the `-n` option is not specified, a  
99857 `<newline>` shall be written after the last message string. If the `-s` and `-n` options are specified,  
99858 the trailing `<newline>` shall be omitted.

99859 Under conditions where the *textdomainname* variable in the above code would be NULL, these  
99860 utilities may write a diagnostic message to standard error and exit with non-zero status.

## 99861 OPTIONS

99862 These utilities shall conform to XBD [Section 12.2](#) (on page 215).

99863 The following options shall be supported:

- 99864 `-d textdomain`  
99865 Retrieve the translated message from the domain *textdomain*, if *textdomain* is not  
99866 specified as an operand.
- 99867 `-e` Process C-language escape sequences in *msgid* and *msgid\_plural* operands.
- 99868 `-E` Do not process C-language escape sequences in *msgid* and *msgid\_plural* operands.

99869 The *gettext* utility shall also support the following options:

- 99870 `-n` Modify the behavior of the `-s` option such that a `<newline>` is not appended to the  
99871 output.
- 99872 `-s` Separate the message strings obtained from each *msgid* operand with `<space>`  
99873 characters in the output, and (if `-n` is not also specified) append a `<newline>` to the  
99874 output.

99875 If neither of the mutually exclusive `-e` and `-E` options is specified, it is unspecified which is the  
99876 default, except that if the `-s` option of *gettext* is specified then `-E` shall be the default.

99877 **OPERANDS**

99878 The following operands shall be supported:

99879 *textdomain* A text domain name used to retrieve the translated message. This shall override  
99880 the specification by the `-d` option, if present.99881 *msgid* A key to retrieve the translated message.99882 *msgid\_plural* A default plural if no corresponding plural message can be found.99883 *n* A non-negative decimal integer to be used as the *n* argument to `dngettext()` (see the  
99884 DESCRIPTION).99885 **STDIN**

99886 Not used.

99887 **INPUT FILES**99888 The input files are messages object files (see *msgfmt*).99889 **ENVIRONMENT VARIABLES**99890 The following environment variables shall affect the execution of *gettext* and *ngettext*:99891 *LANG* Provide a default value for the internationalization variables that are unset or null.  
99892 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
99893 variables used to determine the values of locale categories.)99894 XSI *LANGUAGE* Determine the location of messages objects if *NLSPATH* is not set or the evaluation  
99895 of *NLSPATH* did not lead to a suitable messages object being found.99896 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
99897 internationalization variables.99898 *LC\_MESSAGES*99899 Determine the locale name used to locate messages objects, and the locale that  
99900 should be used to affect the format and contents of diagnostic messages written to  
99901 standard error.99902 XSI *NLSPATH* Determine the location of messages objects and message catalogs.99903 *TEXTDOMAIN*

99904 Specify the text domain name. (See XBD Section 3.386 (on page 88).)

99905 *TEXTDOMAINDIR*99906 XSI Specify the pathname to the messages object hierarchy. *NLSPATH* shall have  
99907 precedence over *TEXTDOMAINDIR*.99908 **ASYNCHRONOUS EVENTS**

99909 Default.

99910 **STDOUT**

99911 See DESCRIPTION.

99912 **STDERR**

99913 The standard error shall be used only for diagnostic messages.

99914 **OUTPUT FILES**

99915 None.

99916 **EXTENDED DESCRIPTION**

99917 None.

99918 **EXIT STATUS**

99919 The following exit values shall be returned:

99920 0 Successful completion.

99921 &gt;0 An error occurred.

99922 **CONSEQUENCES OF ERRORS**

99923 Default.

99924 **APPLICATION USAGE**

99925 Since it is unspecified which of the `-e` or `-E` options is the default, except when the `-s` option of  
 99926 `gettext` is specified, portable applications need to ensure that `-e`, `-E`, or (for `gettext`) `-s` is specified  
 99927 whenever a `msgid` or `msgid_plural` operand contains, or might contain, a `<backslash>` character.

99928 Note that, unless the `-s` option of `gettext` is specified without `-n`, the message(s) written to  
 99929 standard output are not followed by a `<newline>`. (Therefore the output only ends with a  
 99930 `<newline>` if the last message ends with one.)

99931 Both `msgid` and `msgid_plural` should be properly quoted for the shell.99932 **EXAMPLES**

99933 The following examples assume that the following portable messages object source file (dot-po  
 99934 file) has been compiled to a valid file **mail.mo** by the `msgfmt` utility. See the EXTENDED  
 99935 DESCRIPTION section of the `msgfmt` utility for a description of the dot-po file format.

```
99936 msgid ""
99937 msgstr ""
99938 "Content-Type: text/plain; charset=utf-8\n"
99939 "Plural-Forms: nplurals=4; plural=n==1?0: (n>1&&n<=10)?1: (n==0)?2:3;\n"
```

```
99940 msgid "recipient"
99941 msgid_plural "recipients"
99942 msgstr[0] "1 recipient"
99943 msgstr[1] "2 to 10 recipients"
99944 msgstr[2] "no recipients"
99945 msgstr[3] "more than 10 recipients"
```

```
99946 msgid "%d attachment\n"
99947 msgid_plural "%d attachments\n"
99948 msgstr[0] "1 (%d) attachment\n"
99949 msgstr[1] "2 to 10 (%d) attachments\n"
99950 msgstr[2] "no (%d) attachments\n"
99951 msgstr[3] "more than 10 (%d) attachments\n"
```

99952 They also assume that **mail.mo** is installed in the directory that `gettext` and `ngettext` search for the  
 99953 current locale. See the **OPTIONS** and **ENVIRONMENT VARIABLES** sections above and the  
 99954 description of `gettext()` for details on how this search is performed.

99955 The command

99956 `ngettext -d mail recipient recipients 0`

99957 will write "no recipients".

99958 The command

99959 `ngettext -d mail recipient recipients 1`  
99960 will write "1 recipient".  
99961 The command  
99962 `ngettext -d mail recipient recipients 5`  
99963 will write "2 to 10 recipients".  
99964 The command  
99965 `ngettext -d mail recipient recipients 11`  
99966 will write "more than 10 recipients".  
99967 The command  
99968 `ngettext -d mail Call Calls 1`  
99969 will write "Call". Note that "Call" is not in the messages object.  
99970 The command  
99971 `ngettext -d mail Call Calls 0`  
99972 will write "Calls".  
99973 The command  
99974 `ngettext -d mail Call Calls 10`  
99975 will write "Calls".  
99976 The command  
99977 `ngettext -ed mail "%d attachment\n" "%d attachments\n" 1`  
99978 will write the same as  
99979 `printf "1 (%d) attachment\n"`  
99980 (i.e. "1 (%d) attachment" followed by a <newline> character). The output of *ngettext* can be  
99981 used as a format string for *printf*.  
99982 The command  
99983 `printf "$ (ngettext -ed mail "%d attachment\n" "%d attachments\n" 1)" 10`  
99984 will write the same as  
99985 `printf "1 (%d) attachment\n" 10`  
99986 (i.e. "1 (10) attachment" followed by a <newline> character).  
99987 The command  
99988 `ngettext -e -d mail "\tsubject\n" "\tsubjects\n" 0`  
99989 will write the same as  
99990 `printf "\tsubjects\n"`  
99991 (i.e. a <tab> character, followed by "subjects" followed by a <newline> character). Note that  
99992 "\tsubject\n" is not in the messages object.  
99993 The command

99994 `printf "%s\n" "$(gettext -E -d mail "subject" "subjects" 0) "`  
99995 will write the same as  
99996 `printf "subjects\n"`  
99997 (i.e. "subjects" followed by a <newline> character). Note that "subject" is not in the  
99998 messages object.  
99999 The command  
100000 `gettext -s -d mail "recipient"`  
100001 will write "1 recipient" followed by a <newline> character.  
100002 The command  
100003 `gettext -s -n -d mail "recipient"`  
100004 will write "1 recipient" without a <newline> character.  
100005 **RATIONALE**  
100006 Historical implementations did not support the '\a' C-language escape sequence. This  
100007 standard requires it to be supported for consistency with other utilities that support the table in  
100008 XBD [Chapter 5](#) (on page 113).  
100009 Unlike other standard utilities, the behavior of *gettext* and *gettext* is not undefined when  
100010 *NLSPATH* overrides the system default path; see XBD [Section 8.2](#) (on page 169). This is so that  
100011 applications can use these utilities to obtain message strings from messages objects in other  
100012 locations. However, it also means that they need to be implemented in such a way that they do  
100013 not do anything that would result in undefined behavior when they need to write a diagnostic  
100014 message. In particular, they should not use a string obtained from a message catalog or a  
100015 messages object as a format string (or should only do so after checking that the string contains  
100016 the correct conversions).  
100017 **FUTURE DIRECTIONS**  
100018 None.  
100019 **SEE ALSO**  
100020 *msgfmt*, *printf*  
100021 XBD [Chapter 7](#) (on page 127), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)  
100022 XSH *gettext*, *iconv()*, *setlocale()*  
100023 **CHANGE HISTORY**  
100024 First released in Issue 8.

100025 **NAME**100026 `grep` — search a file for a pattern100027 **SYNOPSIS**

```
100028  grep [-E|-F] [-c|-l|-q] [-insvx] -e pattern_list
100029      [-e pattern_list]... [-f pattern_file]... [file...]
100030  grep [-E|-F] [-c|-l|-q] [-insvx] [-e pattern_list]...
100031      -f pattern_file [-f pattern_file]... [file...]
100032  grep [-E|-F] [-c|-l|-q] [-insvx] pattern_list [file...]
```

100033 **DESCRIPTION**

100034 The `grep` utility shall search the input files, selecting lines matching one or more patterns; the  
 100035 types of patterns are controlled by the options specified. The patterns are specified by the `-e`  
 100036 option, `-f` option, or the `pattern_list` operand. The `pattern_list`'s value shall consist of one or more  
 100037 patterns separated by <newline> characters; the `pattern_file`'s contents shall consist of one or  
 100038 more patterns terminated by a <newline> character. By default, an input line shall be selected if  
 100039 any pattern, treated as an entire basic regular expression (BRE) as described in XBD [Section 9.3](#)  
 100040 (on page 181), matches any part of the line excluding the terminating <newline>; a null BRE  
 100041 shall match every line. By default, each selected input line shall be written to the standard  
 100042 output.

100043 Regular expression matching shall be based on text lines. Since a <newline> separates or  
 100044 terminates patterns (see the `-e` and `-f` options below), regular expressions cannot contain a  
 100045 <newline>. Similarly, since patterns are matched against individual lines (excluding the  
 100046 terminating <newline> characters) of the input, there is no way for a pattern to match a  
 100047 <newline> found in the input.

100048 **OPTIONS**100049 The `grep` utility shall conform to XBD [Section 12.2](#) (on page 215).

100050 The following options shall be supported:

100051 `-E` Match using extended regular expressions. Treat each pattern specified as an ERE,  
 100052 as described in XBD [Section 9.4](#) (on page 187). If any entire ERE pattern matches  
 100053 some part of an input line excluding the terminating <newline>, the line shall be  
 100054 matched. A null ERE shall match every line.

100055 `-F` Match using fixed strings. Treat each pattern specified as a string instead of a  
 100056 regular expression. If an input line contains any of the patterns as a contiguous  
 100057 sequence of bytes, the line shall be matched. A null string shall match every line.

100058 `-c` Write only a count of selected lines to standard output.

100059 `-e pattern_list`

100060 Specify one or more patterns to be used during the search for input. The  
 100061 application shall ensure that patterns in `pattern_list` are separated by a <newline>.  
 100062 A null pattern can be specified by two adjacent <newline> characters in  
 100063 `pattern_list`. Unless the `-E` or `-F` option is also specified, each pattern shall be  
 100064 treated as a BRE, as described in XBD [Section 9.3](#) (on page 181). Multiple `-e` and `-f`  
 100065 options shall be accepted by the `grep` utility. All of the specified patterns shall be  
 100066 used when matching lines, but the order of evaluation is unspecified.

100067 `-f pattern_file`

100068 Read one or more patterns from the file named by the pathname `pattern_file`.  
 100069 Patterns in `pattern_file` shall be terminated by a <newline>. A null pattern can be  
 100070 specified by an empty line in `pattern_file`. Unless the `-E` or `-F` option is also

- 100071 specified, each pattern shall be treated as a BRE, as described in XBD [Section 9.3](#)  
 100072 (on page 181).
- 100073 **-i** Perform pattern matching in a case-insensitive manner; see XBD [Section 9.2](#) (on  
 100074 page 180).
- 100075 **-l** (The letter ell.) Write only the names of files containing selected lines to standard  
 100076 output. Pathnames shall be written once per file searched. If the standard input is  
 100077 searched, a pathname of "(standard input)" shall be written, in the POSIX  
 100078 locale. In other locales, "standard input" may be replaced by something more  
 100079 appropriate in those locales.
- 100080 **-n** Precede each output line by its relative line number in the file, each file starting at  
 100081 line 1. The line number counter shall be reset for each file processed.
- 100082 **-q** Quiet. Nothing shall be written to the standard output, regardless of matching  
 100083 lines. Exit with zero status if an input line is selected.
- 100084 **-s** Suppress the error messages ordinarily written for nonexistent or unreadable files.  
 100085 Other error messages shall not be suppressed.
- 100086 **-v** Select lines not matching any of the specified patterns. If the **-v** option is not  
 100087 specified, selected lines shall be those that match any of the specified patterns.
- 100088 **-x** Consider only input lines that use all characters in the line excluding the  
 100089 terminating <newline> to match an entire fixed string or regular expression to be  
 100090 matching lines.

**100091 OPERANDS**

100092 The following operands shall be supported:

- 100093 *pattern\_list* Specify one or more patterns to be used during the search for input. This operand  
 100094 shall be treated as if it were specified as **-e pattern\_list**.
- 100095 *file* A pathname of a file to be searched for the patterns. If no *file* operands are  
 100096 specified, the standard input shall be used.

**100097 STDIN**

100098 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
 100099 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
 100100 the standard input shall not be used. See the INPUT FILES section.

**100101 INPUT FILES**

100102 The input files shall be text files.

**100103 ENVIRONMENT VARIABLES**

100104 The following environment variables shall affect the execution of *grep*:

- 100105 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 100106 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 100107 variables used to determine the values of locale categories.)
- 100108 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 100109 internationalization variables.
- 100110 *LC\_COLLATE*  
 100111 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 100112 character collating elements within regular expressions.



|        |                             |                                                                                                                                                                                                                                                            |
|--------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 100113 | <i>LC_CTYPE</i>             | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions. |
| 100114 |                             |                                                                                                                                                                                                                                                            |
| 100115 |                             |                                                                                                                                                                                                                                                            |
| 100116 |                             |                                                                                                                                                                                                                                                            |
| 100117 | <i>LC_MESSAGES</i>          |                                                                                                                                                                                                                                                            |
| 100118 |                             | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                               |
| 100119 |                             |                                                                                                                                                                                                                                                            |
| 100120 | XSI <i>NLSPATH</i>          | Determine the location of messages objects and message catalogs.                                                                                                                                                                                           |
| 100121 | <b>ASYNCHRONOUS EVENTS</b>  |                                                                                                                                                                                                                                                            |
| 100122 |                             | Default.                                                                                                                                                                                                                                                   |
| 100123 | <b>STDOUT</b>               |                                                                                                                                                                                                                                                            |
| 100124 |                             | If the <b>-l</b> option is in effect, the following shall be written for each file containing at least one selected input line:                                                                                                                            |
| 100125 |                             |                                                                                                                                                                                                                                                            |
| 100126 |                             | <code>"%s\n", &lt;file&gt;</code>                                                                                                                                                                                                                          |
| 100127 |                             | Otherwise, if more than one <i>file</i> argument appears, and <b>-q</b> is not specified, the <i>grep</i> utility shall prefix each output line by:                                                                                                        |
| 100128 |                             |                                                                                                                                                                                                                                                            |
| 100129 |                             | <code>"%s:", &lt;file&gt;</code>                                                                                                                                                                                                                           |
| 100130 |                             | The remainder of each output line shall depend on the other options specified:                                                                                                                                                                             |
| 100131 |                             | • If the <b>-c</b> option is in effect, the remainder of each output line shall contain:                                                                                                                                                                   |
| 100132 |                             | <code>"%d\n", &lt;count&gt;</code>                                                                                                                                                                                                                         |
| 100133 |                             | • Otherwise, if <b>-c</b> is not in effect and the <b>-n</b> option is in effect, the following shall be written to standard output:                                                                                                                       |
| 100134 |                             |                                                                                                                                                                                                                                                            |
| 100135 |                             | <code>"%d:", &lt;line number&gt;</code>                                                                                                                                                                                                                    |
| 100136 |                             | • Finally, the following shall be written to standard output:                                                                                                                                                                                              |
| 100137 |                             | <code>"%s", &lt;selected-line contents&gt;</code>                                                                                                                                                                                                          |
| 100138 | <b>STDERR</b>               |                                                                                                                                                                                                                                                            |
| 100139 |                             | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                             |
| 100140 | <b>OUTPUT FILES</b>         |                                                                                                                                                                                                                                                            |
| 100141 |                             | None.                                                                                                                                                                                                                                                      |
| 100142 | <b>EXTENDED DESCRIPTION</b> |                                                                                                                                                                                                                                                            |
| 100143 |                             | None.                                                                                                                                                                                                                                                      |
| 100144 | <b>EXIT STATUS</b>          |                                                                                                                                                                                                                                                            |
| 100145 |                             | The following exit values shall be returned:                                                                                                                                                                                                               |
| 100146 | 0                           | One or more lines were selected and the output specified in <b>STDOUT</b> was successfully written to standard output.                                                                                                                                     |
| 100147 |                             |                                                                                                                                                                                                                                                            |
| 100148 | 1                           | No lines were selected.                                                                                                                                                                                                                                    |
| 100149 | >1                          | An error occurred.                                                                                                                                                                                                                                         |

## 100150 CONSEQUENCES OF ERRORS

100151 If the `-q` option is specified, the exit status shall be zero if an input line is selected, even if an  
 100152 error was detected. Otherwise, default actions shall be performed.

## 100153 APPLICATION USAGE

100154 Care should be taken when using characters in *pattern\_list* that may also be meaningful to the  
 100155 command interpreter. It is safest to enclose the entire *pattern\_list* argument in single-quotes:

100156 ' . . . '

100157 The `-e pattern_list` option has the same effect as the *pattern\_list* operand, but is useful when  
 100158 *pattern\_list* begins with the <hyphen-minus> delimiter. It is also useful when it is more  
 100159 convenient to provide multiple patterns as separate arguments.

100160 Multiple `-e` and `-f` options are accepted and *grep* uses all of the patterns it is given while  
 100161 matching input text lines. (Note that the order of evaluation is not specified. If an  
 100162 implementation finds a null string as a pattern, it is allowed to use that pattern first, matching  
 100163 every line, and effectively ignore any other patterns.)

100164 The `-q` option provides a means of easily determining whether or not a pattern (or string) exists  
 100165 in a group of files. When searching several files, it provides a performance improvement  
 100166 (because it can quit as soon as it finds the first match) and requires less care by the user in  
 100167 choosing the set of files to supply as arguments (because it exits zero if it finds a match even if  
 100168 *grep* detected an access or read error on earlier *file* operands).

100169 When using *grep* to process pathnames, it is recommended that `LC_ALL`, or at least `LC_CTYPE`  
 100170 and `LC_COLLATE`, are set to `POSIX` or `C` in the environment, since pathnames can contain byte  
 100171 sequences that do not form valid characters in some locales, in which case the utility's behavior  
 100172 would be undefined. In the `POSIX` locale each byte is a valid single-byte character, and therefore  
 100173 this problem is avoided.

## 100174 EXAMPLES

100175 1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line  
 100176 numbers:

100177 `grep -i -n posix text.mm`

100178 2. To find all empty lines in the standard input:

100179 `grep ^$`

100180 or:

100181 `grep -v .`

100182 3. Both of the following commands print all lines containing strings "abc" or "def" or  
 100183 both:

100184 `grep -E 'abc|def'`

100185 `grep -F 'abc  
 100186 def'`

100187 4. Both of the following commands print all lines matching exactly "abc" or "def":

100188 `grep -E '^abc$|^def$'`

100189 `grep -F -x 'abc  
 100190 def'`

100191 **RATIONALE**

100192 This *grep* has been enhanced in an upwards-compatible way to provide the exact functionality of  
 100193 the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard  
 100194 developers to consolidate the three *greps* into a single command.

100195 The old *egrep* and *fgrep* commands are likely to be supported for many years to come as  
 100196 implementation extensions, allowing historical applications to operate unmodified.

100197 Historical implementations usually silently ignored all but one of multiply-specified *-e* and *-f*  
 100198 options, but were not consistent as to which specification was actually used.

100199 The *-b* option was omitted from the OPTIONS section because block numbers are  
 100200 implementation-defined.

100201 The System V restriction on using *-* to mean standard input was omitted.

100202 A definition of action taken when given a null BRE or ERE is specified. This is an error  
 100203 condition in some historical implementations.

100204 The *-I* option previously indicated that its use was undefined when no files were explicitly  
 100205 named. This behavior was historical and placed an unnecessary restriction on future  
 100206 implementations. It has been removed.

100207 The historical BSD *grep -s* option practice is easily duplicated by redirecting standard output to  
 100208 */dev/null*. The *-s* option required here is from System V.

100209 The *-x* option, historically available only with *fgrep*, is available here for all of the non-  
 100210 obsolescent versions.

100211 **FUTURE DIRECTIONS**

100212 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 100213 value of a <newline> character when <newline> is a terminator or separator in the output  
 100214 format being used, implementations are encouraged to treat this as an error. A future version of  
 100215 this standard may require implementations to treat this as an error.

100216 **SEE ALSO**

100217 *sed*

100218 XBD [Chapter 8](#) (on page 167), [Chapter 9](#) (on page 179), [Section 12.2](#) (on page 215)

100219 **CHANGE HISTORY**

100220 First released in Issue 2.

100221 **Issue 6**

100222 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.

100223 The normative text is reworded to avoid use of the term “must” for application requirements.

100224 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/28 is applied, correcting the examples  
 100225 using the *grep -F* option which did not match the normative description of the *-F* option.

100226 **Issue 7**

100227 Austin Group Interpretation 1003.1-2001 #092 is applied.

100228 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100229 SD5-XCU-ERN-98 is applied, updating the STDOUT section.

100230 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0105 [584] and XCU/TC2-2008/0106  
 100231 [663] are applied.

100232 **Issue 8**

- 100233 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used.
- 100234
- 100235
- 100236
- 100237 Austin Group Defect 1031 is applied, changing the description of the `-i` option.
- 100238 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.
- 100239 Austin Group Defect 1502 is applied, changing the EXIT STATUS section.

100240 **NAME**

100241 hash — remember or report utility locations

100242 **SYNOPSIS**100243 hash [*utility*...]

100244 hash -r

100245 **DESCRIPTION**

100246 The *hash* utility shall affect the way the current shell environment remembers the locations of  
 100247 utilities found as described in [Section 2.9.1.4](#) (on page 2502). Depending on the arguments  
 100248 specified, it shall add utility locations to its list of remembered locations or it shall purge the  
 100249 contents of the list. When no arguments are specified, it shall report on the contents of the list.

100250 Utilities provided as built-ins to the shell and functions shall not be reported by *hash*.100251 **OPTIONS**100252 The *hash* utility shall conform to XBD [Section 12.2](#) (on page 215).

100253 The following option shall be supported:

100254 -r Forget all previously remembered utility locations.

100255 **OPERANDS**

100256 The following operand shall be supported:

100257 *utility* The name of a utility to be searched for and added to the list of remembered  
 100258 locations. If the search does not find *utility*, it is unspecified whether or not this is  
 100259 treated as an error. If *utility* contains one or more <slash> characters, the results are  
 100260 unspecified.

100261 **STDIN**

100262 Not used.

100263 **INPUT FILES**

100264 None.

100265 **ENVIRONMENT VARIABLES**100266 The following environment variables shall affect the execution of *hash*:

100267 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 100268 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 100269 variables used to determine the values of locale categories.)

100270 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 100271 internationalization variables.

100272 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 100273 characters (for example, single-byte as opposed to multi-byte characters in  
 100274 arguments).

100275 *LC\_MESSAGES*

100276 Determine the locale that should be used to affect the format and contents of  
 100277 diagnostic messages written to standard error.

100278 XSI *NLSPATH* Determine the location of messages objects and message catalogs.100279 *PATH* Determine the location of *utility*, as described in XBD [Chapter 8](#) (on page 167).

100280 **ASYNCHRONOUS EVENTS**

100281 Default.

100282 **STDOUT**

100283 The standard output of *hash* shall be used when no arguments are specified. Its format is  
100284 unspecified, but includes the pathname of each utility in the list of remembered locations for the  
100285 current shell environment. This list shall consist of those utilities named in previous *hash*  
100286 invocations that have been invoked, and may contain those invoked and found through the  
100287 normal command search process. This list shall be cleared when the contents of the *PATH*  
100288 environment variable are changed.

100289 **STDERR**

100290 The standard error shall be used only for diagnostic messages.

100291 **OUTPUT FILES**

100292 None.

100293 **EXTENDED DESCRIPTION**

100294 None.

100295 **EXIT STATUS**

100296 The following exit values shall be returned:

100297 0 Successful completion.

100298 &gt;0 An error occurred.

100299 **CONSEQUENCES OF ERRORS**

100300 Default.

100301 **APPLICATION USAGE**100302 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

100303 Since *hash* affects the current shell execution environment, it is always provided as a shell  
100304 regular built-in. If it is called in a separate utility execution environment, such as one of the  
100305 following:

```
100306 nohup hash -r  
100307 find . -type f -exec hash {} +
```

100308 it does not affect the command search process of the caller's environment.

100309 The *hash* utility may be implemented as an alias—for example, *alias -t -*, in which case utilities  
100310 found through normal command search are not listed by the *hash* command.

100311 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the  
100312 simplest form, this can be:

100313 `PATH="$PATH"`

100314 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a  
100315 performance improvement on a few implementations; normally, the hashing process is included  
100316 by default.

100317 **EXAMPLES**

100318 None.

100319 **RATIONALE**

100320 None.

100321 **FUTURE DIRECTIONS**

100322 If this utility is directed to display a pathname that contains any bytes that have the encoded  
100323 value of a <newline> character when <newline> is a terminator or separator in the output  
100324 format being used, implementations are encouraged to treat this as an error. A future version of  
100325 this standard may require implementations to treat this as an error.

100326 **SEE ALSO**100327 [Section 2.9.1.4](#) (on page 2502)100328 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)100329 **CHANGE HISTORY**

100330 First released in Issue 2.

100331 **Issue 7**100332 The *hash* utility is moved from the XSI option to the Base.

100333 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0093 [241] is applied.

100334 **Issue 8**100335 Austin Group Defect 248 is applied, changing a command line in the APPLICATION USAGE  
100336 section.

100337 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
100338 directed to display a pathname that contains any bytes that have the encoded value of a  
100339 <newline> character when <newline> is a terminator or separator in the output format being  
100340 used.

100341 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
100342 this utility is required to be intrinsic.

100343 Austin Group Defect 1063 is applied, clarifying that functions are not reported by *hash*, and that  
100344 the list of remembered locations is cleared when the contents of *PATH* are changed.

100345 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

100346 Austin Group Defect 1460 is applied, making it explicitly unspecified whether or not *hash*  
100347 reports an error if it cannot find *utility*.

100348 **NAME**

100349 head — copy the first part of files

100350 **SYNOPSIS**100351 head [-c *number*|-n *number*] [*file*...]100352 **DESCRIPTION**100353 The *head* utility shall copy its input files to the standard output, ending the output for each file at  
100354 a designated point.100355 Copying shall end at the point in the file indicated by the *-c number* or *-n number* options. The  
100356 option-argument *number* shall be counted in units of lines or bytes, according to the options *-n*  
100357 and *-c*. Both line and byte counts start from 1.100358 **OPTIONS**100359 The *head* utility shall conform to XBD [Section 12.2](#) (on page 215).

100360 The following options shall be supported:

100361 *-c number* The first *number* bytes of each input file shall be copied to standard output. The  
100362 application shall ensure that the *number* option-argument is a positive decimal  
100363 integer.100364 *-n number* This option shall be equivalent to *-c number*, except that the ending location in the  
100365 file shall be measured in lines instead of bytes.100366 When a file contains less than *number* bytes or lines, it shall be copied to standard output in its  
100367 entirety. This shall not be an error.100368 If no options are specified, *head* shall act as if *-n 10* had been specified.100369 **OPERANDS**

100370 The following operand shall be supported:

100371 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
100372 shall be used.100373 **STDIN**100374 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
100375 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
100376 the standard input shall not be used. See the INPUT FILES section.100377 **INPUT FILES**100378 If the *-c* option is specified, the input files can contain arbitrary data; otherwise, the input files  
100379 shall be text files, but the line length shall not be restricted to {LINE\_MAX} bytes.100380 **ENVIRONMENT VARIABLES**100381 The following environment variables shall affect the execution of *head*:100382 *LANG* Provide a default value for the internationalization variables that are unset or null.  
100383 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
100384 variables used to determine the values of locale categories.)100385 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
100386 internationalization variables.100387 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
100388 characters (for example, single-byte as opposed to multi-byte characters in  
100389 arguments and input files).



- 100390 **LC\_MESSAGES**
- 100391 Determine the locale that should be used to affect the format and contents of
- 100392 diagnostic messages written to standard error.
- 100393 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 100394 **ASYNCHRONOUS EVENTS**
- 100395 Default.
- 100396 **STDOUT**
- 100397 The standard output shall contain designated portions of the input files.
- 100398 If multiple *file* operands are specified, *head* shall precede the output for each with the header:
- 100399 "\n==> %s <==\n", <pathname>
- 100400 except that the first header written shall not include the initial <newline>.
- 100401 **STDERR**
- 100402 The standard error shall be used only for diagnostic messages.
- 100403 **OUTPUT FILES**
- 100404 None.
- 100405 **EXTENDED DESCRIPTION**
- 100406 None.
- 100407 **EXIT STATUS**
- 100408 The following exit values shall be returned:
- 100409 0 Successful completion.
- 100410 >0 An error occurred.
- 100411 **CONSEQUENCES OF ERRORS**
- 100412 Default.
- 100413 **APPLICATION USAGE**
- 100414 When using *head* to process pathnames, it is recommended that *LC\_ALL*, or at least *LC\_CTYPE*
- 100415 and *LC\_COLLATE*, are set to *POSIX* or *C* in the environment, since pathnames can contain byte
- 100416 sequences that do not form valid characters in some locales, in which case the utility's behavior
- 100417 would be undefined. In the *POSIX* locale each byte is a valid single-byte character, and therefore
- 100418 this problem is avoided.
- 100419 **EXAMPLES**
- 100420 To write the first ten lines of all files (except those with a leading period) in the directory:
- 100421 `head -- *`
- 100422 **RATIONALE**
- 100423 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers
- 100424 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside
- 100425 *tail*.
- 100426 *POSIX.1-2024* version of *head* follows the Utility Syntax Guidelines. The `-n` option was added to
- 100427 this new interface so that *head* and *tail* would be more logically related. Earlier versions of this
- 100428 standard allowed a `-number` option. This form is no longer specified by *POSIX.1-2024* but may
- 100429 be present in some implementations.
- 100430 The *head* and *tail* utilities have not historically been symmetric. For example, this standard only
- 100431 requires *tail* to support at most one file operand, while *head* must operate on multiple files.

100432 Conversely, this standard requires *tail* to be able to start at a position relative to the start of a file,  
100433 but *head* need not support stopping at a position relative to the end of the file. Implementations  
100434 may choose to make *head* and *tail* symmetric as an extension, but applications should not rely on  
100435 this.

100436 Older implementations of *head* did not support `-c number`, but emulating this via `dd ibs=1`  
100437 `count=number` is much less efficient and emulating via `dd obs=pipe_buf | dd ibs=size`  
100438 `count=number_of_blocks` is cumbersome, somewhat less efficient, and can only be used if  
100439 the number of bytes to be copied is a multiple of a suitable block size less than or equal to  
100440 `{PIPE_BUF}`.

#### 100441 FUTURE DIRECTIONS

100442 If this utility is directed to display a pathname that contains any bytes that have the encoded  
100443 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
100444 format being used, implementations are encouraged to treat this as an error. A future version of  
100445 this standard may require implementations to treat this as an error.

#### 100446 SEE ALSO

100447 *dd*, *sed*, *tail*

100448 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

#### 100449 CHANGE HISTORY

100450 First released in Issue 4.

#### 100451 Issue 6

100452 The obsolescent `-number` form is removed.

100453 The normative text is reworded to avoid use of the term “must” for application requirements.

100454 The DESCRIPTION is updated to clarify that when a file contains less than the number of lines  
100455 requested, the entire file is copied to standard output.

#### 100456 Issue 7

100457 Austin Group Interpretations 1003.1-2001 #027 and #092 are applied.

100458 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100459 The APPLICATION USAGE section is removed and the EXAMPLES section is corrected.

100460 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0107 [663] is applied.

#### 100461 Issue 8

100462 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
100463 directed to display a pathname that contains any bytes that have the encoded value of a  
100464 `<newline>` character when `<newline>` is a terminator or separator in the output format being  
100465 used.

100466 Austin Group Defect 407 is applied, adding the `-c number` option.

100467 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

100468 **NAME**

100469 iconv — codeset conversion

100470 **SYNOPSIS**100471 iconv [-cs] -f *frommap* -t *tomap* [*file...*]100472 iconv -f *fromcode* [-cs] [-t *tocode*] [*file...*]100473 iconv -t *tocode* [-cs] [-f *fromcode*] [*file...*]

100474 iconv -l

100475 **DESCRIPTION**100476 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and  
100477 write the results to standard output.100478 When the options indicate that charmap files are used to specify the codesets (see **OPTIONS**),  
100479 the codeset conversion shall be accomplished by performing a logical join on the symbolic  
100480 character names in the two charmaps. The implementation need not support the use of charmap  
100481 files for codeset conversion unless the POSIX2\_LOCALEDEF symbol is defined on the system.100482 **OPTIONS**100483 The *iconv* utility shall conform to XBD [Section 12.2](#) (on page 215).

100484 The following options shall be supported:

100485 **-c** Omit any characters that are invalid in the codeset of the input file from the  
100486 output. When **-c** is not used, the results of encountering invalid characters in the  
100487 input stream (either those that are not characters in the codeset of the input file or  
100488 that have no corresponding character in the codeset of the output file) shall be  
100489 specified in the system documentation. The presence or absence of **-c** shall not  
100490 affect the exit status of *iconv*.100491 **-f** *fromcodeset*100492 Identify the codeset of the input file. The implementation shall recognize the  
100493 following two forms of the *fromcodeset* option-argument:100494 *fromcode* The *fromcode* option-argument can not contain a <slash> character. It  
100495 shall be interpreted as the name of one of the codeset descriptions  
100496 provided by the implementation in an unspecified format. Valid  
100497 values of *fromcode* are implementation-defined.100498 *frommap* The *frommap* option-argument needs to contain a <slash> character. It  
100499 shall be interpreted as the pathname of a charmap file as defined in  
100500 XBD [Section 6.4](#) (on page 121). If the pathname does not represent a  
100501 valid, readable charmap file, the results are undefined.

100502 If this option is omitted, the codeset of the current locale shall be used.

100503 **-l** Write all supported *fromcode* and *tocode* values to standard output in an unspecified  
100504 format.100505 **-s** Suppress any messages written to standard error concerning invalid characters.  
100506 When **-s** is not used, the results of encountering invalid characters in the input  
100507 stream (either those that are not valid characters in the codeset of the input file or  
100508 that have no corresponding character in the codeset of the output file) shall be  
100509 specified in the system documentation. The presence or absence of **-s** shall not  
100510 affect the exit status of *iconv*.

- 100511        **-t tocodeset** Identify the codeset to be used for the output file. The implementation shall  
100512                                   recognize the following two forms of the *tocodeset* option-argument:
- 100513                    *tocode*            The semantics shall be equivalent to the **-f fromcode** option.
- 100514                    *tomap*             The semantics shall be equivalent to the **-f frommap** option.
- 100515                    If this option is omitted, the codeset of the current locale shall be used.
- 100516            If either **-f** or **-t** represents a charmap file, but the other does not (or is omitted), or both **-f** and  
100517            **-t** are omitted, the results are undefined.
- 100518 **OPERANDS**
- 100519        The following operand shall be supported:
- 100520        *file*            A pathname of an input file. If no *file* operands are specified, or if a *file* operand is  
100521                                   '-' , the standard input shall be used.
- 100522 **STDIN**
- 100523        The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-' .
- 100524 **INPUT FILES**
- 100525        The input file shall be a text file.
- 100526 **ENVIRONMENT VARIABLES**
- 100527        The following environment variables shall affect the execution of *iconv*:
- 100528        **LANG**            Provide a default value for the internationalization variables that are unset or null.  
100529                                   (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
100530                                   variables used to determine the values of locale categories.)
- 100531        **LC\_ALL**           If set to a non-empty string value, override the values of all the other  
100532                                   internationalization variables.
- 100533        **LC\_CTYPE**        Determine the locale for the interpretation of sequences of bytes of text data as  
100534                                   characters (for example, single-byte as opposed to multi-byte characters in  
100535                                   arguments). During translation of the file, this variable is superseded by the use of  
100536                                   the *fromcode* option-argument.
- 100537        **LC\_MESSAGES**
- 100538                                   Determine the locale that should be used to affect the format and contents of  
100539                                   diagnostic messages written to standard error.
- 100540 XSI        **NLSPATH**        Determine the location of messages objects and message catalogs.
- 100541 **ASYNCHRONOUS EVENTS**
- 100542        Default.
- 100543 **STDOUT**
- 100544        When the **-I** option is used, the standard output shall contain all supported *fromcode* and *tocode*  
100545                                   values, written in an unspecified format.
- 100546        When the **-I** option is not used, the standard output shall contain the sequence of characters  
100547                                   read from the input files, translated to the specified codeset. Nothing else shall be written to the  
100548                                   standard output.
- 100549 **STDERR**
- 100550        The standard error shall be used only for diagnostic messages.

100551 **OUTPUT FILES**

100552 None.

100553 **EXTENDED DESCRIPTION**

100554 None.

100555 **EXIT STATUS**

100556 The following exit values shall be returned:

100557 0 Successful completion.

100558 &gt;0 An error occurred.

100559 **CONSEQUENCES OF ERRORS**

100560 Default.

100561 **APPLICATION USAGE**100562 The user must ensure that both charmap files use the same symbolic names for characters the  
100563 two codesets have in common.100564 **EXAMPLES**100565 The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:2001  
100566 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file  
100567 **mail.local**:100568 `iconv -f IS6937 -t IS8859 mail.x400 > mail.local`100569 **RATIONALE**100570 The *iconv* utility can be used portably only when the user provides two charmap files as option-  
100571 arguments. This is because a single charmap provided by the user cannot reliably be joined with  
100572 the names in a system-provided character set description. The valid values for *fromcode* and  
100573 *tocode* are implementation-defined and do not have to have any relation to the charmap  
100574 mechanisms. As an aid to interactive users, the `-I` option was adopted from the Plan 9 operating  
100575 system. It writes information concerning these implementation-defined values. The format is  
100576 unspecified because there are many possible useful formats that could be chosen, such as a  
100577 matrix of valid combinations of *fromcode* and *tocode*. The `-I` option is not intended for shell script  
100578 usage; conforming applications will have to use charmaps.100579 The *iconv* utility may support the conversion between ASCII and EBCDIC-based encodings, but  
100580 is not required to do so. In an XSI-compliant implementation, the *dd* utility is the only method  
100581 guaranteed to support conversion between these two character sets.100582 **FUTURE DIRECTIONS**

100583 None.

100584 **SEE ALSO**100585 *dd*, *gencat*100586 XBD [Section 6.4](#) (on page 121), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)100587 **CHANGE HISTORY**

100588 First released in Issue 3.

100589 **Issue 6**100590 This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the  
100591 ability to use charmap files for conversion has been added.100592 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/29 is applied, making changes to address  
100593 inconsistencies with the *iconv()* function in the System Interfaces volume of POSIX.1-2024.

100594 **Issue 7**

100595 Austin Group Interpretation 1003.1-2001 #206 is applied, correcting the *tomap* option.

100596 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100597 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0094 [291] and XCU/TC1-2008/0095  
100598 [291] are applied.

100599 **Issue 8**

100600 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

100601 **NAME**100602 `id` — return user identity100603 **SYNOPSIS**100604 `id` [*user*]100605 `id -G [-n] [user]`100606 `id -g [-nr] [user]`100607 `id -u [-nr] [user]`100608 **DESCRIPTION**

100609 If no *user* operand is provided, the *id* utility shall write the user and group IDs and the  
 100610 corresponding user and group names of the invoking process to standard output. If the effective  
 100611 and real IDs do not match, both shall be written. If multiple groups are supported by the  
 100612 underlying system (see the description of {NGROUPS\_MAX} in the System Interfaces volume of  
 100613 POSIX.1-2024), the supplementary group affiliations of the invoking process shall also be  
 100614 written.

100615 If a *user* operand is provided and the process has appropriate privileges, the user and group IDs  
 100616 of the selected user shall be written. In this case, effective IDs shall be assumed to be identical to  
 100617 real IDs. If the selected user has more than one allowable group membership listed in the group  
 100618 database, these shall be written in the same manner as the supplementary groups described in  
 100619 the preceding paragraph.

100620 **OPTIONS**100621 The *id* utility shall conform to XBD [Section 12.2](#) (on page 215).

100622 The following options shall be supported:

100623 **-G** Output all different group IDs (effective, real, and supplementary) only, using the  
 100624 format "`%u\n`". If there is more than one distinct group affiliation, output each  
 100625 such affiliation, using the format "`%u`", before the <newline> is output.

100626 **-g** Output only the effective group ID, using the format "`%u\n`".

100627 **-n** Output the name in the format "`%s`" instead of the numeric ID using the format  
 100628 "`%u`".

100629 **-r** Output the real ID instead of the effective ID.

100630 **-u** Output only the effective user ID, using the format "`%u\n`".

100631 **OPERANDS**

100632 The following operand shall be supported:

100633 *user* The login name for which information is to be written.

100634 **STDIN**

100635 Not used.

100636 **INPUT FILES**

100637 None.

100638 **ENVIRONMENT VARIABLES**100639 The following environment variables shall affect the execution of *id*:

100640 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 100641 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 100642 variables used to determine the values of locale categories.)

100643 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
100644 internationalization variables.

100645 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
100646 characters (for example, single-byte as opposed to multi-byte characters in  
100647 arguments).

100648 *LC\_MESSAGES*  
100649 Determine the locale that should be used to affect the format and contents of  
100650 diagnostic messages written to standard error and informative messages written to  
100651 standard output.

100652 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

100653 **ASYNCHRONOUS EVENTS**  
100654 Default.

100655 **STDOUT**  
100656 The following formats shall be used when the *LC\_MESSAGES* locale category specifies the  
100657 POSIX locale. In other locales, the strings *uid*, *gid*, *euid*, *egid*, and *groups* may be replaced with  
100658 more appropriate strings corresponding to the locale.

100659 "uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,  
100660 <real group ID>, <group-name>

100661 If the effective and real user IDs do not match, the following shall be inserted immediately  
100662 before the '\n' character in the previous format:

100663 " euid=%u(%s) "

100664 with the following arguments added at the end of the argument list:

100665 <effective user ID>, <effective user-name>

100666 If the effective and real group IDs do not match, the following shall be inserted directly before  
100667 the '\n' character in the format string (and after any addition resulting from the effective and  
100668 real user IDs not matching):

100669 " egid=%u(%s) "

100670 with the following arguments added at the end of the argument list:

100671 <effective group-ID>, <effective group name>

100672 If the process has supplementary group affiliations or the selected user is allowed to belong to  
100673 multiple groups, the first shall be added directly before the <newline> in the format string:

100674 " groups=%u(%s) "

100675 with the following arguments added at the end of the argument list:

100676 <supplementary group ID>, <supplementary group name>

100677 and the necessary number of the following added after that for any remaining supplementary  
100678 group IDs:

100679 ", %u(%s) "

100680 and the necessary number of the following arguments added at the end of the argument list:

100681 <supplementary group ID>, <supplementary group name>

100682 If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple



100683 group IDs cannot be mapped by the system into printable user or group names, the  
100684 corresponding "(%s)" and *name* argument shall be omitted from the corresponding format  
100685 string.

100686 When any of the options are specified, the output format shall be as described in the OPTIONS  
100687 section.

#### 100688 **STDERR**

100689 The standard error shall be used only for diagnostic messages.

#### 100690 **OUTPUT FILES**

100691 None.

#### 100692 **EXTENDED DESCRIPTION**

100693 None.

#### 100694 **EXIT STATUS**

100695 The following exit values shall be returned:

100696 0 Successful completion.

100697 >0 An error occurred.

#### 100698 **CONSEQUENCES OF ERRORS**

100699 Default.

#### 100700 **APPLICATION USAGE**

100701 Output produced by the **-G** option and by the default case could potentially produce very long  
100702 lines on systems that support large numbers of supplementary groups. (On systems with user  
100703 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per  
100704 name, 93 supplementary groups plus distinct effective and real group and user IDs could  
100705 theoretically overflow the 2048-byte {LINE\_MAX} text file line limit on the default output case.  
100706 It would take about 186 supplementary groups to overflow the 2048-byte barrier using *id -G*.)  
100707 This is not expected to be a problem in practice, but in cases where it is a concern, applications  
100708 should consider using *fold -s* before post-processing the output of *id*.

#### 100709 **EXAMPLES**

100710 None.

#### 100711 **RATIONALE**

100712 The functionality provided by the 4 BSD *groups* utility can be simulated using:

```
100713 id -Gn [ user ]
```

100714 The 4 BSD command *groups* was considered, but it was not included because it did not provide  
100715 the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to  
100716 modify *id* to provide the additional functionality necessary to systems with multiple groups than  
100717 to invent another command.

100718 The options **-u**, **-g**, **-n**, and **-r** were added to ease the use of *id* with shell commands  
100719 substitution. Without these options it is necessary to use some preprocessor such as *sed* to select  
100720 the desired piece of information. Since output such as that produced by:

```
100721 id -u -n
```

100722 is frequently wanted, it seemed desirable to add the options.

100723 **FUTURE DIRECTIONS**

100724 None.

100725 **SEE ALSO**100726 *fold, logname, who*

100727 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

100728 XSH *getgid()*, *getgroups()*, *getuid()*100729 **CHANGE HISTORY**

100730 First released in Issue 2.

100731 **Issue 7**

100732 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100733 **Issue 8**100734 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

100735 **NAME**

100736 ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

100737 **SYNOPSIS**

```
100738 XSI ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey]...
```

100739 **DESCRIPTION**

100740 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory  
100741 segments. The interprocess communication facilities to be removed are specified by the options.

100742 Only a user with appropriate privileges shall be allowed to remove an interprocess  
100743 communication facility that was not created by or owned by the user invoking *ipcrm*.

100744 **OPTIONS**

100745 The *ipcrm* utility shall conform to XBD [Section 12.2](#) (on page 215).

100746 The following options shall be supported:

100747 **-q** *msgid* Remove the message queue identifier *msgid* from the system and destroy the  
100748 message queue and data structure associated with it.

100749 **-m** *shmid* Remove the shared memory identifier *shmid* from the system. The shared memory  
100750 segment and data structure associated with it shall be destroyed when all  
100751 processes with the segment attached have either detached the segment or  
100752 terminated. If the segment is not attached to any process, it shall be destroyed  
100753 immediately.

100754 **-s** *semid* Remove the semaphore identifier *semid* from the system and destroy the set of  
100755 semaphores and data structure associated with it.

100756 **-Q** *msgkey* Remove the message queue identifier, created with key *msgkey*, from the system  
100757 and destroy the message queue and data structure associated with it.

100758 **-M** *shmkey* Remove the shared memory identifier, created with key *shmkey*, from the system.  
100759 The shared memory segment and data structure associated with it shall be  
100760 destroyed after the last detach.

100761 **-S** *semkey* Remove the semaphore identifier, created with key *semkey*, from the system and  
100762 destroy the set of semaphores and data structure associated with it.

100763 **OPERANDS**

100764 None.

100765 **STDIN**

100766 Not used.

100767 **INPUT FILES**

100768 None.

100769 **ENVIRONMENT VARIABLES**

100770 The following environment variables shall affect the execution of *ipcrm*:

100771 **LANG** Provide a default value for the internationalization variables that are unset or null.  
100772 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
100773 variables used to determine the values of locale categories.)

100774 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
100775 internationalization variables.

- 100776 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 100777
- 100778
- 100779 **LC\_MESSAGES**
- 100780 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 100781
- 100782 **NLSPATH** Determine the location of messages objects and message catalogs.
- 100783 **ASYNCHRONOUS EVENTS**
- 100784 Default.
- 100785 **STDOUT**
- 100786 Not used.
- 100787 **STDERR**
- 100788 The standard error shall be used only for diagnostic messages.
- 100789 **OUTPUT FILES**
- 100790 None.
- 100791 **EXTENDED DESCRIPTION**
- 100792 None.
- 100793 **EXIT STATUS**
- 100794 The following exit values shall be returned:
- 100795 0 Successful completion.
- 100796 >0 An error occurred.
- 100797 **CONSEQUENCES OF ERRORS**
- 100798 Default.
- 100799 **APPLICATION USAGE**
- 100800 None.
- 100801 **EXAMPLES**
- 100802 None.
- 100803 **RATIONALE**
- 100804 None.
- 100805 **FUTURE DIRECTIONS**
- 100806 None.
- 100807 **SEE ALSO**
- 100808 [\*ipcs\*](#)
- 100809 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)
- 100810 XSH [\*msgctl\(\)\*](#), [\*semctl\(\)\*](#), [\*shmctl\(\)\*](#)
- 100811 **CHANGE HISTORY**
- 100812 First released in Issue 5.
- 100813 **Issue 7**
- 100814 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100815 **Issue 8**

100816 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

100817 Austin Group Defect 1240 is applied, clarifying the description of the **-m** option.

100818 **NAME**

100819 ipcs — report XSI interprocess communication facilities status

100820 **SYNOPSIS**100821 XSI `ipcs [-qms] [-a|-bcopt]`100822 **DESCRIPTION**100823 The *ipcs* utility shall write information about active interprocess communication facilities.

100824 Without options, information shall be written in short format for message queues, shared  
 100825 memory segments, and semaphore sets that are currently active in the system. Otherwise, the  
 100826 information that is displayed is controlled by the options specified.

100827 **OPTIONS**100828 The *ipcs* utility shall conform to XBD [Section 12.2](#) (on page 215).100829 The *ipcs* utility accepts the following options:100830 **-q** Write information about active message queues.100831 **-m** Write information about active shared memory segments.100832 **-s** Write information about active semaphore sets.

100833 If **-q**, **-m**, or **-s** are specified, only information about those facilities shall be written. If none of  
 100834 these three are specified, information about all three shall be written subject to the following  
 100835 options:

100836 **-a** Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

100837 **-b** Write information on maximum allowable size. (Maximum number of bytes in  
 100838 messages on queue for message queues, size of segments for shared memory, and  
 100839 number of semaphores in each set for semaphores.)

100840 **-c** Write creator's user name and group name; see below.

100841 **-o** Write information on outstanding usage. (Number of messages on queue and total  
 100842 number of bytes in messages on queue for message queues, and number of  
 100843 processes attached to shared memory segments.)

100844 **-p** Write process number information. (Process ID of the last process to send a  
 100845 message and process ID of the last process to receive a message on message  
 100846 queues, process ID of the creating process, and process ID of the last process to  
 100847 attach or detach on shared memory segments.)

100848 **-t** Write time information. (Time of the last control operation that changed the access  
 100849 permissions for all facilities, time of the last *msgsnd()* and *msgrcv()* operations on  
 100850 message queues, time of the last *shmat()* and *shmdt()* operations on shared  
 100851 memory, and time of the last *semop()* operation on semaphores.)

100852 **OPERANDS**

100853 None.

100854 **STDIN**

100855 Not used.

100856 **INPUT FILES**

100857 • The group database

- The user database

## 100859 ENVIRONMENT VARIABLES

100860 The following environment variables shall affect the execution of *ipcs*:

100861 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 100862 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 100863 variables used to determine the values of locale categories.)

100864 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 100865 internationalization variables.

100866 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 100867 characters (for example, single-byte as opposed to multi-byte characters in  
 100868 arguments).

100869 *LC\_MESSAGES*  
 100870 Determine the locale that should be used to affect the format and contents of  
 100871 diagnostic messages written to standard error.

100872 *NLSPATH* Determine the location of messages objects and message catalogs.

100873 *TZ* Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset  
 100874 or null, an unspecified default timezone shall be used.

## 100875 ASYNCHRONOUS EVENTS

100876 Default.

## 100877 STDOUT

100878 An introductory line shall be written with the format:

100879 `"IPC status from %s as of %s\n", <source>, <date>`

100880 where *<source>* indicates the source used to gather the statistics and *<date>* is the information  
 100881 that would be produced by the *date* command when invoked in the POSIX locale.

100882 The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options.  
 100883 The first report shall indicate the status of message queues, the second report shall indicate the  
 100884 status of shared memory segments, and the third report shall indicate the status of semaphore  
 100885 sets.

100886 If the corresponding facility is not installed or has not been used since the last reboot, then the  
 100887 report shall be written out in the format:

100888 `"%s facility not in system.\n", <facility>`

100889 where *<facility>* is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has  
 100890 been installed and has been used since the last reboot, column headings separated by one or  
 100891 more *<space>* characters and followed by a *<newline>* shall be written as indicated below  
 100892 followed by the facility name written out using the format:

100893 `"%s:\n", <facility>`

100894 where *<facility>* is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second  
 100895 and third reports the column headings need not be written if the last column headings written  
 100896 already provide column headings for all information in that report.

100897 The column headings provided in the first column below and the meaning of the information in  
 100898 those columns shall be given in order below; the letters in parentheses indicate the options that  
 100899 shall cause the corresponding column to appear; "all" means that the column shall always  
 100900 appear. Each column is separated by one or more *<space>* characters. Note that these options

|        |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 100901 |            | only determine what information is provided for each report; they do not determine which reports are written.                                                                                                                                                                                                                                                                                                                               |
| 100902 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100903 | T (all)    | Type of facility:                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 100904 |            | q Message queue.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 100905 |            | m Shared memory segment.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 100906 |            | s Semaphore.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 100907 |            | This field is a single character written using the format %c.                                                                                                                                                                                                                                                                                                                                                                               |
| 100908 | ID (all)   | The identifier for the facility entry. This field shall be written using the format %d.                                                                                                                                                                                                                                                                                                                                                     |
| 100909 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100910 | KEY (all)  | The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the facility entry.                                                                                                                                                                                                                                                                                                                         |
| 100911 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100912 |            | <b>Note:</b> The key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.                                                                                                                                                                                                                                                                          |
| 100913 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100914 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100915 |            | This field shall be written using the format 0x%x.                                                                                                                                                                                                                                                                                                                                                                                          |
| 100916 | MODE (all) | The facility access modes and flags. The mode shall consist of 11 characters that are interpreted as follows.                                                                                                                                                                                                                                                                                                                               |
| 100917 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100918 |            | The first character shall be:                                                                                                                                                                                                                                                                                                                                                                                                               |
| 100919 |            | S If a process is waiting on a <i>msgsnd()</i> operation.                                                                                                                                                                                                                                                                                                                                                                                   |
| 100920 |            | – If the above is not true.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 100921 |            | The second character shall be:                                                                                                                                                                                                                                                                                                                                                                                                              |
| 100922 |            | R If a process is waiting on a <i>msgrcv()</i> operation.                                                                                                                                                                                                                                                                                                                                                                                   |
| 100923 |            | C or – If the associated shared memory segment is to be cleared when the first attach operation is executed.                                                                                                                                                                                                                                                                                                                                |
| 100924 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100925 |            | – If none of the above is true.                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100926 |            | The next nine characters shall be interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the usergroup of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is a <hyphen-minus> ('-'). |
| 100927 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100928 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100929 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100930 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100931 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100932 |            | The permissions shall be indicated as follows:                                                                                                                                                                                                                                                                                                                                                                                              |
| 100933 |            | r If read permission is granted.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 100934 |            | w If write permission is granted.                                                                                                                                                                                                                                                                                                                                                                                                           |
| 100935 |            | a If alter permission is granted.                                                                                                                                                                                                                                                                                                                                                                                                           |
| 100936 |            | – If the indicated permission is not granted.                                                                                                                                                                                                                                                                                                                                                                                               |
| 100937 |            | The first character following the permissions specifies if there is an alternate or additional access control method associated with the facility. If there is no alternate or additional access control method associated with the facility, a single <space> shall be written; otherwise, another printable character is                                                                                                                  |
| 100938 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100939 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 100940 |            |                                                                                                                                                                                                                                                                                                                                                                                                                                             |



|        |               |                                                                                                                                                                                                                                                                                                               |
|--------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 100941 |               | written.                                                                                                                                                                                                                                                                                                      |
| 100942 | OWNER (all)   | The user name of the owner of the facility entry. If the user name of the owner is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the owner shall be written using the format %d.                          |
| 100943 |               |                                                                                                                                                                                                                                                                                                               |
| 100944 |               |                                                                                                                                                                                                                                                                                                               |
| 100945 |               |                                                                                                                                                                                                                                                                                                               |
| 100946 | GROUP (all)   | The group name of the owner of the facility entry. If the group name of the owner is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the owner shall be written using the format %d.                      |
| 100947 |               |                                                                                                                                                                                                                                                                                                               |
| 100948 |               |                                                                                                                                                                                                                                                                                                               |
| 100949 |               |                                                                                                                                                                                                                                                                                                               |
| 100950 |               | The following nine columns shall be only written out for message queues:                                                                                                                                                                                                                                      |
| 100951 | CREATOR (a,c) | The user name of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.                    |
| 100952 |               |                                                                                                                                                                                                                                                                                                               |
| 100953 |               |                                                                                                                                                                                                                                                                                                               |
| 100954 |               |                                                                                                                                                                                                                                                                                                               |
| 100955 | CGROUP (a,c)  | The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.                |
| 100956 |               |                                                                                                                                                                                                                                                                                                               |
| 100957 |               |                                                                                                                                                                                                                                                                                                               |
| 100958 |               |                                                                                                                                                                                                                                                                                                               |
| 100959 | CBYTES (a,o)  | The number of bytes in messages currently outstanding on the associated message queue. This field shall be written using the format %d.                                                                                                                                                                       |
| 100960 |               |                                                                                                                                                                                                                                                                                                               |
| 100961 | QNUM (a,o)    | The number of messages currently outstanding on the associated message queue. This field shall be written using the format %d.                                                                                                                                                                                |
| 100962 |               |                                                                                                                                                                                                                                                                                                               |
| 100963 | QBYTES (a,b)  | The maximum number of bytes allowed in messages outstanding on the associated message queue. This field shall be written using the format %d.                                                                                                                                                                 |
| 100964 |               |                                                                                                                                                                                                                                                                                                               |
| 100965 | LSPID (a,p)   | The process ID of the last process to send a message to the associated queue. This field shall be written using the format:                                                                                                                                                                                   |
| 100966 |               |                                                                                                                                                                                                                                                                                                               |
| 100967 |               | "%d", <pid>                                                                                                                                                                                                                                                                                                   |
| 100968 |               | where <pid> is 0 if no message has been sent to the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to send a message to the queue.                                                                                                                                 |
| 100969 |               |                                                                                                                                                                                                                                                                                                               |
| 100970 |               |                                                                                                                                                                                                                                                                                                               |
| 100971 | LRPID (a,p)   | The process ID of the last process to receive a message from the associated queue. This field shall be written using the format:                                                                                                                                                                              |
| 100972 |               |                                                                                                                                                                                                                                                                                                               |
| 100973 |               | "%d", <pid>                                                                                                                                                                                                                                                                                                   |
| 100974 |               | where <pid> is 0 if no message has been received from the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to receive a message from the queue.                                                                                                                      |
| 100975 |               |                                                                                                                                                                                                                                                                                                               |
| 100976 |               |                                                                                                                                                                                                                                                                                                               |
| 100977 | STIME (a,t)   | The time the last message was sent to the associated queue. If a message has been sent to the corresponding message queue, the hour, minute, and second of the last time a message was sent to the queue shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format "no-entry" shall be written. |
| 100978 |               |                                                                                                                                                                                                                                                                                                               |
| 100979 |               |                                                                                                                                                                                                                                                                                                               |
| 100980 |               |                                                                                                                                                                                                                                                                                                               |
| 100981 |               |                                                                                                                                                                                                                                                                                                               |
| 100982 | RTIME (a,t)   | The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue                                                                                    |
| 100983 |               |                                                                                                                                                                                                                                                                                                               |
| 100984 |               |                                                                                                                                                                                                                                                                                                               |

100985 shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format  
100986 " no-entry" shall be written.

100987 The following eight columns shall be only written out for shared memory segments.

100988 CREATOR (a,c) The user of the creator of the facility entry. If the user name of the creator is  
100989 found in the user database, at least the first eight column positions of the  
100990 name shall be written using the format %s. Otherwise, the user ID of the  
100991 creator shall be written using the format %d.

100992 CGROUP (a,c) The group name of the creator of the facility entry. If the group name of the  
100993 creator is found in the group database, at least the first eight column positions  
100994 of the name shall be written using the format %s. Otherwise, the group ID of  
100995 the creator shall be written using the format %d.

100996 NATTCH (a,o) The number of processes attached to the associated shared memory segment.  
100997 This field shall be written using the format %d.

100998 SEGSZ (a,b) The size of the associated shared memory segment. This field shall be written  
100999 using the format %d.

101000 CPID (a,p) The process ID of the creator of the shared memory entry. This field shall be  
101001 written using the format %d.

101002 LPID (a,p) The process ID of the last process to attach or detach the shared memory  
101003 segment. This field shall be written using the format:

101004 "%d", <pid>

101005 where <pid> is 0 if no process has attached the corresponding shared memory  
101006 segment; otherwise, <pid> shall be the process ID of the last process to attach  
101007 or detach the segment.

101008 ATIME (a,t) The time the last attach on the associated shared memory segment was  
101009 completed. If the corresponding shared memory segment has ever been  
101010 attached, the hour, minute, and second of the last time the segment was  
101011 attached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the  
101012 format " no-entry" shall be written.

101013 DTIME (a,t) The time the last detach on the associated shared memory segment was  
101014 completed. If the corresponding shared memory segment has ever been  
101015 detached, the hour, minute, and second of the last time the segment was  
101016 detached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the  
101017 format " no-entry" shall be written.

101018 The following four columns shall be only written out for semaphore sets:

101019 CREATOR (a,c) The user of the creator of the facility entry. If the user name of the creator is  
101020 found in the user database, at least the first eight column positions of the  
101021 name shall be written using the format %s. Otherwise, the user ID of the  
101022 creator shall be written using the format %d.

101023 CGROUP (a,c) The group name of the creator of the facility entry. If the group name of the  
101024 creator is found in the group database, at least the first eight column positions  
101025 of the name shall be written using the format %s. Otherwise, the group ID of  
101026 the creator shall be written using the format %d.

|        |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
|--------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 101027 | <b>NSEMS (a,b)</b>            | The number of semaphores in the set associated with the semaphore entry.                                                                                                                                                                                                                                                                                                         |
| 101028 |                               | This field shall be written using the format %d.                                                                                                                                                                                                                                                                                                                                 |
| 101029 | <b>OTIME (a,t)</b>            | The time the last semaphore operation on the set associated with the semaphore entry was completed. If a semaphore operation has ever been performed on the corresponding semaphore set, the hour, minute, and second of the last semaphore operation on the semaphore set shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written. |
| 101030 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101031 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101032 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101033 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101034 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101035 |                               | The following column shall be written for all three reports when it is requested:                                                                                                                                                                                                                                                                                                |
| 101036 | <b>CTIME (a,t)</b>            | The time the associated entry was created or changed. The hour, minute, and second of the time when the associated entry was created shall be written using the format %d:%2.2d:%2.2d.                                                                                                                                                                                           |
| 101037 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101038 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101039 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101040 |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                   |
| 101041 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101042 |                               | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 101043 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101044 |                               | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 101045 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101046 |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                     |
| 101047 |                               | 0 Successful completion.                                                                                                                                                                                                                                                                                                                                                         |
| 101048 |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                                            |
| 101049 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101050 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                         |
| 101051 | <b>APPLICATION USAGE</b>      |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101052 |                               | Things can change while <i>ipcs</i> is running; the information it gives is guaranteed to be accurate only when it was retrieved.                                                                                                                                                                                                                                                |
| 101053 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101054 | <b>EXAMPLES</b>               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101055 |                               | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 101056 | <b>RATIONALE</b>              |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101057 |                               | None.                                                                                                                                                                                                                                                                                                                                                                            |
| 101058 | <b>FUTURE DIRECTIONS</b>      |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101059 |                               | If this utility is directed to display a pathname that contains any bytes that have the encoded value of a <newline> character when <newline> is a terminator or separator in the output format being used, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.                   |
| 101060 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101061 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101062 |                               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101063 | <b>SEE ALSO</b>               |                                                                                                                                                                                                                                                                                                                                                                                  |
| 101064 |                               | <i>ipcrm</i>                                                                                                                                                                                                                                                                                                                                                                     |
| 101065 |                               | XBD Chapter 8 (on page 167), Section 12.2 (on page 215)                                                                                                                                                                                                                                                                                                                          |
| 101066 |                               | XSH <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmdt()</i> , <i>shmget()</i>                                                                                                                                                                                                                                                     |

101067 **CHANGE HISTORY**

101068 First released in Issue 5.

101069 **Issue 6**

101070 The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

101071 The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.

101072 The Open Group Base Resolution bwg98-004 is applied.

101073 **Issue 7**

101074 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

101075 SD5-XCU-ERN-139 is applied, adding the *ipcrm* utility to the SEE ALSO section.

101076 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0108 [584] is applied.

101077 **Issue 8**101078 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
101079 directed to display a pathname that contains any bytes that have the encoded value of a  
101080 <newline> character when <newline> is a terminator or separator in the output format being  
101081 used.101082 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

101083 **NAME**

101084 jobs — display status of jobs in the current shell execution environment

101085 **SYNOPSIS**101086 UP `jobs [-l | -p] [job_id...]`101087 **DESCRIPTION**

101088 If the current shell execution environment (see [Section 2.13](#), on page 2522) is not a subshell  
 101089 environment, the *jobs* utility shall display the status of background jobs that were created in the  
 101090 current shell execution environment; it may also do so if the current shell execution environment  
 101091 is a subshell environment.

101092 When *jobs* reports the termination status of a job, the shell shall remove the job from the  
 101093 background jobs list and the associated process ID from the list of those “known in the current  
 101094 shell execution environment”; see [Section 2.9.3.1](#) (on page 2506). If a write error occurs when  
 101095 *jobs* writes to standard output, some process IDs might have been removed from the list but not  
 101096 successfully reported.

101097 **OPTIONS**101098 The *jobs* utility shall conform to XBD [Section 12.2](#) (on page 215).

101099 The following options shall be supported:

101100 **-l** (The letter ell.) Provide more information about each job listed. See STDOUT for  
 101101 details.

101102 **-p** Display only the process IDs for the process group leaders of job-control  
 101103 background jobs and the process IDs associated with non-job-control background  
 101104 jobs (if supported).

101105 By default, the *jobs* utility shall display the status of all background jobs, both running and  
 101106 suspended, and all jobs whose status has changed and have not been reported by the shell.

101107 **OPERANDS**

101108 The following operand shall be supported:

101109 *job\_id* Specifies the jobs for which the status is to be displayed. If no *job\_id* is given, the  
 101110 status information for all jobs shall be displayed. The format of *job\_id* is described  
 101111 in XBD [Section 3.182](#) (on page 57).

101112 **STDIN**

101113 Not used.

101114 **INPUT FILES**

101115 None.

101116 **ENVIRONMENT VARIABLES**101117 The following environment variables shall affect the execution of *jobs*:

101118 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 101119 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 101120 variables used to determine the values of locale categories.)

101121 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 101122 internationalization variables.

101123 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 101124 characters (for example, single-byte as opposed to multi-byte characters in  
 101125 arguments).

- 101126 *LC\_MESSAGES*
- 101127 Determine the locale that should be used to affect the format and contents of
- 101128 diagnostic messages written to standard error and informative messages written to
- 101129 standard output.
- 101130 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 101131 **ASYNCHRONOUS EVENTS**
- 101132 Default.
- 101133 **STDOUT**
- 101134 If the **-p** option is specified, the output shall consist of one line for each process ID:
- 101135 "%d\n", <process ID>
- 101136 Otherwise, if the **-l** option is not specified, the output shall be a series of lines of the form:
- 101137 "[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>
- 101138 where the fields shall be as follows:
- 101139 <current> The character '+' identifies the job that would be used as a default for the *fg* or *bg*
- 101140 utilities; this job can also be specified using the *job\_id* %+ or "%%". The character
- 101141 '-' identifies the job that would become the default if the current default job were
- 101142 to exit; this job can also be specified using the *job\_id* %-. For other jobs, this field is
- 101143 a <space>. At most one job can be identified with '+' and at most one job can be
- 101144 identified with '-'. If there is any suspended job, then the current job shall be a
- 101145 suspended job. If there are at least two suspended jobs, then the previous job also
- 101146 shall be a suspended job.
- 101147 <job-number> A number that can be used to identify the job to the *wait*, *fg*, *bg*, and *kill* utilities.
- 101148 Using these utilities, the job can be identified by prefixing the job number with
- 101149 '% '.
- 101150 <state> One of the following strings (in the POSIX locale):
- 101151 **Running** Indicates that the job has not been suspended by a signal and has not
- 101152 exited.
- 101153 **Done** Indicates that the job completed and returned exit status zero.
- 101154 **Done(code)** Indicates that the job completed normally and that it exited with the
- 101155 specified non-zero exit status, *code*, expressed as a decimal number.
- 101156 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.
- 101157 **Stopped (SIGTSTP)**
- 101158 Indicates that the job was suspended by the SIGTSTP signal.
- 101159 **Stopped (SIGSTOP)**
- 101160 Indicates that the job was suspended by the SIGSTOP signal.
- 101161 **Stopped (SIGTTIN)**
- 101162 Indicates that the job was suspended by the SIGTTIN signal.
- 101163 **Stopped (SIGTTOU)**
- 101164 Indicates that the job was suspended by the SIGTTOU signal.
- 101165 The implementation may substitute the string **Suspended** in place of **Stopped**. If
- 101166 the job was terminated by a signal, the format of <state> is unspecified, but it shall
- 101167 be visibly distinct from all of the other <state> formats shown here and shall

- 101168 indicate the name or description of the signal causing the termination.
- 101169 `<command>` The associated command that was given to the shell.
- 101170 If the `-l` option is specified:
- 101171 • For job-control background jobs, a field containing the process group ID shall be inserted
  - 101172 before the `<state>` field. Also, more processes in a process group may be output on separate
  - 101173 lines, using only the process ID and `<command>` fields.
  - 101174 • For non-job-control background jobs (if supported), a field containing the process ID
  - 101175 associated with the job shall be inserted before the `<state>` field. Also, more processes
  - 101176 created to execute the job may be output on separate lines, using only the process ID and
  - 101177 `<command>` fields.
- 101178 **STDERR**
- 101179 The standard error shall be used only for diagnostic messages.
- 101180 **OUTPUT FILES**
- 101181 None.
- 101182 **EXTENDED DESCRIPTION**
- 101183 None.
- 101184 **EXIT STATUS**
- 101185 The following exit values shall be returned:
- 101186 0 The output specified in `STDOUT` was successfully written to standard output.
- 101187 >0 An error occurred.
- 101188 **CONSEQUENCES OF ERRORS**
- 101189 Default.
- 101190 **APPLICATION USAGE**
- 101191 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.
- 101192 The `-p` option is the only portable way to find out the process group of a job-control background
- 101193 job because different implementations have different strategies for defining the process group of
- 101194 the job. Usage such as `$(jobs -p)` provides a way of referring to the process group of the job in an
- 101195 implementation-independent way.
- 101196 The `jobs` utility does not work as expected when it is operating in its own utility execution
- 101197 environment because that environment has no applicable jobs to manipulate. See the
- 101198 **APPLICATION USAGE** section for `bg`. For this reason, `jobs` is generally implemented as a shell
- 101199 regular built-in.
- 101200 **EXAMPLES**
- 101201 None.
- 101202 **RATIONALE**
- 101203 Both "`%%`" and "`%+`" are used to refer to the current job. Both forms are of equal validity—the
- 101204 "`%%`" mirroring "`$$`" and "`%+`" mirroring the output of `jobs`. Both forms reflect historical
- 101205 practice of the KornShell and the C shell with job control.
- 101206 The job control features provided by `bg`, `fg`, and `jobs` are based on the KornShell. The standard
- 101207 developers examined the characteristics of the C shell versions of these utilities and found that
- 101208 differences exist. Despite widespread use of the C shell, the KornShell versions were selected for
- 101209 this volume of POSIX.1-2024 to maintain a degree of uniformity with the rest of the KornShell
- 101210 features selected (such as the very popular command line editing features).

101211 The *jobs* utility is not dependent on job control being enabled, as are the seemingly related *bg*  
 101212 and *fg* utilities because *jobs* is useful for examining background jobs, regardless of the current  
 101213 state of job control. When job control has been disabled using *set +m*, the *jobs* utility can still be  
 101214 used to examine the job-control background jobs and (if supported) non-job-control background  
 101215 jobs that were created in the current shell execution environment. See also the RATIONALE for  
 101216 *kill* and *wait*.

101217 The output for terminated jobs is left unspecified to accommodate various historical systems.  
 101218 The following formats have been witnessed:

- 101219 1. **Killed**(*signal name*)
- 101220 2. *signal name*
- 101221 3. *signal name*(**coredump**)
- 101222 4. *signal description*– **core dumped**

101223 Most users should be able to understand these formats, although it means that applications have  
 101224 trouble parsing them.

101225 The calculation of job IDs was not described since this would suggest an implementation, which  
 101226 may impose unnecessary restrictions.

101227 In an early proposal, a **-n** option was included to “Display the status of jobs that have changed,  
 101228 exited, or stopped since the last status report”. It was removed because the shell always writes  
 101229 any changed status of jobs before each prompt.

101230 If *jobs* uses buffered writes to standard output, a write error could be detected when attempting  
 101231 to flush a buffer containing multiple reports of terminated jobs, resulting in some unreported  
 101232 jobs having their process IDs removed from the list of those known in the current shell execution  
 101233 environment (because they were removed when the report was added to the buffer).

#### 101234 FUTURE DIRECTIONS

101235 None.

#### 101236 SEE ALSO

101237 [Section 2.13](#) (on page 2522), *bg*, *fg*, *kill*, *wait*

101238 [XBD Section 3.182](#) (on page 57), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 101239 CHANGE HISTORY

101240 First released in Issue 4.

#### 101241 Issue 6

101242 This utility is marked as part of the User Portability Utilities option.

101243 The JC shading is removed as job control is mandatory in this version.

#### 101244 Issue 7

101245 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

#### 101246 Issue 8

101247 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
 101248 this utility is required to be intrinsic.

101249 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

101250 Austin Group Defect 1254 is applied, updating various requirements for the *jobs* utility to  
 101251 account for the addition of [Section 2.11](#) (on page 2518).

101252 Austin Group Defect 1492 is applied, clarifying the requirements when a write error to standard



101253

output occurs.

101254 **NAME**

101255 join — relational database operator

101256 **SYNOPSIS**

```
101257 join [-a file_number|-v file_number] [-e string] [-o list] [-t char]
101258      [-1 field] [-2 field] file1 file2
```

101259 **DESCRIPTION**

101260 The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be  
 101261 written to the standard output.

101262 The join field is a field in each file on which the files are compared. The *join* utility shall write  
 101263 one line in the output for each pair of lines in *file1* and *file2* that have join fields that collate  
 101264 equally. The output line by default shall consist of the join field, then the remaining fields from  
 101265 *file1*, then the remaining fields from *file2*. This format can be changed by using the **-o** option  
 101266 (see below). The **-a** option can be used to add unmatched lines to the output. The **-v** option can  
 101267 be used to output only unmatched lines.

101268 The files *file1* and *file2* shall be ordered in the collating sequence of *sort -b* on the fields on which  
 101269 they shall be joined, by default the first in each line. All selected output shall be written in the  
 101270 same collating sequence.

101271 The default input field separators shall be <blank> characters. In this case, multiple separators  
 101272 shall count as one field separator, and leading separators shall be ignored. The default output  
 101273 field separator shall be a <space>.

101274 The field separator and collating sequence can be changed by using the **-t** option (see below).

101275 If the same key appears more than once in either file, all combinations of the set of remaining  
 101276 fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines  
 101277 encountered.

101278 If the input files are not in the appropriate collating sequence, the results are unspecified.

101279 **OPTIONS**

101280 The *join* utility shall conform to XBD [Section 12.2](#) (on page 215).

101281 The following options shall be supported:

101282 **-a** *file\_number*

101283 Produce a line for each unpairable line in file *file\_number*, where *file\_number* is 1 or  
 101284 2, in addition to the default output. If both **-a1** and **-a2** are specified, all unpairable  
 101285 lines shall be output.

101286 **-e** *string* Replace empty output fields in the list selected by **-o** with the string *string*.

101287 **-o** *list* Construct the output line to comprise the fields specified in *list*, each element of  
 101288 which shall have one of the following two forms:

101289 1. *file\_number.field*, where *file\_number* is a file number and *field* is a decimal  
 101290 integer field number

101291 2. 0 (zero), representing the join field

101292 The elements of *list* shall be either <comma>-separated or <blank>-separated, as  
 101293 specified in Guideline 8 of XBD [Section 12.2](#) (on page 215). The fields specified by  
 101294 *list* shall be written for all selected output lines. Fields selected by *list* that do not  
 101295 appear in the input shall be treated as empty output fields. (See the **-e** option.)  
 101296 Only specifically requested fields shall be written. The application shall ensure that  
 101297 *list* is a single command line argument.

101298        **-t char**        Use character *char* as a separator, for both input and output. Every appearance of  
 101299        *char* in a line shall be significant. When this option is specified, the collating  
 101300        sequence shall be the same as *sort* without the **-b** option.

101301        **-v file\_number**  
 101302                    Instead of the default output, produce a line only for each unpairable line in  
 101303                    *file\_number*, where *file\_number* is 1 or 2. If both **-v1** and **-v2** are specified, all  
 101304                    unpairable lines shall be output.

101305        **-1 field**        Join on the *fieldth* field of file 1. Fields are decimal integers starting with 1.

101306        **-2 field**        Join on the *fieldth* field of file 2. Fields are decimal integers starting with 1.

#### 101307 OPERANDS

101308        The following operands shall be supported:

101309        *file1, file2*     A pathname of a file to be joined. If either of the *file1* or *file2* operands is '-', the  
 101310        standard input shall be used in its place.

#### 101311 STDIN

101312        The standard input shall be used only if the *file1* or *file2* operand is '-'. See the INPUT FILES  
 101313        section.

#### 101314 INPUT FILES

101315        The input files shall be text files.

#### 101316 ENVIRONMENT VARIABLES

101317        The following environment variables shall affect the execution of *join*:

101318        **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 101319                    (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 101320                    variables used to determine the values of locale categories.)

101321        **LC\_ALL**         If set to a non-empty string value, override the values of all the other  
 101322        internationalization variables.

101323        **LC\_COLLATE**

101324                    Determine the locale of the collating sequence *join* expects to have been used when  
 101325                    the input files were sorted.

101326        **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
 101327                    characters (for example, single-byte as opposed to multi-byte characters in  
 101328                    arguments and input files).

101329        **LC\_MESSAGES**

101330                    Determine the locale that should be used to affect the format and contents of  
 101331                    diagnostic messages written to standard error.

101332 XSI        **NLSPATH**     Determine the location of messages objects and message catalogs.

#### 101333 ASYNCHRONOUS EVENTS

101334        Default.

#### 101335 STDOUT

101336        The *join* utility output shall be a concatenation of selected character fields. When the **-o** option  
 101337        is not specified, the output shall be:

101338        "*%s%s%s\n*", *<join field>*, *<other file1 fields>*,  
 101339                    *<other file2 fields>*

101340        If the join field is not the first field in a file, the *<other file fields>* for that file shall be:

101341 <fields preceding join field>, <fields following join field>

101342 When the **-o** option is specified, the output format shall be:

101343 "%s\n", <concatenation of fields>

101344 where the concatenation of fields is described by the **-o** option, above.

101345 For either format, each field (except the last) shall be written with its trailing separator character.  
 101346 If the separator is the default (<blank> characters), a single <space> shall be written after each  
 101347 field (except the last).

#### 101348 **STDERR**

101349 The standard error shall be used only for diagnostic messages.

#### 101350 **OUTPUT FILES**

101351 None.

#### 101352 **EXTENDED DESCRIPTION**

101353 None.

#### 101354 **EXIT STATUS**

101355 The following exit values shall be returned:

101356 0 All input files were output successfully.

101357 >0 An error occurred.

#### 101358 **CONSEQUENCES OF ERRORS**

101359 Default.

#### 101360 **APPLICATION USAGE**

101361 Pathnames consisting of numeric digits or of the form *string.string* should not be specified  
 101362 directly following the **-o** list.

101363 If the collating sequence of the current locale does not have a total ordering of all characters (see  
 101364 XBD Section 7.3.2, on page 139), *join* treats fields that collate equally but are not identical as  
 101365 being the same. If this behavior is not desired, it can be avoided by forcing the use of the POSIX  
 101366 locale (although this means re-sorting the input files into the POSIX locale collating sequence.)

101367 When using *join* to process pathnames, it is recommended that **LC\_ALL**, or at least **LC\_CTYPE**  
 101368 and **LC\_COLLATE**, are set to **POSIX** or **C** in the environment, since pathnames can contain byte  
 101369 sequences that do not form valid characters in some locales, in which case the utility's behavior  
 101370 would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore  
 101371 this problem is avoided.

#### 101372 **EXAMPLES**

101373 The **-o 0** field essentially selects the union of the join fields. For example, given file **phone**:

```
101374 !Name           Phone Number
101375 Don             +1 123-456-7890
101376 Hal            +1 234-567-8901
101377 Yasushi        +2 345-678-9012
```

101378 and file **fax**:

```
101379 !Name           Fax Number
101380 Don             +1 123-456-7899
101381 Keith          +1 456-789-0122
101382 Yasushi        +2 345-678-9011
```

101383 (where the large expanses of white space are meant to each represent a single <tab>), the  
101384 command:

```
101385 join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax
```

101386 (where <tab> is a literal <tab> character) would produce:

| 101387 | !Name   | Phone Number    | Fax Number      |
|--------|---------|-----------------|-----------------|
| 101388 | Don     | +1 123-456-7890 | +1 123-456-7899 |
| 101389 | Hal     | +1 234-567-8901 | (unknown)       |
| 101390 | Keith   | (unknown)       | +1 456-789-0122 |
| 101391 | Yasushi | +2 345-678-9012 | +2 345-678-9011 |

101392 Multiple instances of the same key will produce combinatorial results. The following:

```
101393 fa:
101394     a x
101395     a y
101396     a z
101397 fb:
101398     a p
```

101399 will produce:

```
101400 a x p
101401 a y p
101402 a z p
```

101403 And the following:

```
101404 fa:
101405     a b c
101406     a d e
101407 fb:
101408     a w x
101409     a y z
101410     a o p
```

101411 will produce:

```
101412 a b c w x
101413 a b c y z
101414 a b c o p
101415 a d e w x
101416 a d e y z
101417 a d e o p
```

#### 101418 RATIONALE

101419 The **-e** option is only effective when used with **-o** because, unless specific fields are identified  
101420 using **-o**, *join* is not aware of what fields might be empty. The exception to this is the join field,  
101421 but identifying an empty join field with the **-e** string is not historical practice and some scripts  
101422 might break if this were changed.

101423 The 0 field in the **-o** list was adopted from the Tenth Edition version of *join* to satisfy  
101424 international objections that the *join* in the base documents for IEEE Std 1003.2-1992 did not  
101425 support the "full join" or "outer join" described in relational database literature. Although it has  
101426 been possible to include a join field in the output (by default, or by field number using **-o**), the  
101427 join field could not be included for an unpaired line selected by **-a**. The **-o 0** field essentially

- 101428 selects the union of the join fields.
- 101429 This sort of outer join was not possible with the *join* commands in the base documents for  
101430 IEEE Std 1003.2-1992. The `-o 0` field was chosen because it is an upwards-compatible change for  
101431 applications. An alternative was considered: have the join field represent the union of the fields  
101432 in the files (where they are identical for matched lines, and one or both are null for unmatched  
101433 lines). This was not adopted because it would break some historical applications.
- 101434 The ability to specify *file2* as `-` is not historical practice; it was added for completeness.
- 101435 The `-v` option is not historical practice, but was considered necessary because it permitted the  
101436 writing of *only* those lines that do not match on the join field, as opposed to the `-a` option, which  
101437 prints both lines that do and do not match. This additional facility is parallel with the `-v` option  
101438 of *grep*.
- 101439 Some historical implementations have been encountered where a blank line in one of the input  
101440 files was considered to be the end of the file; the description in this volume of POSIX.1-2024 does  
101441 not cite this as an allowable case.
- 101442 Earlier versions of this standard allowed `-j`, `-j1`, `-j2` options, and a form of the `-o` option that  
101443 allowed the *list* option-argument to be multiple arguments. These forms are no longer specified  
101444 by POSIX.1-2024 but may be present in some implementations.
- 101445 **FUTURE DIRECTIONS**
- 101446 None.
- 101447 **SEE ALSO**
- 101448 [\*awk\*](#), [\*comm\*](#), [\*sort\*](#), [\*uniq\*](#)
- 101449 XBD [Section 7.3.2](#) (on page 139), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)
- 101450 **CHANGE HISTORY**
- 101451 First released in Issue 2.
- 101452 **Issue 6**
- 101453 The obsolescent `-j` options and the multi-argument `-o` option are removed in this version.
- 101454 The normative text is reworded to avoid use of the term “must” for application requirements.
- 101455 **Issue 7**
- 101456 Austin Group Interpretation 1003.1-2001 #027 is applied.
- 101457 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 101458 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0109 [963], XCU/TC2-2008/0110 [663],  
101459 XCU/TC2-2008/0111 [971], and XCU/TC2-2008/0112 [885] are applied.
- 101460 **Issue 8**
- 101461 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

101462 **NAME**

101463 kill — terminate or signal processes

101464 **SYNOPSIS**101465 kill [-s *signal\_name*] *pid*...101466 kill -l [*exit\_status*]101467 XSI kill [-*signal\_name*] *pid*...101468 kill [-*signal\_number*] *pid*...101469 **DESCRIPTION**101470 The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.101471 For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function  
101472 defined in the System Interfaces volume of POSIX.1-2024 called with the following arguments:

- 101473 • The value of the *pid* operand shall be used as the *pid* argument.
- 101474 • The *sig* argument is the value specified by the `-s` option, `-signal_number` option, or the  
101475 `-signal_name` option, or by SIGTERM, if none of these options is specified.

101476 **OPTIONS**101477 XSI The *kill* utility shall conform to XBD Section 12.2 (on page 215), except that in the last two  
101478 SYNOPSIS forms, the `-signal_number` and `-signal_name` options are usually more than a single  
101479 character.

101480 The following options shall be supported:

101481 **-l** (The letter ell.) Write all values of *signal\_name* supported by the implementation, if  
101482 no operand is given. If an *exit\_status* operand is given and it is a value of the '?'  
101483 shell special parameter (see Section 2.5.2 (on page 2479) and *wait*) corresponding to  
101484 a process that was terminated or stopped by a signal, the *signal\_name*  
101485 corresponding to the signal that terminated or stopped the process shall be written.  
101486 If an *exit\_status* operand is given and it is the unsigned decimal integer value of a  
101487 signal number, the *signal\_name* (the symbolic constant name without the **SIG** prefix  
101488 defined in the Base Definitions volume of POSIX.1-2024) corresponding to that  
101489 signal shall be written. Otherwise, the results are unspecified.

101490 **-s *signal\_name***

101491 Specify the signal to send, using one of the symbolic names defined in the  
101492 `<signal.h>` header. Values of *signal\_name* shall be recognized in a case-independent  
101493 fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be  
101494 recognized, representing the signal value zero. The corresponding signal shall be  
101495 sent instead of SIGTERM.

101496 XSI **-*signal\_name***101497 Equivalent to `-s signal_name`.101498 XSI **-*signal\_number***

101499 Specify a non-negative decimal integer, *signal\_number*, representing the signal to be  
101500 used instead of SIGTERM, as the *sig* argument in the effective call to *kill()*. The  
101501 correspondence between integer values and the *sig* value used is shown in the  
101502 following list.

101503 The effects of specifying any *signal\_number* other than those listed below are  
101504 undefined.

|        |                                                                                                 |         |
|--------|-------------------------------------------------------------------------------------------------|---------|
| 101505 | 0                                                                                               | 0       |
| 101506 | 1                                                                                               | SIGHUP  |
| 101507 | 2                                                                                               | SIGINT  |
| 101508 | 3                                                                                               | SIGQUIT |
| 101509 | 6                                                                                               | SIGABRT |
| 101510 | 9                                                                                               | SIGKILL |
| 101511 | 14                                                                                              | SIGALRM |
| 101512 | 15                                                                                              | SIGTERM |
| 101513 | If the first argument is a negative integer, it shall be interpreted as a <i>-signal_number</i> |         |
| 101514 | option, not as a negative <i>pid</i> operand specifying a process group.                        |         |

### 101515 OPERANDS

101516 The following operands shall be supported:

101517 *pid* One of the following:

- 101518 1. A decimal integer specifying a process or process group to be signaled. The  
101519 process or processes selected by positive, negative, and zero values of the  
101520 *pid* operand shall be as described for the *kill()* function. If process number 0  
101521 is specified, all processes in the current process group shall be signaled. For  
101522 the effects of negative *pid* numbers, see the *kill()* function defined in the  
101523 System Interfaces volume of POSIX.1-2024. If the first *pid* operand is  
101524 negative, it should be preceded by "--" to keep it from being interpreted as  
101525 an option.
  - 101526 2. A job ID (see XBD Section 3.182, on page 57) that identifies a process group  
101527 in the case of a job-control background job, or a process ID in the case of a  
101528 non-job-control background job (if supported), to be signaled. The job ID  
101529 notation is applicable only for invocations of *kill* in the current shell  
101530 execution environment; see Section 2.13 (on page 2522).
- 101531 **Note:** The job ID type of *pid* is only available on systems supporting the User  
101532 Portability Utilities option or supporting non-job-control background  
101533 jobs.

101534 *exit\_status* A decimal integer specifying a signal number or the exit status of a process  
101535 terminated by a signal.

### 101536 STDIN

101537 Not used.

### 101538 INPUT FILES

101539 None.

### 101540 ENVIRONMENT VARIABLES

101541 The following environment variables shall affect the execution of *kill*:

101542 *LANG* Provide a default value for the internationalization variables that are unset or null.  
101543 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
101544 variables used to determine the values of locale categories.)

101545 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
101546 internationalization variables.



- 101547 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 101548 characters (for example, single-byte as opposed to multi-byte characters in  
 101549 arguments).
- 101550 *LC\_MESSAGES*  
 101551 Determine the locale that should be used to affect the format and contents of  
 101552 diagnostic messages written to standard error.
- 101553 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 101554 **ASYNCHRONOUS EVENTS**  
 101555 Default.
- 101556 **STDOUT**  
 101557 When the `-l` option is not specified, the standard output shall not be used.  
 101558 When the `-l` option is specified, the symbolic name of each signal shall be written in the  
 101559 following format:  
 101560 `"%s%c", <signal_name>, <separator>`  
 101561 where the `<signal_name>` is in uppercase, without the **SIG** prefix, and the `<separator>` shall be  
 101562 either a `<newline>` or a `<space>`. For the last signal written, `<separator>` shall be a `<newline>`.  
 101563 When both the `-l` option and `exit_status` operand are specified, the symbolic name of the  
 101564 corresponding signal shall be written in the following format:  
 101565 `"%s\n", <signal_name>`
- 101566 **STDERR**  
 101567 The standard error shall be used only for diagnostic messages.
- 101568 **OUTPUT FILES**  
 101569 None.
- 101570 **EXTENDED DESCRIPTION**  
 101571 None.
- 101572 **EXIT STATUS**  
 101573 The following exit values shall be returned:  
 101574 0 The `-l` option was specified and the output specified in **STDOUT** was successfully written  
 101575 to standard output; or, the `-l` option was not specified, at least one matching process was  
 101576 found for each `pid` operand, and the specified signal was successfully processed for at least  
 101577 one matching process.  
 101578 >0 An error occurred.
- 101579 **CONSEQUENCES OF ERRORS**  
 101580 Default.

101581 **APPLICATION USAGE**

101582 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

101583 Process numbers can be found by using *ps*.

101584 The use of job ID notation is not dependent on job control being enabled. When job control has  
 101585 been disabled using *set +m*, *kill* can still be used to signal the process group associated with a  
 101586 job-control background job, or the process ID associated with a non-control background job (if  
 101587 supported), using

```
101588 kill %<background job number>
```

101589 See also the RATIONALE for *jobs* and *wait*.

101590 The job ID notation is not required to work as expected when *kill* is operating in its own utility  
 101591 execution environment. In either of the following examples:

```
101592 nohup kill %1 &  
101593 system("kill %1");
```

101594 the *kill* operates in a different environment and does not share the shell's understanding of job  
 101595 numbers.

101596 **EXAMPLES**

101597 Any of the commands:

```
101598 kill -9 100 -165  
101599 kill -s kill 100 -165  
101600 kill -s KILL 100 -165
```

101601 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose  
 101602 process group ID is 165, assuming the sending process has permission to send that signal to the  
 101603 specified processes, and that they exist.

101604 The System Interfaces volume of POSIX.1-2024 and this volume of POSIX.1-2024 do not require  
 101605 specific signal numbers for any *signal\_names*. Even the *-signal\_number* option provides symbolic  
 101606 (although numeric) names for signals. If a process is terminated by a signal, its exit status  
 101607 indicates the signal that killed it, but the exact values are not specified. The *kill -l* option,  
 101608 however, can be used to map decimal signal numbers and exit status values into the name of a  
 101609 signal. The following example reports the status of a terminated job:

```
101610 job  
101611 stat=$?  
101612 if [ $stat -eq 0 ]  
101613 then  
101614     echo job completed successfully.  
101615 elif [ $stat -gt 128 ]  
101616 then  
101617     echo job terminated by signal SIG$(kill -l $stat).  
101618 else  
101619     echo job terminated with error code $stat.  
101620 fi
```

101621 To send the default signal to a process group (say 123), an application should use a command  
 101622 similar to one of the following:

```
101623 kill -s TERM -- -123  
101624 kill -- -123
```

101625 **RATIONALE**

101626 The `-l` option originated from the C shell, and is also implemented in the KornShell. The C shell  
 101627 output can consist of multiple output lines because the signal names do not always fit on a  
 101628 single line on some terminal screens. The KornShell output also included the implementation-  
 101629 defined signal numbers and was considered by the standard developers to be too difficult for  
 101630 scripts to parse conveniently. The specified output format is intended not only to accommodate  
 101631 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing  
 101632 on systems for which this is appropriate.

101633 An early proposal invented the name `SIGNULL` as a *signal\_name* for signal 0 (used by the System  
 101634 Interfaces volume of POSIX.1-2024 to test for the existence of a process without sending it a  
 101635 signal). Since the *signal\_name* 0 can be used in this case unambiguously, `SIGNULL` has been  
 101636 removed.

101637 An early proposal also required symbolic *signal\_names* to be recognized with or without the **SIG**  
 101638 prefix. Historical versions of *kill* have not written the **SIG** prefix for the `-l` option and have not  
 101639 recognized the **SIG** prefix on *signal\_names*. Since neither applications portability nor ease-of-use  
 101640 would be improved by requiring this extension, it is no longer required.

101641 To avoid an ambiguity of an initial negative number argument specifying either a signal number  
 101642 or a process group, POSIX.1-2024 mandates that it is always considered the former by  
 101643 implementations that support the XSI option. It also requires that conforming applications  
 101644 always use the `--` options terminator argument when specifying a process group.

101645 The `-s` option was added in response to international interest in providing some form of *kill* that  
 101646 meets the Utility Syntax Guidelines.

101647 The job ID notation is not required to work as expected when *kill* is operating in its own utility  
 101648 execution environment. In either of the following examples:

```
101649 nohup kill %1 &  
101650 system("kill %1");
```

101651 the *kill* operates in a different environment and does not understand how the shell has managed  
 101652 its job numbers.

101653 **FUTURE DIRECTIONS**

101654 None.

101655 **SEE ALSO**

101656 [Chapter 2](#) (on page 2472), *ps*, *wait*

101657 [XBD Section 3.182](#) (on page 57), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215), [<signal.h>](#)

101658 [XSH \*kill\*\( \)](#)

101659 **CHANGE HISTORY**

101660 First released in Issue 2.

101661 **Issue 6**

101662 The obsolescent versions of the SYNOPSIS are turned into non-obsolescent features of the XSI  
 101663 option, corresponding to a similar change in the *trap* special built-in.

101664 **Issue 7**

101665 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

101666 **Issue 8**

101667 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
101668 this utility is required to be intrinsic.

101669 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

101670 Austin Group Defect 1254 is applied, clarifying the *-I* option with regard to an *exit\_status*  
101671 operand corresponding to a stopped process, changing ``job control job ID'' to ``job ID'', and  
101672 adding a paragraph to the RATIONALE section.

101673 Austin Group Defect 1260 is applied, changing the SYNOPSIS and EXAMPLES sections in  
101674 relation to the *-s* option, and the RATIONALE section in relation to the use of "--" when  
101675 specifying a process group.

101676 Austin Group Defect 1504 is applied, changing the EXIT STATUS section.

101677 **NAME**101678 lex — generate programs for lexical tasks (**DEVELOPMENT**)101679 **SYNOPSIS**101680 CD `lex [-t] [-n|-v] [file...]`101681 **DESCRIPTION**

101682 The *lex* utility shall generate C programs to be used in lexical processing of character input, and  
 101683 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code  
 101684 and conform to the ISO C standard, without depending on any undefined, unspecified, or  
 101685 implementation-defined behavior, except in cases where the code is copied directly from the  
 101686 supplied source, or in cases that are documented by the implementation. Usually, the *lex* utility  
 101687 shall write the program it generates to the file `lex.yy.c`; the state of this file is unspecified if *lex*  
 101688 exits with a non-zero exit status. See the EXTENDED DESCRIPTION section for a complete  
 101689 description of the *lex* input language.

101690 **OPTIONS**101691 The *lex* utility shall conform to XBD [Section 12.2](#) (on page 215), except for Guideline 9.

101692 The following options shall be supported:

- 101693 **-n** Suppress the summary of statistics usually written with the `-v` option. If no table  
 101694 sizes are specified in the *lex* source code and the `-v` option is not specified, then `-n`  
 101695 is implied.
- 101696 **-t** Write the resulting program to standard output instead of `lex.yy.c`.
- 101697 **-v** Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*  
 101698 table sizes in [Definitions in lex](#) (on page 3039).) If the `-t` option is specified and `-n`  
 101699 is not specified, this report shall be written to standard error. If table sizes are  
 101700 specified in the *lex* source code, and if the `-n` option is not specified, the `-v` option  
 101701 may be enabled.

101702 **OPERANDS**

101703 The following operand shall be supported:

- 101704 *file* A pathname of an input file. If more than one such *file* is specified, all files shall be  
 101705 concatenated to produce a single *lex* program. If no *file* operands are specified, or if  
 101706 a *file* operand is `'-'`, the standard input shall be used.

101707 **STDIN**

101708 The standard input shall be used if no *file* operands are specified, or if a *file* operand is `'-'`. See  
 101709 INPUT FILES.

101710 **INPUT FILES**

101711 The input files shall be text files containing *lex* source code, as described in the EXTENDED  
 101712 DESCRIPTION section.

101713 **ENVIRONMENT VARIABLES**101714 The following environment variables shall affect the execution of *lex*:

- 101715 *LANG* Provide a default value for the internationalization variables that are unset or null.
- 101716 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 101717 internationalization variables.
- 101718 *LC\_COLLATE*  
 101719 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 101720 character collating elements within regular expressions.

101721 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 101722 characters (for example, single-byte as opposed to multi-byte characters in  
 101723 arguments and input files), and the behavior of character classes within regular  
 101724 expressions.

101725 **LC\_MESSAGES**  
 101726 Determine the locale that should be used to affect the format and contents of  
 101727 diagnostic messages written to standard error.

101728 XSI **NLSPATH** Determine the location of messages objects and message catalogs.  
 101729 If the values (if any) of the *LANG*, *LC\_COLLATE*, *LC\_CTYPE*, and *LC\_ALL* variables result in the  
 101730 locale in effect for the *LC\_CTYPE* or *LC\_COLLATE* category not being the POSIX locale, the  
 101731 behavior is unspecified. (See XBD Section 8.2 (on page 169) for the precedence of  
 101732 internationalization variables used to determine the values of locale categories.)

### 101733 ASYNCHRONOUS EVENTS

101734 Default.

### 101735 STDOUT

101736 If the `-t` option is specified, the text file of C source code output of *lex* shall be written to  
 101737 standard output.

101738 If the `-t` option is not specified:

- 101739 • Implementation-defined informational, error, and warning messages concerning the  
 101740 contents of *lex* source code input shall be written to either the standard output or standard  
 101741 error.
- 101742 • If the `-v` option is specified and the `-n` option is not specified, *lex* statistics shall also be  
 101743 written to either the standard output or standard error, in an implementation-defined  
 101744 format. These statistics may also be generated if table sizes are specified with a '%'   
 101745 operator in the *Definitions* section, as long as the `-n` option is not specified.

### 101746 STDERR

101747 If the `-t` option is specified, implementation-defined informational, error, and warning messages  
 101748 concerning the contents of *lex* source code input shall be written to the standard error.

101749 If the `-t` option is not specified:

- 101750 1. Implementation-defined informational, error, and warning messages concerning the  
 101751 contents of *lex* source code input shall be written to either the standard output or  
 101752 standard error.
- 101753 2. If the `-v` option is specified and the `-n` option is not specified, *lex* statistics shall also be  
 101754 written to either the standard output or standard error, in an implementation-defined  
 101755 format. These statistics may also be generated if table sizes are specified with a '%'   
 101756 operator in the *Definitions* section, as long as the `-n` option is not specified.

### 101757 OUTPUT FILES

101758 A text file containing C source code shall be written to **lex.yy.c**, or to the standard output if the `-t`  
 101759 option is present.

### 101760 EXTENDED DESCRIPTION

101761 Each input file shall contain *lex* source code, which is a table of regular expressions with  
 101762 corresponding actions in the form of C program fragments.

101763 When **lex.yy.c** is compiled and linked with the *lex* library (using the `-ll` operand with *c17*), the  
 101764 resulting program shall read character input from the standard input and shall partition it into

101765 strings that match the given expressions.

101766 When an expression is matched, these actions shall occur:

- 101767 • The input string that was matched shall be left in *ytext* as a null-terminated string; *ytext*
- 101768 shall either be an external character array or a pointer to a character string. As explained in
- 101769 [Definitions in lex](#), the type can be explicitly selected using the `%array` or `%pointer`
- 101770 declarations, but the default is implementation-defined.
- 101771 • The external `int yyleng` shall be set to the length of the matching string.
- 101772 • The expression's corresponding program fragment, or action, shall be executed.

101773 During pattern matching, *lex* shall search the set of patterns for the single longest possible

101774 match. Among rules that match the same number of characters, the rule given first shall be

101775 chosen.

101776 The general format of *lex* source shall be:

```
101777     Definitions
101778     %%
101779     Rules
101780     %%
101781     UserSubroutines
```

101782 The first "%%" is required to mark the beginning of the rules (regular expressions and actions);

101783 the second "%%" is required only if user subroutines follow.

101784 Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program

101785 fragment and shall be copied to the external definition area of the `lex.yy.c` file. Similarly,

101786 anything in the *Definitions* section included between delimiter lines containing only "%{" and

101787 "%}" shall also be copied unchanged to the external definition area of the `lex.yy.c` file.

101788 Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing

101789 at the beginning of the *Rules* section before any rules are specified shall be written to `lex.yy.c`

101790 after the declarations of variables for the `yylex()` function and before the first line of code in

101791 `yylex()`. Thus, user variables local to `yylex()` can be declared here, as well as application code to

101792 execute upon entry to `yylex()`.

101793 The action taken by *lex* when encountering any input beginning with a <blank> or within "%{"

101794 and "%}" delimiter lines appearing in the *Rules* section but coming after one or more rules is

101795 undefined. The presence of such input may result in an erroneous definition of the `yylex()`

101796 function.

101797 C-language code in the input shall not contain C-language trigraphs. The C-language code

101798 within "%{" and "%}" delimiter lines shall not contain any lines consisting only of "%}", or

101799 only of "%%".

## 101800 **Definitions in lex**

101801 *Definitions* appear before the first "%%" delimiter. Any line in this section not contained between

101802 "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a *lex*

101803 substitution string. The format of these lines shall be:

```
101804     name substitute
```

101805 If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is

101806 undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name*

101807 string shall be recognized in this context only when the braces are provided and when it does

101808 not appear within a bracket expression or within double-quotes.

101809 In the *Definitions* section, any line beginning with a <percent-sign> ('%') character and followed  
 101810 by an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions.  
 101811 Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall  
 101812 define a set of exclusive start conditions. When the generated scanner is in a %s state, patterns  
 101813 with no state specified shall be also active; in a %x state, such patterns shall not be active. The  
 101814 rest of the line, after the first word, shall be considered to be one or more <blank>-separated  
 101815 names of start conditions. Start condition names shall be constructed in the same way as  
 101816 definition names. Start conditions can be used to restrict the matching of regular expressions to  
 101817 one or more states as described in [Regular Expressions in lex](#) (on page 3041).

101818 Implementations shall accept either of the following two mutually-exclusive declarations in the  
 101819 *Definitions* section:

101820 **%array** Declare the type of *yytext* to be a null-terminated character array.

101821 **%pointer** Declare the type of *yytext* to be a pointer to a null-terminated character string.

101822 The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of  
 101823 the scanner source file (that is, via an **extern**), the application shall include the appropriate  
 101824 **%array** or **%pointer** declaration in the scanner source file.

101825 Implementations shall accept declarations in the *Definitions* section for setting certain internal  
 101826 table sizes. The declarations are shown in the following table.

101827 **Table 3-11** Table Size Declarations in *lex*

| Declaration        | Description                        | Minimum Value |
|--------------------|------------------------------------|---------------|
| 101828 %p <i>n</i> | Number of positions                | 2 500         |
| 101829 %n <i>n</i> | Number of states                   | 500           |
| 101830 %a <i>n</i> | Number of transitions              | 2 000         |
| 101831 %e <i>n</i> | Number of parse tree nodes         | 1 000         |
| 101832 %k <i>n</i> | Number of packed character classes | 1 000         |
| 101833 %o <i>n</i> | Size of the output array           | 3 000         |

101835 In the table, *n* represents a positive decimal integer, preceded by one or more <blank>  
 101836 characters. The exact meaning of these table size numbers is implementation-defined. The  
 101837 implementation shall document how these numbers affect the *lex* utility and how they are  
 101838 related to any output that may be generated by the implementation should limitations be  
 101839 encountered during the execution of *lex*. It shall be possible to determine from this output  
 101840 which of the table size values needs to be modified to permit *lex* to successfully generate tables  
 101841 for the input language. The values in the column Minimum Value represent the lowest values  
 101842 conforming implementations shall provide.

### 101843 Rules in *lex*

101844 The rules in *lex* source files are a table in which the left column contains regular expressions and  
 101845 the right column contains actions (C program fragments) to be executed when the expressions  
 101846 are recognized.

101847 *ERE action*

101848 *ERE action*

101849 ...

101850 The extended regular expression (ERE) portion of a row shall be separated from *action* by one or  
 101851 more <blank> characters. A regular expression containing <blank> characters shall be



101852 recognized under one of the following conditions:

- 101853 • The entire expression appears within double-quotes.
- 101854 • The <blank> characters appear within double-quotes or square brackets.
- 101855 • Each <blank> is preceded by a <backslash> character.

### 101856 **User Subroutines in lex**

101857 Anything in the user subroutines section shall be copied to `lex.yy.c` following `yylex()`.

### 101858 **Regular Expressions in lex**

101859 The *lex* utility shall support the set of extended regular expressions (see XBD [Section 9.4](#), on page  
101860 187), with the following additions and exceptions to the syntax:

101861 ". . ." Any string enclosed in double-quotes shall represent the characters within the  
101862 double-quotes as themselves, except that <backslash>-escapes (which appear in  
101863 the following table) shall be recognized. Any <backslash>-escape sequence shall be  
101864 terminated by the closing quote. For example, "\01"1" represents a single  
101865 string: the octal value 1 followed by the character '1'.

101866 <state>*r*, <state1, state2, . . .>*r*

101867 The regular expression *r* shall be matched only when the program is in one of the  
101868 start conditions indicated by *state*, *state1*, and so on; see [Actions in lex](#) (on page  
101869 3043). (As an exception to the typographical conventions of the rest of this volume  
101870 of POSIX.1-2024, in this case <state> does not represent a metavariable, but the  
101871 literal angle-bracket characters surrounding a symbol.) The start condition shall be  
101872 recognized as such only at the beginning of a regular expression.

101873 *r/x* The regular expression *r* shall be matched only if it is followed by an occurrence of  
101874 regular expression *x* (*x* is the instance of trailing context, further defined below).  
101875 The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches  
101876 the beginning of *x*, the result is unspecified. The *r* expression cannot include  
101877 further trailing context or the '\$' (match-end-of-line) operator; *x* cannot include  
101878 the '^' (match-beginning-of-line) operator, nor trailing context, nor the '\$'  
101879 operator. That is, only one occurrence of trailing context is allowed in a *lex* regular  
101880 expression, and the '^' operator only can be used at the beginning of such an  
101881 expression.

101882 {*name*} When *name* is one of the substitution symbols from the *Definitions* section, the  
101883 string, including the enclosing braces, shall be replaced by the *substitute* value. The  
101884 *substitute* value shall be treated in the extended regular expression as if it were  
101885 enclosed in parentheses. No substitution shall occur if {*name*} occurs within a  
101886 bracket expression or within double-quotes.

101887 Within an ERE, a <backslash> character shall be considered to begin an escape sequence as  
101888 specified in the table in XBD [Chapter 5](#) (on page 113) ('\ ', '\a', '\b', '\f', '\n', '\r',  
101889 '\t', '\v'). In addition, the escape sequences in the following table shall be recognized.

101890 A literal <newline> cannot occur within an ERE; the escape sequence '\n' can be used to  
101891 represent a <newline>. A <newline> shall not be matched by a period operator.

101892

Table 3-12 Escape Sequences in *lex*

101893

101894

101895

101896

101897

101898

101899

101900

101901

101902

101903

101904

101905

101906

101907

101908

101909

101910

101911

101912

101913

| Escape Sequence       | Description                                                                                                                                                                                                                                                                                                       | Meaning                                                                                                                                                                                                                                            |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\digits</code>  | A <code>&lt;backslash&gt;</code> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.                                                                        | The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <code>&lt;backslash&gt;</code> for each byte. |
| <code>\xdigits</code> | A <code>&lt;backslash&gt;</code> character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.                                                                         | The character whose encoding is represented by the hexadecimal integer.                                                                                                                                                                            |
| <code>\c</code>       | A <code>&lt;backslash&gt;</code> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 113) ( <code>'\'</code> , <code>'\a'</code> , <code>'\b'</code> , <code>'\f'</code> , <code>'\n'</code> , <code>'\r'</code> , <code>'\t'</code> , <code>'\v'</code> ). | The character <code>'c'</code> , unchanged.                                                                                                                                                                                                        |

101914

101915

101916

**Note:** If a `'\x'` sequence needs to be immediately followed by a hexadecimal digit character, a sequence such as `"\x1" "1"` can be used, which represents a character containing the value 1, followed by the character `'1'`.

101917

101918

101919

The order of precedence given to extended regular expressions for *lex* differs from that specified in XBD Section 9.4 (on page 187). The order of precedence for *lex* shall be as shown in the following table, from high to low.

101920

101921

101922

101923

101924

**Note:** The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

101925

Table 3-13 ERE Precedence in *lex*

101926

101927

101928

101929

101930

101931

101932

101933

101934

101935

101936

| Extended Regular Expression              | Precedence                                               |
|------------------------------------------|----------------------------------------------------------|
| <i>collation-related bracket symbols</i> | <code>[= =]</code> <code>[: :]</code> <code>[. .]</code> |
| <i>escaped characters</i>                | <code>\&lt;special character&gt;</code>                  |
| <i>bracket expression</i>                | <code>[ ]</code>                                         |
| <i>quoting</i>                           | <code>"..."</code>                                       |
| <i>grouping</i>                          | <code>( )</code>                                         |
| <i>definition</i>                        | <code>{name}</code>                                      |
| <i>single-character RE duplication</i>   | <code>*</code> <code>+</code> <code>?</code>             |
| <i>concatenation</i>                     |                                                          |
| <i>interval expression</i>               | <code>{m,n}</code>                                       |
| <i>alternation</i>                       | <code> </code>                                           |

101937

The ERE anchoring operators `'^'` and `'$'` do not appear in the table. With *lex* regular

101938 expressions, these operators are restricted in their use: the '^' operator can only be used at the  
 101939 beginning of an entire regular expression, and the '\$' operator only at the end. The operators  
 101940 apply to the entire regular expression. Thus, for example, the pattern "(^abc) | (def\$)" is  
 101941 undefined; it can instead be written as two separate rules, one with the regular expression  
 101942 "^abc" and one with "def\$", which share a common action via the special '|' action (see  
 101943 below). If the pattern were written "^abc|def\$", it would match either "abc" or "def" on a  
 101944 line by itself.

101945 Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex*  
 101946 implementations. An example of embedded anchoring would be for patterns such as  
 101947 "(^| )foo( |\$)" to match "foo" when it exists as a complete word. This functionality can  
 101948 be obtained using existing *lex* features:

```
101949 ^foo/[ \n]      |  
101950 "foo"/[ \n]    /* Found foo as a separate word. */
```

101951 Note also that '\$' is a form of trailing context (it is equivalent to "/\n") and as such cannot be  
 101952 used with regular expressions containing another instance of the operator (see the preceding  
 101953 discussion of trailing context).

101954 The additional regular expressions trailing-context operator '/' can be used as an ordinary  
 101955 character if presented within double-quotes, "/"; preceded by a <backslash>, "\/"; or within a  
 101956 bracket expression, "[/]". The start-condition '<' and '>' operators shall be special only in a  
 101957 start condition at the beginning of a regular expression; elsewhere in the regular expression they  
 101958 shall be treated as ordinary characters.

#### 101959 **Actions in lex**

101960 The action to be taken when an ERE is matched can be a C program fragment or the special  
 101961 actions described below; the program fragment can contain one or more C statements, and can  
 101962 also include special actions. The empty C statement ';' shall be a valid action; any string in the  
 101963 **lex.yy.c** input that matches the pattern portion of such a rule is effectively ignored or skipped.  
 101964 However, the absence of an action shall not be valid, and the action *lex* takes in such a condition  
 101965 is undefined.

101966 The specification for an action, including C statements and special actions, can extend across  
 101967 several lines if enclosed in braces:

```
101968 ERE <one or more blanks> { program statement  
101969                          program statement }
```

101970 The program statements shall not contain unbalanced curly brace preprocessing tokens.

101971 The default action when a string in the input to a **lex.yy.c** program is not matched by any  
 101972 expression shall be to copy the string to the output. Because the default behavior of a program  
 101973 generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that  
 101974 has just "%%" shall generate a C program that simply copies the input to the output unchanged.

101975 Four special actions shall be available:

```
101976 | ECHO; REJECT; BEGIN
```

101977 | The action '|' means that the action for the next rule is the action for this rule.  
 101978 Unlike the other three actions, '|' cannot be enclosed in braces or be  
 101979 <semicolon>-terminated; the application shall ensure that it is specified alone, with  
 101980 no other actions.

101981       **ECHO;**       Write the contents of the string *yytext* on the output.

101982       **REJECT;**      Usually only a single expression is matched by a given string in the input.  
 101983       **REJECT** means “continue to the next expression that matches the current input”,  
 101984       and shall cause whatever rule was the second choice after the current rule to be  
 101985       executed for the same input. Thus, multiple rules can be matched and executed for  
 101986       one input string or overlapping input strings. For example, given the regular  
 101987       expressions "xyz" and "xy" and the input "xyz", usually only the regular  
 101988       expression "xyz" would match. The next attempted match would start after **z**. If  
 101989       the last action in the "xyz" rule is **REJECT**, both this rule and the "xy" rule  
 101990       would be executed. The **REJECT** action may be implemented in such a fashion that  
 101991       flow of control does not continue after it, as if it were equivalent to a **goto** to  
 101992       another part of *yylex()*. The use of **REJECT** may result in somewhat larger and  
 101993       slower scanners.

101994       **BEGIN**        The action:  
 101995                    BEGIN *newstate*;  
 101996                    switches the state (start condition) to *newstate*. If the string *newstate* has not been  
 101997                    declared previously as a start condition in the *Definitions* section, the results are  
 101998                    unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.

101999       The functions or macros described below are accessible to user code included in the *lex* input. It  
 102000       is unspecified whether they appear in the C code output of *lex*, or are accessible only through the  
 102001       **-ll** operand to *c17* (the *lex* library).

102002       **int yylex(void)**  
 102003                    Performs lexical analysis on the input; this is the primary function generated by the *lex*  
 102004                    utility. The function shall return zero when the end of input is reached; otherwise, it shall  
 102005                    return non-zero values (tokens) determined by the actions that are selected.

102006       **int yymore(void)**  
 102007                    When called, indicates that when the next input string is recognized, it is to be appended to  
 102008                    the current value of *yytext* rather than replacing it; the value in *yylen* shall be adjusted  
 102009                    accordingly.

102010       **int yyless(int n)**  
 102011                    Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters as  
 102012                    if they had not been read; the value in *yylen* shall be adjusted accordingly.

102013       **int input(void)**  
 102014                    Returns the next character from the input, or zero on end-of-file. It shall obtain input from  
 102015                    the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning  
 102016                    has begun, the effect of altering the value of *yyin* is undefined. The character read shall be  
 102017                    removed from the input stream of the scanner without any processing by the scanner.

102018       **int unput(int c)**  
 102019                    Returns the character 'c' to the input; *yytext* and *yylen* are undefined until the next  
 102020                    expression is matched. The result of using *unput()* for more characters than have been input  
 102021                    is unspecified.

102022       The following functions shall appear only in the *lex* library accessible through the **-ll** operand;  
 102023       they can therefore be redefined by a conforming application:

102024       **int yywrap(void)**  
 102025                    Called by *yylex()* at end-of-file; the default *yywrap()* shall always return 1. If the application  
 102026                    requires *yylex()* to continue processing with another source of input, then the application

102027 can include a function *yywrap()*, which associates another file with the external variable  
 102028 **FILE** \* *yyin* and shall return a value of zero.

102029 **int** *main*(**int** *argc*, **char** \**argv*[])

102030 Calls *yylex()* to perform lexical analysis, then exits. The user code can contain *main()* to  
 102031 perform application-specific operations, calling *yylex()* as applicable.

102032 Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin  
 102033 with the prefix **yy** or **YY**.

#### 102034 EXIT STATUS

102035 The following exit values shall be returned:

102036 0 Successful completion.

102037 >0 An error occurred.

#### 102038 CONSEQUENCES OF ERRORS

102039 Default.

#### 102040 APPLICATION USAGE

102041 Conforming applications are warned that in the *Rules* section, an ERE without an action is not  
 102042 acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or  
 102043 runtime errors.

102044 The purpose of *input()* is to take characters off the input stream and discard them as far as the  
 102045 lexical analysis is concerned. A common use is to discard the body of a comment once the  
 102046 beginning of a comment is recognized.

102047 The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex*  
 102048 source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer  
 102049 interpret the regular expressions given in the *lex* source according to the environment specified  
 102050 when the lexical analyzer is executed, but this is not possible with the current *lex* technology.  
 102051 Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the  
 102052 lexical requirements of the input language being described, which is frequently locale-specific  
 102053 anyway. (For example, writing an analyzer that is used for French text is not automatically  
 102054 useful for processing other languages.)

#### 102055 EXAMPLES

102056 The following is an example of a *lex* program that implements a rudimentary scanner for a  
 102057 Pascal-like syntax:

```

102058 %{
102059 /* Need this for the call to atof() below. */
102060 #include <math.h>
102061 /* Need this for printf(), fopen(), and stdin below. */
102062 #include <stdio.h>
102063 %}
102064
102064 DIGIT    [0-9]
102065 ID       [a-z][a-z0-9]*
102066 %%
102067
102067 {DIGIT}+ {
102068     printf("An integer: %s (%d)\n", yytext,
102069         atoi(yytext));
102070 }
```

```

102071     {DIGIT}+"."{DIGIT}*      {
102072         printf("A float: %s (%g)\n", yytext,
102073             atof(yytext));
102074     }

102075     if|then|begin|end|procedure|function      {
102076         printf("A keyword: %s\n", yytext);
102077     }

102078     {ID}      printf("An identifier: %s\n", yytext);

102079     "+"|"-"|"*"|"|" /"      printf("An operator: %s\n", yytext);

102080     "{"[^]\n}*"      /* Eat up one-line comments. */
102081     [ \t\n]+      /* Eat up white space. */

102082     .      printf("Unrecognized character: %s\n", yytext);
102083     %%

102084     int main(int argc, char *argv[])
102085     {
102086         ++argv, --argc; /* Skip over program name. */
102087         if (argc > 0)
102088             yyin = fopen(argv[0], "r");
102089         else
102090             yyin = stdin;

102091         yylex();
102092     }

```

## 102093 RATIONALE

102094 Even though references to the C language are retained in this description, *lex* may be generalized  
102095 to other languages, as was done at one time for EFL, the Extended FORTRAN Language. Since  
102096 the *lex* input specification is essentially language-independent, versions of this utility could be  
102097 written to produce Ada, Modula-2, or Pascal code, and there are known historical  
102098 implementations that do so.

102099 The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex*  
102100 source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is  
102101 assumed to be presented in the POSIX locale, but input and output are in the locale specified by  
102102 the environment variables), then the tables in the lexical analyzer produced by *lex* would  
102103 interpret EREs specified in the *lex* source in terms of the environment variables specified when  
102104 *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs  
102105 given in the *lex* source according to the environment specified when the lexical analyzer is  
102106 executed, but this is not possible with the current *lex* technology.

102107 The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard  
102108 use of escape sequences.

102109 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
102110 but this has been modified in this version.

102111 There is no detailed output format specification. The observed behavior of *lex* under four  
102112 different historical implementations was that none of these implementations consistently  
102113 reported the line numbers for error and warning messages. Furthermore, there was a desire that  
102114 *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified  
102115 avoids these formatting questions and problems with internationalization.

102116 Although the %x specifier for *exclusive* start conditions is not historical practice, it is believed to  
102117 be a minor change to historical implementations and greatly enhances the usability of *lex*  
102118 programs since it permits an application to obtain the expected functionality with fewer  
102119 statements.

102120 The %array and %pointer declarations were added as a compromise between historical systems.  
102121 The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in  
102122 BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements  
102123 are available for some scanners. Most historical programs should require no change in porting  
102124 from one system to another because the string being referenced is null-terminated in both cases.  
102125 (The method used by *flex* in its case is to null-terminate the token in place by remembering the  
102126 character that used to come right after the token and replacing it before continuing on to the next  
102127 scan.) Multi-file programs with external references to *yytext* outside the scanner source file  
102128 should continue to operate on their historical systems, but would require one of the new  
102129 declarations to be considered strictly portable.

102130 The description of EREs avoids unnecessary duplication of ERE details because their meanings  
102131 within a *lex* ERE are the same as that for the ERE in this volume of POSIX.1-2024.

102132 The reason for the undefined condition associated with text beginning with a <blank> or within  
102133 "%{ " and "%}" delimiter lines appearing in the *Rules* section is historical practice. Both the BSD  
102134 and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the  
102135 beginning) to unreachable areas of the *yylex()* function (the code is written directly after a *break*  
102136 statement). In some cases, the System V *lex* generates an error message or a syntax error,  
102137 depending on the form of indented input.

102138 The intention in breaking the list of functions into those that may appear in *lex.yy.c* versus those  
102139 that only appear in *libl.a* is that only those functions in *libl.a* can be reliably redefined by a  
102140 conforming application.

102141 The descriptions of standard output and standard error are somewhat complicated because  
102142 historical *lex* implementations chose to issue diagnostic messages to standard output (unless *-t*  
102143 was given). POSIX.1-2024 allows this behavior, but leaves an opening for the more expected  
102144 behavior of using standard error for diagnostics. Also, the System V behavior of writing the  
102145 statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The  
102146 programmer can always precisely obtain the desired results by using either the *-t* or *-n* options.

102147 The OPERANDS section does not mention the use of *-* as a synonym for standard input; not all  
102148 historical implementations support such usage for any of the *file* operands.

102149 A description of the *translation table* was deleted from early proposals because of its relatively  
102150 low usage in historical applications.

102151 The change to the definition of the *input()* function that allows buffering of input presents the  
102152 opportunity for major performance gains in some applications.

102153 The following examples clarify the differences between *lex* regular expressions and regular  
102154 expressions appearing elsewhere in this volume of POSIX.1-2024. For regular expressions of the  
102155 form "*r/x*", the string matching *r* is always returned; confusion may arise when the beginning  
102156 of *x* matches the trailing portion of *r*. For example, given the regular expression "*a\*b/cc*" and  
102157 the input "*aaabcc*", *yytext* would contain the string "*aaab*" on this match. But given the  
102158 regular expression "*x\*/xy*" and the input "*xxxxy*", the token *xxx*, not *xx*, is returned by some  
102159 implementations because *xxx* matches "*x\**".

102160 In the rule "*ab\*/bc*", the "*b\**" at the end of *r* extends *r*'s match into the beginning of the  
102161 trailing context, so the result is unspecified. If this rule were "*ab/bc*", however, the rule  
102162 matches the text "*ab*" when it is followed by the text "*bc*". In this latter case, the matching of *r*

- 102163 cannot extend into the beginning of *x*, so the result is specified.
- 102164 **FUTURE DIRECTIONS**
- 102165 None.
- 102166 **SEE ALSO**
- 102167 *c17, ed, yacc*
- 102168 XBD [Chapter 5](#) (on page 113), [Chapter 8](#) (on page 167), [Chapter 9](#) (on page 179), [Section 12.2](#) (on  
102169 page 215)
- 102170 **CHANGE HISTORY**
- 102171 First released in Issue 2.
- 102172 **Issue 6**
- 102173 This utility is marked as part of the C-Language Development Utilities option.
- 102174 The obsolescent `-c` option is removed.
- 102175 The normative text is reworded to avoid use of the term “must” for application requirements.
- 102176 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/14 is applied, removing text describing  
102177 behavior on systems with bytes consisting of more than eight bits.
- 102178 **Issue 7**
- 102179 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for  
102180 generated code to conform to the ISO C standard.
- 102181 Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language  
102182 trigraphs and curly brace preprocessing tokens.
- 102183 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
102184 apply.
- 102185 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 102186 **Issue 8**
- 102187 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 102188 Austin Group Defect 1453 is applied, changing the ENVIRONMENT VARIABLES section.
- 102189 Austin Group Defect 1517 is applied, removing a reference to the `-c` option from the  
102190 RATIONALE section.



102191 **NAME**102192 link — call *link()* function102193 **SYNOPSIS**102194 XSI `link file1 file2`102195 **DESCRIPTION**102196 The *link* utility shall perform the function call:102197 `link(file1, file2);`102198 A user may need appropriate privileges to invoke the *link* utility.102199 **OPTIONS**

102200 None.

102201 **OPERANDS**

102202 The following operands shall be supported:

102203 *file1* The pathname of an existing file.102204 *file2* The pathname of the new directory entry to be created.102205 **STDIN**

102206 Not used.

102207 **INPUT FILES**

102208 Not used.

102209 **ENVIRONMENT VARIABLES**102210 The following environment variables shall affect the execution of *link*:102211 *LANG* Provide a default value for the internationalization variables that are unset or null.  
102212 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
102213 variables used to determine the values of locale categories.)102214 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
102215 internationalization variables.102216 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
102217 characters (for example, single-byte as opposed to multi-byte characters in  
102218 arguments).102219 *LC\_MESSAGES*102220 Determine the locale that should be used to affect the format and contents of  
102221 diagnostic messages written to standard error.102222 *NLSPATH* Determine the location of messages objects and message catalogs.102223 **ASYNCHRONOUS EVENTS**

102224 Default.

102225 **STDOUT**

102226 None.

102227 **STDERR**

102228 The standard error shall be used only for diagnostic messages.

**102229 OUTPUT FILES**

102230 None.

**102231 EXTENDED DESCRIPTION**

102232 None.

**102233 EXIT STATUS**

102234 The following exit values shall be returned:

102235 0 Successful completion.

102236 >0 An error occurred.

**102237 CONSEQUENCES OF ERRORS**

102238 Default.

**102239 APPLICATION USAGE**

102240 None.

**102241 EXAMPLES**

102242 None.

**102243 RATIONALE**

102244 None.

**102245 FUTURE DIRECTIONS**

102246 If this utility is directed to create a new directory entry that contains any bytes that have the  
102247 encoded value of a <newline> character, implementations are encouraged to treat this as an  
102248 error. A future version of this standard may require implementations to treat this as an error.

**102249 SEE ALSO**

102250 *ln*, *unlink*

102251 XBD [Chapter 8](#) (on page 167)

102252 XSH [link\(\)](#)

**102253 CHANGE HISTORY**

102254 First released in Issue 5.

**102255 Issue 8**

102256 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
102257 filenames containing any bytes that have the encoded value of a <newline> character.

102258 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

102259 **NAME**

102260 ln — link files

102261 **SYNOPSIS**102262 ln [-fs] [-L|-P] *source\_file target\_file*102263 ln [-fs] [-L|-P] *source\_file... target\_dir*102264 **DESCRIPTION**

102265 In the first synopsis form, the *ln* utility shall create a new directory entry at the destination path  
 102266 specified by the *target\_file* operand. If the *-s* option is specified, a symbolic link shall be created  
 102267 with the contents specified by the *source\_file* operand (which need not name an existing file);  
 102268 otherwise, a hard link shall be created to the file named by the *source\_file* operand. This first  
 102269 synopsis form shall be assumed when the final operand does not name an existing directory; if  
 102270 more than two operands are specified and the final is not an existing directory, an error shall  
 102271 result.

102272 In the second synopsis form, the *ln* utility shall create a new directory entry for each *source\_file*  
 102273 operand, at a destination path in the existing directory named by *target\_dir*. If the *-s* option is  
 102274 specified, a symbolic link shall be created with the contents specified by each *source\_file* operand  
 102275 (which need not name an existing file); otherwise, a hard link shall be created to each file named  
 102276 by a *source\_file* operand.

102277 If the last operand specifies an existing file of a type not specified by the System Interfaces  
 102278 volume of POSIX.1-2024, the behavior is implementation-defined.

102279 The corresponding destination path for each *source\_file* shall be the concatenation of the target  
 102280 directory pathname, a <slash> character if the target directory pathname did not end in a  
 102281 <slash>, and the last pathname component of the *source\_file*. The second synopsis form shall be  
 102282 assumed when the final operand names an existing directory.

102283 For each *source\_file*:

- 102284 1. If the destination path exists and was created by a previous step, it is unspecified whether  
 102285 *ln* writes a diagnostic message to standard error, does nothing more with the current  
 102286 *source\_file*, and goes on to any remaining *source\_files*; or continues processing the current  
 102287 *source\_file*. If the destination path exists:
  - 102288 a. If the *-f* option is not specified, *ln* shall write a diagnostic message to standard  
 102289 error, do nothing more with the current *source\_file*, and go on to any remaining  
 102290 *source\_files*.
  - 102291 b. If the destination path names the same directory entry as the current *source\_file* *ln*  
 102292 shall write a diagnostic message to standard error, do nothing more with the  
 102293 current *source\_file*, and go on to any remaining *source\_files*.
  - 102294 c. Actions shall be performed equivalent to the *unlink()* function defined in the  
 102295 System Interfaces volume of POSIX.1-2024, called using the destination path as the  
 102296 *path* argument. If this fails for any reason, *ln* shall write a diagnostic message to  
 102297 standard error, do nothing more with the current *source\_file*, and go on to any  
 102298 remaining *source\_files*.
- 102299 2. If the *-s* option is specified, actions shall be performed equivalent to the *symlink()*  
 102300 function with *source\_file* as the *path1* argument and the destination path as the *path2*  
 102301 argument. The *ln* utility shall do nothing more with *source\_file* and shall go on to any  
 102302 remaining files.

- 102303 3. If *source\_file* is a symbolic link:
- 102304 a. If the **-P** option is in effect, actions shall be performed equivalent to the *linkat()*
- 102305 function with *source\_file* as the *path1* argument, the destination path as the *path2*
- 102306 argument, `AT_FDCWD` as the *fd1* and *fd2* arguments, and zero as the *flag*
- 102307 argument.
- 102308 b. If the **-L** option is in effect, actions shall be performed equivalent to the *linkat()*
- 102309 function with *source\_file* as the *path1* argument, the destination path as the *path2*
- 102310 argument, `AT_FDCWD` as the *fd1* and *fd2* arguments, and
- 102311 `AT_SYMLINK_FOLLOW` as the *flag* argument.
- 102312 The *ln* utility shall do nothing more with *source\_file* and shall go on to any remaining files.
- 102313 4. Actions shall be performed equivalent to the *link()* function defined in the System
- 102314 Interfaces volume of POSIX.1-2024 using *source\_file* as the *path1* argument, and the
- 102315 destination path as the *path2* argument.

### 102316 OPTIONS

102317 The *ln* utility shall conform to XBD [Section 12.2](#) (on page 215).

102318 The following options shall be supported:

- 102319 **-f** Force existing destination pathnames to be removed to allow the link.
- 102320 **-L** For each *source\_file* operand that names a file of type symbolic link, create a hard
- 102321 link to the file referenced by the symbolic link.
- 102322 **-P** For each *source\_file* operand that names a file of type symbolic link, create a hard
- 102323 link to the symbolic link itself.
- 102324 **-s** Create symbolic links instead of hard links. If the **-s** option is specified, the **-L** and
- 102325 **-P** options shall be silently ignored.

102326 Specifying more than one of the mutually-exclusive options **-L** and **-P** shall not be considered

102327 an error. The last option specified shall determine the behavior of the utility (unless the **-s**

102328 option causes it to be ignored).

102329 If the **-s** option is not specified and neither a **-L** nor a **-P** option is specified, it is

102330 implementation-defined which of the **-L** and **-P** options is used as the default.

### 102331 OPERANDS

102332 The following operands shall be supported:

- 102333 *source\_file* A pathname of a file to be linked. If the **-s** option is specified, no restrictions on the
- 102334 type of file or on its existence shall be made. If the **-s** option is not specified,
- 102335 whether a directory can be linked is implementation-defined.
- 102336 *target\_file* The pathname of the new directory entry to be created.
- 102337 *target\_dir* A pathname of an existing directory in which the new directory entries are created.

### 102338 STDIN

102339 Not used.

### 102340 INPUT FILES

102341 None.

102342 **ENVIRONMENT VARIABLES**

102343 The following environment variables shall affect the execution of *ln*:

102344 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 102345 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 102346 variables used to determine the values of locale categories.)

102347 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 102348 internationalization variables.

102349 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 102350 characters (for example, single-byte as opposed to multi-byte characters in  
 102351 arguments).

102352 *LC\_MESSAGES*

102353 Determine the locale that should be used to affect the format and contents of  
 102354 diagnostic messages written to standard error.

102355 *XSI NLSPATH* Determine the location of messages objects and message catalogs.

102356 **ASYNCHRONOUS EVENTS**

102357 Default.

102358 **STDOUT**

102359 Not used.

102360 **STDERR**

102361 The standard error shall be used only for diagnostic messages.

102362 **OUTPUT FILES**

102363 None.

102364 **EXTENDED DESCRIPTION**

102365 None.

102366 **EXIT STATUS**

102367 The following exit values shall be returned:

102368 0 Successful completion.

102369 >0 An error occurred.

102370 **CONSEQUENCES OF ERRORS**

102371 Default.

102372 **APPLICATION USAGE**

102373 None.

102374 **EXAMPLES**

102375 None.

102376 **RATIONALE**

102377 The CONSEQUENCES OF ERRORS section does not require *ln -f a b* to remove *b* if a  
 102378 subsequent link operation would fail.

102379 Some historic versions of *ln* (including the one specified by the SVID) unlink the destination file,  
 102380 if it exists, by default. If the mode does not permit writing, these versions prompt for  
 102381 confirmation before attempting the unlink. In these versions the *-f* option causes *ln* not to  
 102382 attempt to prompt for confirmation.

102383 This allows *ln* to succeed in creating links when the target file already exists, even if the file itself

- 102384 is not writable (although the directory must be). Early proposals specified this functionality.
- 102385 This volume of POSIX.1-2024 does not allow the *ln* utility to unlink existing destination paths by  
102386 default for the following reasons:
- 102387 • The *ln* utility has historically been used to provide locking for shell applications, a usage  
102388 that is incompatible with *ln* unlinking the destination path by default. There was no  
102389 corresponding technical advantage to adding this functionality.
  - 102390 • This functionality gave *ln* the ability to destroy the link structure of files, which changes  
102391 the historical behavior of *ln*.
  - 102392 • This functionality is easily replicated with a combination of *rm* and *ln*.
  - 102393 • It is not historical practice in many systems; BSD and BSD-derived systems do not support  
102394 this behavior. Unfortunately, whichever behavior is selected can cause scripts written  
102395 expecting the other behavior to fail.
  - 102396 • It is preferable that *ln* perform in the same manner as the *link()* function, which does not  
102397 permit the target to exist already.
- 102398 This volume of POSIX.1-2024 retains the `-f` option to provide support for shell scripts depending  
102399 on the SVID semantics. It seems likely that shell scripts would not be written to handle  
102400 prompting by *ln* and would therefore have specified the `-f` option.
- 102401 The `-f` option is an undocumented feature of many historical versions of the *ln* utility, allowing  
102402 linking to directories. These versions require modification.
- 102403 Early proposals of this volume of POSIX.1-2024 also required a `-i` option, which behaved like the  
102404 `-i` options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This was not  
102405 historical practice for the *ln* utility and has been omitted.
- 102406 The `-L` and `-P` options allow for implementing both common behaviors of the *ln* utility. Earlier  
102407 versions of this standard did not specify these options and required the behavior now described  
102408 for the `-L` option. Many systems by default or as an alternative provided a non-conforming *ln*  
102409 utility with the behavior now described for the `-P` option. Since applications could not rely on *ln*  
102410 following links in practice, the `-L` and `-P` options were added to specify the desired behavior for  
102411 the application.
- 102412 The `-L` and `-P` options are ignored when `-s` is specified in order to allow an alias to be created to  
102413 alter the default behavior when creating hard links (for example, *alias ln='ln -L'*). They serve no  
102414 purpose when `-s` is specified, since *source\_file* is then just a string to be used as the contents of  
102415 the created symbolic link and need not exist as a file.
- 102416 The specification ensures that *ln a a* with or without the `-f` option will not unlink the file *a*.  
102417 Earlier versions of this standard were unclear in this case.
- 102418 **FUTURE DIRECTIONS**
- 102419 If this utility is directed to create a new directory entry that contains any bytes that have the  
102420 encoded value of a <newline> character, implementations are encouraged to treat this as an  
102421 error. A future version of this standard may require implementations to treat this as an error.
- 102422 **SEE ALSO**
- 102423 *chmod, find, pax, readlink, realpath, rm*
- 102424 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)
- 102425 XSH *link(), unlink()*

102426 **CHANGE HISTORY**

102427 First released in Issue 2.

102428 **Issue 6**102429 The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b  
102430 draft standard.102431 **Issue 7**

102432 Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

102433 SD5-XCU-ERN-27 is applied, adding a new paragraph to the RATIONALE.

102434 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

102435 The **-L** and **-P** options are added to make it implementation-defined whether the *ln* utility  
102436 follows symbolic links.

102437 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0096 [136] is applied.

102438 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0113 [930] is applied.

102439 **Issue 8**102440 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
102441 filenames containing any bytes that have the encoded value of a <newline> character.102442 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.102443 Austin Group Defect 1380 is applied, changing text using the term “link” in line with its  
102444 updated definition.102445 Austin Group Defect 1457 is applied, adding *readlink* and *realpath* to the SEE ALSO section.

102446 Austin Group Defect 1506 is applied, changing the EXIT STATUS section.

102447 **NAME**

102448 locale — get locale-specific information

102449 **SYNOPSIS**

102450 locale [-a|-m]

102451 locale [-ck] name...

102452 **DESCRIPTION**

102453 The *locale* utility shall write information about the current locale environment, or all public  
 102454 locales, to the standard output. For the purposes of this section, a *public locale* is one provided by  
 102455 the implementation that is accessible to the application.

102456 When *locale* is invoked without any arguments, it shall summarize the current locale  
 102457 environment for each locale category as determined by the settings of the environment variables  
 102458 defined in XBD [Chapter 7](#) (on page 127).

102459 When invoked with operands, it shall write values that have been assigned to the keywords in  
 102460 the locale categories, as follows:

- 102461 • Specifying a keyword name shall select the named keyword and the category containing  
 102462 that keyword.
- 102463 • Specifying a category name shall select the named category and all keywords in that  
 102464 category.

102465 **OPTIONS**102466 The *locale* utility shall conform to XBD [Section 12.2](#) (on page 215).

102467 The following options shall be supported:

- 102468 **-a** Write information about all available public locales. The available locales shall  
 102469 include **POSIX**, representing the POSIX locale. The manner in which the  
 102470 implementation determines what other locales are available is implementation-  
 102471 defined.
- 102472 **-c** Write the names of selected locale categories; see the **STDOUT** section. The **-c**  
 102473 option increases readability when more than one category is selected (for example,  
 102474 via more than one keyword name or via a category name). It is valid both with  
 102475 and without the **-k** option.
- 102476 **-k** Write the names and values of selected keywords. The implementation may omit  
 102477 values for some keywords; see the **OPERANDS** section.
- 102478 **-m** Write names of available charmaps; see XBD [Section 6.1](#) (on page 117).

102479 **OPERANDS**

102480 The following operand shall be supported:

- 102481 *name* The name of a locale category as defined in XBD [Chapter 7](#) (on page 127), the name  
 102482 of a keyword in a locale category, or the reserved name **charmap**. The named  
 102483 category or keyword shall be selected for output. If a single *name* represents both a  
 102484 locale category name and a keyword name in the current locale, the results are  
 102485 unspecified. Otherwise, both category and keyword names can be specified as  
 102486 *name* operands, in any sequence. It is implementation-defined whether any  
 102487 keyword values are written for the categories *LC\_CTYPE* and *LC\_COLLATE*.



102488 **STDIN**

102489 Not used.

102490 **INPUT FILES**

102491 None.

102492 **ENVIRONMENT VARIABLES**102493 The following environment variables shall affect the execution of *locale*:

102494 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 102495 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 102496 variables used to determine the values of locale categories.)

102497 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 102498 internationalization variables.

102499 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 102500 characters (for example, single-byte as opposed to multi-byte characters in  
 102501 arguments and input files).

102502 *LC\_MESSAGES*

102503 Determine the locale that should be used to affect the format and contents of  
 102504 diagnostic messages written to standard error.

102505 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

102506 XSI The application shall ensure that the *LANG*, *LC\_\**, and *NLSPATH* environment variables specify  
 102507 the current locale environment to be written out; they shall be used if the **-a** option is not  
 102508 specified.

102509 **ASYNCHRONOUS EVENTS**

102510 Default.

102511 **STDOUT**102512 The *LANG* variable shall be written first using the format:102513 "*LANG*=%s\n", *<value>*102514 If *LANG* is not set or is an empty string, the value is the empty string.

102515 If *locale* is invoked without any options or operands, the names and values of the *LC\_\**  
 102516 environment variables described in this volume of POSIX.1-2024 shall be written to the standard  
 102517 output, one variable per line, and each line using the following format. Only those variables set  
 102518 in the environment and not overridden by *LC\_ALL* shall be written using this format:

102519 "%s=%s\n", *<variable\_name>*, *<value>*

102520 The names of those *LC\_\** variables associated with locale categories defined in this volume of  
 102521 POSIX.1-2024 that are not set in the environment or are overridden by *LC\_ALL* shall be written  
 102522 in the following format:

102523 "%s=\"%s\"\n", *<variable\_name>*, *<implied value>*

102524 The *<implied value>* shall be the name of the locale that has been selected for that category by the  
 102525 implementation, based on the values in *LANG* and *LC\_ALL*, as described in XBD Chapter 8 (on  
 102526 page 167).

102527 The *<value>* and *<implied value>* shown above shall be properly quoted for possible later reentry  
 102528 to the shell. The *<value>* shall not be quoted using double-quotes (so that it can be distinguished  
 102529 by the user from the *<implied value>* case, which always requires double-quotes).

102530 The `LC_ALL` variable shall be written last, using the first format shown above. If it is not set, it  
 102531 shall be written as:

102532 `"LC_ALL=\n"`

102533 If any arguments are specified:

102534 1. If the `-a` option is specified, the names of all the public locales shall be written, each in the  
 102535 following format:

102536 `"%s\n", <locale name>`

102537 2. If the `-c` option is specified, the names of all selected categories shall be written, each in  
 102538 the following format:

102539 `"%s\n", <category name>`

102540 If keywords are also selected for writing (see following items), the category name output  
 102541 shall precede the keyword output for that category.

102542 If the `-c` option is not specified, the names of the categories shall not be written; only the  
 102543 keywords, as selected by the `<name>` operand, shall be written.

102544 3. If the `-k` option is specified, the names and values of selected keywords shall be written.  
 102545 If a value is non-numeric and is not a compound keyword value, it shall be written in the  
 102546 following format:

102547 `"%s=\"%s\"\n", <keyword name>, <keyword value>`

102548 If a value is a non-numeric compound keyword value, it shall either be written in the  
 102549 format:

102550 `"%s=\"%s\"\n", <keyword name>, <keyword value>`

102551 where the `<keyword value>` is a single string of values separated by `<semicolon>`  
 102552 characters, or it shall be written in the format:

102553 `"%s=%s\n", <keyword name>, <keyword value>`

102554 where the `<keyword value>` is encoded as a set of strings, each enclosed in double-  
 102555 quotation-marks, separated by `<semicolon>` characters.

102556 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
 102557 `localedef -f` option when the locale was created shall be written, with the word **charmap** as  
 102558 `<keyword name>`.

102559 If a value is numeric, it shall be written in one of the following formats:

102560 `"%s=%d\n", <keyword name>, <keyword value>`

102561 `"%s=%c%o\n", <keyword name>, <escape character>, <keyword value>`

102562 `"%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>`

102563 where the `<escape character>` is that identified by the **escape\_char** keyword in the current  
 102564 locale; see XBD [Section 7.3](#) (on page 128).

102565 Compound keyword values (list entries) shall be separated in the output by `<semicolon>`  
 102566 characters. When included in keyword values, the `<semicolon>`, `<backslash>`, double-  
 102567 quote, and any control character shall be preceded (escaped) with the escape character.

102568 4. If the `-k` option is not specified, selected keyword values shall be written, each in the  
102569 following format:

102570 "%s\n", <keyword value>

102571 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the  
102572 `localedef -f` option when the locale was created shall be written.

102573 5. If the `-m` option is specified, then a list of all available charmaps shall be written, each in  
102574 the format:

102575 "%s\n", <charmap>

102576 where <charmap> is in a format suitable for use as the option-argument to the `localedef -f`  
102577 option.

#### 102578 **STDERR**

102579 The standard error shall be used only for diagnostic messages.

#### 102580 **OUTPUT FILES**

102581 None.

#### 102582 **EXTENDED DESCRIPTION**

102583 None.

#### 102584 **EXIT STATUS**

102585 The following exit values shall be returned:

102586 0 All the requested information was found and output successfully.

102587 >0 An error occurred.

#### 102588 **CONSEQUENCES OF ERRORS**

102589 Default.

#### 102590 **APPLICATION USAGE**

102591 If the `LANG` environment variable is not set or set to an empty value, or one of the `LC_*`  
102592 environment variables is set to an unrecognized value, the actual locales assumed (if any) are  
102593 implementation-defined as described in XBD [Chapter 8](#) (on page 167).

102594 Implementations are not required to write out the actual values for keywords in the categories  
102595 `LC_CTYPE` and `LC_COLLATE`; however, they must write out the categories (allowing an  
102596 application to determine, for example, which character classes are available).

#### 102597 **EXAMPLES**

102598 In the following examples, the assumption is that locale environment variables are set as  
102599 follows:

102600 LANG=locale\_x

102601 LC\_COLLATE=locale\_y

102602 The command `locale` would result in the following output:

102603 LANG=locale\_x

102604 LC\_CTYPE="locale\_x"

102605 LC\_COLLATE=locale\_y

102606 LC\_TIME="locale\_x"

102607 LC\_NUMERIC="locale\_x"

102608 LC\_MONETARY="locale\_x"

102609 LC\_MESSAGES="locale\_x"

102610 LC\_ALL=

102611 The order of presentation of the categories is not specified by this volume of POSIX.1-2024.

102612 The command:

```
102613 LC_ALL=POSIX locale -ck decimal_point
```

102614 would produce:

```
102615 LC_NUMERIC
102616 decimal_point="."
```

102617 The following command shows an application of *locale* to determine whether a user-supplied  
102618 response is affirmative:

```
102619 printf 'Prompt for response: '
102620 read response
102621 if printf "%s\n" "$response" | grep -Eq -- "${locale yesexpr}"
102622 then
102623     affirmative processing goes here
102624 else
102625     non-affirmative processing goes here
102626 fi
```

#### 102627 RATIONALE

102628 The output for categories *LC\_CTYPE* and *LC\_COLLATE* has been made implementation-defined  
102629 because there is a questionable value in having a shell script receive an entire array of characters.  
102630 It is also difficult to return a logical collation description, short of returning a complete *localedef*  
102631 source.

102632 The **-m** option was included to allow applications to query for the existence of charmaps. The  
102633 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the  
102634 system.

102635 The **-c** option was included for readability when more than one category is selected (for  
102636 example, via more than one keyword name or via a category name). It is valid both with and  
102637 without the **-k** option.

102638 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the  
102639 current locale was created, was included to allow applications needing the information to  
102640 retrieve it.

102641 According to XBD [Section 6.1](#) (on page 117), the standard requires that all supported locales  
102642 must have the same encoding for <period> and <slash>, because these two characters are used  
102643 within the locale-independent pathname resolution sequence. Therefore, it would be an error if  
102644 *locale -a* listed both ASCII and EBCDIC-based locales, since those two encodings do not share  
102645 the same representation for either <period> or <slash>. Any system that supports both  
102646 environments would be expected to provide two POSIX locales, one in either codeset, where  
102647 only the locales appropriate to the current environment can be visible at a time. In an XSI-  
102648 compliant implementation, the *dd* utility is the only portable means for performing conversions  
102649 between the two character sets.

#### 102650 FUTURE DIRECTIONS

102651 None.

#### 102652 SEE ALSO

102653 *localedef*

102654 XBD [Section 6.1](#) (on page 117), [Chapter 7](#) (on page 127), [Chapter 8](#) (on page 167), [Section 12.2](#) (on  
102655 page 215)

102656 **CHANGE HISTORY**

102657 First released in Issue 4.

102658 **Issue 5**

102659 The FUTURE DIRECTIONS section is added.

102660 **Issue 6**

102661 The normative text is reworded to avoid use of the term “must” for application requirements.

102662 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/30 is applied, correcting an editorial error  
102663 in the STDOUT section.

102664 **Issue 7**

102665 Austin Group Interpretations 1003.1-2001 #017, #021, and #088 are applied, clarifying the  
102666 standard output for the `-k` option when `LANG` is not set or is an empty string.

102667 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

102668 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0097 [291] is applied.

102669 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0114 [941] is applied.

102670 **Issue 8**

102671 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.

102672 Austin Group Defect 1262 is applied, changing the EXAMPLES section.

102673 **NAME**

102674 localedef — define locale environment

102675 **SYNOPSIS**102676 localedef [-c] [-f *charmap*] [-i *sourcefile*] [-u *code\_set\_name*] *name*102677 **DESCRIPTION**

102678 The *localedef* utility shall convert source definitions for locale categories into a format usable by  
 102679 the functions and utilities whose operational behavior is determined by the setting of the locale  
 102680 environment variables defined in XBD [Chapter 7](#) (on page 127). It is implementation-defined  
 102681 whether users have the capability to create new locales, in addition to those supplied by the  
 102682 implementation. If the symbolic constant POSIX2\_LOCALEDEF is defined, the system supports  
 102683 XSI the creation of new locales. On XSI-conformant systems, the symbolic constant  
 102684 POSIX2\_LOCALEDEF shall be defined.

102685 The utility shall read source definitions for one or more locale categories belonging to the same  
 102686 locale from the file named in the -i option (if specified) or from standard input.

102687 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or  
 102688 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may  
 102689 restrict the capability to create or modify public locales to users with appropriate privileges.

102690 Each category source definition shall be identified by the corresponding environment variable  
 102691 name and terminated by an **END** *category-name* statement. The following categories shall be  
 102692 supported. In addition, the input may contain source for implementation-defined categories.

102693 *LC\_CTYPE* Defines character classification and case conversion.

102694 *LC\_COLLATE*  
 102695 Defines collation rules.

102696 *LC\_MONETARY*  
 102697 Defines the format and symbols used in formatting of monetary information.

102698 *LC\_NUMERIC*  
 102699 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary  
 102700 numeric editing.

102701 *LC\_TIME* Defines the format and content of date and time information.

102702 *LC\_MESSAGES*  
 102703 Defines the format and values of affirmative and negative responses.

102704 If the *LC\_COLLATE* category defines a collation sequence that does not have a total ordering of  
 102705 all characters, *localedef* shall write a warning message to standard error and, if the exit status  
 102706 would otherwise have been zero, shall exit with status 1.

102707 **OPTIONS**

102708 The *localedef* utility shall conform to XBD [Section 12.2](#) (on page 215).

102709 The following options shall be supported:

102710 **-c** Create permanent output even if warning messages have been issued.

102711 **-f** *charmap* Specify the pathname of a file containing a mapping of character symbols and  
 102712 collating element symbols to actual character encodings. The format of the  
 102713 *charmap* is described in XBD [Section 6.4](#) (on page 121). The application shall ensure  
 102714 that this option is specified if symbolic names (other than collating symbols  
 102715 defined in a **collating-symbol** keyword) are used. If the -f option is not present, an  
 102716 implementation-defined character mapping shall be used.

102717        **-i** *inputfile*    The pathname of a file containing the source definitions. If this option is not  
 102718                            present, source definitions shall be read from standard input. The format of the  
 102719                            *inputfile* is described in XBD [Section 7.3](#) (on page 128).

102720        **-u** *code\_set\_name*   Specify the name of a codeset used as the target mapping of character symbols and  
 102721                            collating element symbols whose encoding values are defined in terms of the  
 102722                            ISO/IEC 10646: 2020 standard position constant values.  
 102723

#### 102724 OPERANDS

102725        The following operand shall be supported:

102726        *name*            Identifies the locale; see XBD [Chapter 7](#) (on page 127) for a description of the use of  
 102727                            this name. If the name contains one or more <slash> characters, *name* shall be  
 102728                            interpreted as a pathname where the created locale definitions shall be stored. If  
 102729                            *name* does not contain any <slash> characters, the interpretation of the name is  
 102730                            implementation-defined and the locale shall be public. The ability to create public  
 102731                            locales in this way may be restricted to users with appropriate privileges. (As a  
 102732                            consequence of specifying one *name*, although several categories can be processed  
 102733                            in one execution, only categories belonging to the same locale can be processed.)

#### 102734 STDIN

102735        Unless the **-i** option is specified, the standard input shall be a text file containing one or more  
 102736        locale category source definitions, as described in XBD [Section 7.3](#) (on page 128). When lines are  
 102737        continued using the escape character mechanism, there is no limit to the length of the  
 102738        accumulated continued line.

#### 102739 INPUT FILES

102740        The character set mapping file specified as the *charmap* option-argument is described in XBD  
 102741        [Section 6.4](#) (on page 121). If a locale category source definition contains a **copy** statement, as  
 102742        defined in XBD [Chapter 7](#) (on page 127), and the **copy** statement names a valid, existing locale,  
 102743        then *localedef* shall behave as if the source definition had contained a valid category source  
 102744        definition for the named locale.

#### 102745 ENVIRONMENT VARIABLES

102746        The following environment variables shall affect the execution of *localedef*:

102747        **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 102748                            (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 102749                            variables used to determine the values of locale categories.)

102750        **LC\_ALL**         If set to a non-empty string value, override the values of all the other  
 102751                            internationalization variables.

102752        **LC\_COLLATE**     (This variable has no affect on *localedef*; the POSIX locale is used for this category.)

102754        **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
 102755                            characters (for example, single-byte as opposed to multi-byte characters in  
 102756                            arguments and input files). This variable has no affect on the processing of *localedef*  
 102757                            input data; the POSIX locale is used for this purpose, regardless of the value of this  
 102758                            variable.

102759        **LC\_MESSAGES**

102760                            Determine the locale that should be used to affect the format and contents of  
 102761                            diagnostic messages written to standard error.

- 102762 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 102763 **ASYNCHRONOUS EVENTS**
- 102764 Default.
- 102765 **STDOUT**
- 102766 The utility shall report all categories successfully processed, in an unspecified format.
- 102767 **STDERR**
- 102768 The standard error shall be used only for diagnostic messages.
- 102769 **OUTPUT FILES**
- 102770 The format of the created output is unspecified. If the *name* operand does not contain a <slash>, the existence of an output file for the locale is unspecified.
- 102771
- 102772 **EXTENDED DESCRIPTION**
- 102773 When the `-u` option is used, the *code\_set\_name* option-argument shall be interpreted as an implementation-defined name of a codeset to which the ISO/IEC 10646:2020 standard position constant values shall be converted via an implementation-defined method. Both the ISO/IEC 10646:2020 standard position constant values and other formats (decimal, hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset represented by the implementation-defined name can be any codeset that is supported by the implementation.
- 102774
- 102775
- 102776
- 102777
- 102778
- 102779
- 102780 When conflicts occur between the *charmap* specification of <*code\_set\_name*>, <*mb\_cur\_max*>, or <*mb\_cur\_min*> and the implementation-defined interpretation of these respective items for the codeset represented by the `-u` option-argument *code\_set\_name*, the result is unspecified.
- 102781
- 102782
- 102783 When conflicts (including omissions) occur between the *charmap* encoding values specified for symbolic names of characters of the portable character set and the implementation-defined assignment of character encoding values, the result is unspecified. If the result is that *localedef* creates the specified locale, any attempted use of that locale by an application or utility results in undefined behavior.
- 102784
- 102785
- 102786
- 102787
- 102788 If a non-printable character in the *charmap* has a width specified that is not `-1`, the result is undefined.
- 102789
- 102790 **EXIT STATUS**
- 102791 The following exit values shall be returned:
- 102792 0 No errors occurred and the locales were successfully created.
- 102793 1 Warnings occurred and the locales were successfully created.
- 102794 2 The locale specification exceeded implementation limits or the coded character set or sets used were not supported by the implementation, and no locale was created.
- 102795
- 102796 3 The capability to create new locales is not supported by the implementation.
- 102797 >3 Warnings or errors occurred and no output was created.
- 102798 **CONSEQUENCES OF ERRORS**
- 102799 If an error is detected, no permanent output shall be created.
- 102800 If warnings occur, permanent output shall be created if the `-c` option was specified. The following conditions shall cause warning messages to be issued:
- 102801
- 102802 • If a symbolic name not found in the *charmap* file is used for the descriptions of the *LC\_CTYPE* or *LC\_COLLATE* categories (for other categories, this shall be an error condition).
  - 102803
  - 102804



- 102805 • If the number of operands to the **order** keyword exceeds the {COLL\_WEIGHTS\_MAX}
- 102806 limit.
- 102807 • If optional keywords not supported by the implementation are present in the source.
- 102808 Other implementation-defined conditions may also cause warnings.

#### 102809 APPLICATION USAGE

102810 The *charmap* definition is optional, and is contained outside the locale definition. This allows  
 102811 both completely self-defined source files, and generic sources (applicable to more than one  
 102812 codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the  
 102813 portable character set. As explained in XBD [Section 6.4](#) (on page 121), it is implementation-  
 102814 defined whether or not users or applications can provide additional character set description  
 102815 files. Therefore, the **-f** option might be operable only when an implementation-defined *charmap*  
 102816 is named.

#### 102817 EXAMPLES

102818 None.

#### 102819 RATIONALE

102820 The output produced by the *localedef* utility is implementation-defined. The *name* operand is  
 102821 used to identify the specific locale. (As a consequence, although several categories can be  
 102822 processed in one execution, only categories belonging to the same locale can be processed.)

102823 When conflicts (including omissions) occur between the *charmap* encoding values specified for  
 102824 symbolic names of characters of the portable character set and the implementation-defined  
 102825 assignment of character encoding values, it is recommended that *localedef* treats this as an error  
 102826 in order to prevent the undefined behavior that results if *localedef* creates the specified locale and  
 102827 an application or utility attempts to use it.

#### 102828 FUTURE DIRECTIONS

102829 If this utility is directed to create a new directory entry that contains any bytes that have the  
 102830 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 102831 error. A future version of this standard may require implementations to treat this as an error.

#### 102832 SEE ALSO

102833 [locale](#)

102834 XBD [Section 6.4](#) (on page 121), [Chapter 7](#) (on page 127), [Chapter 8](#) (on page 167), [Section 12.2](#) (on  
 102835 page 215)

#### 102836 CHANGE HISTORY

102837 First released in Issue 4.

#### 102838 Issue 6

102839 The **-u** option is added, as specified in the IEEE P1003.2b draft standard.

102840 The normative text is reworded to avoid use of the term “must” for application requirements.

102841 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/15 is applied, rewording text in the  
 102842 OPERANDS section describing the ability to create public locales.

102843 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/16 is applied, making the text consistent  
 102844 with the descriptions of **WIDTH** and **WIDTH\_DEFAULT** in the Base Definitions volume of  
 102845 POSIX.1-2024.

102846 **Issue 7**

102847 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

102848 **Issue 8**

102849 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
102850 filenames containing any bytes that have the encoded value of a <newline> character.

102851 Austin Group Defect 1070 is applied, requiring that *localedef* issues a warning if the  
102852 *LC\_COLLATE* category defines a collation sequence that does not have a total ordering of all  
102853 characters.

102854 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

102855 Austin Group Defect 1609 is applied, clarifying the behavior when conflicts (including  
102856 omissions) occur between the *charmap* encoding values specified for symbolic names of  
102857 characters of the portable character set and the implementation-defined assignment of character  
102858 encoding values.

102859 **NAME**

102860           logger — log messages

102861 **SYNOPSIS**102862           logger [-i] [-f *file*] [-p *priority*] [-t *tag*] [*string...*]102863 **DESCRIPTION**

102864           The *logger* utility shall send messages to an implementation-defined logging facility, which may  
 102865           log them in an implementation-defined system log, write them to the system console, forward  
 102866           them to a list of users, or forward them to the logging facility on another host over the network.  
 102867           Each logged message shall include a message header and a message body. The message header  
 102868           shall contain at least a timestamp and a tag string.

102869           If one or more *string* operands are specified, they shall be logged; otherwise, the message bodies  
 102870           to be logged shall be read from standard input if no **-f** option is specified, or from the specified  
 102871           file if the **-f** option is present.

102872           It is implementation-defined whether messages written in locales other than the POSIX locale  
 102873           are effective.

102874 **OPTIONS**102875           The *logger* utility shall conform to XBD [Section 12.2](#) (on page 215).

102876           The following options shall be supported:

102877           **-f** *file*       Read the log message bodies from *file* instead of standard input.102878           **-i**           Log the process ID of the *logger* process with each message.

102879           **-p** *priority*   Log the message with priority set to *priority*. The priority is specified as a  
 102880           *facility.level* pair. The following values for *facility* shall be supported:

102881           user        Messages generated by arbitrary processes.

102882           local0     Reserved for local use.

102883           local1     Reserved for local use.

102884           local2     Reserved for local use.

102885           local3     Reserved for local use.

102886           local4     Reserved for local use.

102887           local5     Reserved for local use.

102888           local6     Reserved for local use.

102889           local7     Reserved for local use.

102890           The following values for *level* shall be supported:

102891           emerg     A panic condition.

102892           alert     A condition that should be corrected immediately, such as a corrupted  
 102893           system database.

102894           crit       Critical conditions, such as hard device errors.

102895           err        Errors.

102896           warning   Warning messages.

|        |                     |                                                                                                    |
|--------|---------------------|----------------------------------------------------------------------------------------------------|
| 102897 | notice              | Conditions that are not error conditions, but that may require special handling.                   |
| 102898 |                     |                                                                                                    |
| 102899 | info                | Informational messages.                                                                            |
| 102900 | debug               | Messages that contain information normally of use only when debugging a program.                   |
| 102901 |                     |                                                                                                    |
| 102902 |                     | If the <code>-p</code> option is not specified, the priority shall be <code>user.notice</code> .   |
| 102903 | <code>-t tag</code> | Use the string <i>tag</i> as the tag string in the message header. The default tag is unspecified. |
| 102904 |                     |                                                                                                    |

#### 102905 OPERANDS

102906 The following operand shall be supported:

|        |               |                                                                                                                                       |
|--------|---------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 102907 | <i>string</i> | One of the string arguments whose contents are concatenated together, in the order specified, separated by single <space> characters. |
| 102908 |               |                                                                                                                                       |

#### 102909 STDIN

102910 The standard input shall be used if no *string* operands are specified and either the `-f` option is not specified or the `-f` option is specified with a *file* option-argument of `'-'` and the implementation treats the `'-'` as meaning standard input. Otherwise, the standard input shall not be used. See the INPUT FILES section.

102914 Each non-empty line shall be logged as a separate message. It is unspecified whether an empty line is also logged as a separate message.

#### 102916 INPUT FILES

102917 The input files shall be text files.

#### 102918 ENVIRONMENT VARIABLES

102919 The following environment variables shall affect the execution of *logger*:

|        |             |                                                                                                                                                                                                                                    |
|--------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 102920 | <i>LANG</i> | Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.) |
| 102921 |             |                                                                                                                                                                                                                                    |
| 102922 |             |                                                                                                                                                                                                                                    |

|        |               |                                                                                                          |
|--------|---------------|----------------------------------------------------------------------------------------------------------|
| 102923 | <i>LC_ALL</i> | If set to a non-empty string value, override the values of all the other internationalization variables. |
| 102924 |               |                                                                                                          |

|        |                 |                                                                                                                                                                           |
|--------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 102925 | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments). |
| 102926 |                 |                                                                                                                                                                           |
| 102927 |                 |                                                                                                                                                                           |

|        |                    |  |
|--------|--------------------|--|
| 102928 | <i>LC_MESSAGES</i> |  |
|--------|--------------------|--|

|        |  |                                                                                                                                                                                                                                                                                    |
|--------|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 102929 |  | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error. (This means diagnostics from <i>logger</i> to the user or application, not diagnostic messages that the user is sending to the system administrator.) |
| 102930 |  |                                                                                                                                                                                                                                                                                    |
| 102931 |  |                                                                                                                                                                                                                                                                                    |
| 102932 |  |                                                                                                                                                                                                                                                                                    |

|        |            |                |                                                                  |
|--------|------------|----------------|------------------------------------------------------------------|
| 102933 | <i>XSI</i> | <i>NLSPATH</i> | Determine the location of messages objects and message catalogs. |
|--------|------------|----------------|------------------------------------------------------------------|

#### 102934 ASYNCHRONOUS EVENTS

102935 Default.

#### 102936 STDOUT

102937 Not used.

**102938 STDERR**

102939 The standard error shall be used only for diagnostic messages.

**102940 OUTPUT FILES**

102941 Unspecified.

**102942 EXTENDED DESCRIPTION**

102943 None.

**102944 EXIT STATUS**

102945 The following exit values shall be returned:

102946 0 Successful completion.

102947 >0 An error occurred.

**102948 CONSEQUENCES OF ERRORS**

102949 Default.

**102950 APPLICATION USAGE**

102951 This utility allows logging of information for later use by a system administrator or programmer  
102952 in determining why non-interactive utilities have failed. The locations of the saved messages,  
102953 their format, and retention period are all unspecified. There is no method for a conforming  
102954 application to read messages, once written.

**102955 EXAMPLES**

102956 A batch application, running non-interactively, tries to read a configuration file and fails; it may  
102957 attempt to notify the system administrator with:

102958 `logger myname: unable to read file foo. [timestamp]`

**102959 RATIONALE**

102960 The standard developers believed strongly that some method of alerting administrators to errors  
102961 was necessary. The obvious example is a batch utility, running non-interactively, that is unable to  
102962 read its configuration files or that is unable to create or write its results file. However, the  
102963 standard developers did not wish to define the format or delivery mechanisms as they have  
102964 historically been (and will probably continue to be) very system-specific, as well as involving  
102965 functionality clearly outside the scope of this volume of POSIX.1-2024.

102966 The text with *LC\_MESSAGES* about diagnostic messages means diagnostics from *logger* to the  
102967 user or application, not diagnostic messages that the user is sending to the system administrator.

102968 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

102969 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient  
102970 justification to exclude these utilities from this volume of POSIX.1-2024. It is also arguable that  
102971 they are, in fact, testable, but that the tests themselves are not portable.

**102972 FUTURE DIRECTIONS**

102973 None.

**102974 SEE ALSO**

102975 *lp*, *mailx*, *write*

102976 XBD Chapter 8 (on page 167)

**102977 CHANGE HISTORY**

102978 First released in Issue 4.

102979 **Issue 7**

102980 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

102981 **Issue 8**

102982 Austin Group Defect 917 is applied, adding the **-f**, **-i**, **-p**, and **-t** options, and specifying the  
102983 behavior when *logger* is executed with no operands.

102984 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

102985 **NAME**

102986 logname — return the user's login name

102987 **SYNOPSIS**

102988 logname

102989 **DESCRIPTION**

102990 The *logname* utility shall write the user's login name to standard output. The login name shall be  
 102991 the string that would be returned by the *getlogin()* function defined in the System Interfaces  
 102992 volume of POSIX.1-2024. Under the conditions where the *getlogin()* function would fail, the  
 102993 *logname* utility shall write a diagnostic message to standard error and exit with a non-zero exit  
 102994 status.

102995 **OPTIONS**

102996 None.

102997 **OPERANDS**

102998 None.

102999 **STDIN**

103000 Not used.

103001 **INPUT FILES**

103002 None.

103003 **ENVIRONMENT VARIABLES**103004 The following environment variables shall affect the execution of *logname*:

103005 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 103006 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 103007 variables used to determine the values of locale categories.)

103008 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 103009 internationalization variables.

103010 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 103011 characters (for example, single-byte as opposed to multi-byte characters in  
 103012 arguments).

103013 *LC\_MESSAGES*

103014 Determine the locale that should be used to affect the format and contents of  
 103015 diagnostic messages written to standard error.

103016 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

103017 **ASYNCHRONOUS EVENTS**

103018 Default.

103019 **STDOUT**103020 The *logname* utility output shall be a single line consisting of the user's login name:

103021 "%s\n", &lt;login name&gt;

103022 **STDERR**

103023 The standard error shall be used only for diagnostic messages.

103024 **OUTPUT FILES**

103025 None.

103026 **EXTENDED DESCRIPTION**

103027 None.

103028 **EXIT STATUS**

103029 The following exit values shall be returned:

103030 0 Successful completion.

103031 &gt;0 An error occurred.

103032 **CONSEQUENCES OF ERRORS**

103033 Default.

103034 **APPLICATION USAGE**103035 The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment  
103036 changes could produce erroneous results.103037 **EXAMPLES**

103038 None.

103039 **RATIONALE**103040 The **passwd** file is not listed as required because the implementation may have other means of  
103041 mapping login names.103042 **FUTURE DIRECTIONS**

103043 None.

103044 **SEE ALSO**103045 *id*, *who*

103046 XBD Chapter 8 (on page 167)

103047 XSH *getlogin()*103048 **CHANGE HISTORY**

103049 First released in Issue 2.

103050 **Issue 8**103051 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.



103052 **NAME**

103053 lp — send files to a printer

103054 **SYNOPSIS**103055 lp [-c] [-d *dest*] [-n *copies*] [-msw] [-o *option*]... [-t *title*] [*file*...]103056 **DESCRIPTION**

103057 The *lp* utility shall copy the input files to an output destination in an unspecified manner. The  
 103058 default output destination should be to a hardcopy device, such as a printer or microfilm  
 103059 recorder, that produces non-volatile, human-readable documents. If such a device is not  
 103060 available to the application, or if the system provides no such device, the *lp* utility shall exit with  
 103061 a non-zero exit status.

103062 The actual writing to the output device may occur some time after the *lp* utility successfully  
 103063 exits. During the portion of the writing that corresponds to each input file, the implementation  
 103064 shall guarantee exclusive access to the device.

103065 The *lp* utility shall associate a unique *request ID* with each request.

103066 Normally, a banner page is produced to separate and identify each print job. This page may be  
 103067 suppressed by implementation-defined conditions, such as an operator command or one of the  
 103068 **-o** *option* values.

103069 **OPTIONS**

103070 The *lp* utility shall conform to XBD [Section 12.2](#) (on page 215).

103071 The following options shall be supported:

103072 **-c** Exit only after further access to any of the input files is no longer required. The  
 103073 application can then safely delete or modify the files without affecting the output  
 103074 operation. Normally, files are not copied, but are linked whenever possible. If the  
 103075 **-c** option is not given, then the user should be careful not to remove any of the  
 103076 files before the request has been printed in its entirety. It should also be noted that  
 103077 in the absence of the **-c** option, any changes made to the named files after the  
 103078 request is made but before it is printed may be reflected in the printed output. On  
 103079 some implementations, **-c** may be on by default.

103080 **-d** *dest* Specify a string that names the destination (*dest*). If *dest* is a printer, the request  
 103081 shall be printed only on that specific printer. If *dest* is a class of printers, the request  
 103082 shall be printed on the first available printer that is a member of the class. Under  
 103083 certain conditions (printer unavailability, file space limitation, and so on), requests  
 103084 for specific destinations need not be accepted. Destination names vary between  
 103085 systems.

103086 If **-d** is not specified, and neither the *LPDEST* nor *PRINTER* environment variable  
 103087 is set, an unspecified destination is used. The **-d** *dest* option shall take precedence  
 103088 over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are  
 103089 undefined when *dest* contains a value that is not a valid destination name.

103090 **-m** Send mail (see [mailx](#)) after the files have been printed. By default, no mail is sent  
 103091 upon normal completion of the print request.

103092 **-n** *copies* Write *copies* number of copies of the files, where *copies* is a positive decimal integer.  
 103093 The methods for producing multiple copies and for arranging the multiple copies  
 103094 when multiple *file* operands are used are unspecified, except that each file shall be  
 103095 output as an integral whole, not interleaved with portions of other files.

- 103096      **-o option**      Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **-o** option more than once.
- 103097
- 103098      **-s**                 Suppress messages from *lp*.
- 103099      **-t title**         Write *title* on the banner page of the output.
- 103100      **-w**                 Write a message on the user's terminal after the files have been printed. If the user is not logged in, then mail shall be sent instead.
- 103101

#### 103102 OPERANDS

103103      The following operand shall be supported:

- 103104      *file*             A pathname of a file to be output. If no *file* operands are specified, or if a *file* operand is '-', the standard input shall be used. If a *file* operand is used, but the **-c** option is not specified, the process performing the writing to the output device may have user and group permissions that differ from that of the process invoking *lp*.
- 103105
- 103106
- 103107
- 103108

#### 103109 STDIN

103110      The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'. See the INPUT FILES section.

103111

#### 103112 INPUT FILES

103113      The input files shall be text files.

#### 103114 ENVIRONMENT VARIABLES

103115      The following environment variables shall affect the execution of *lp*:

- 103116      **LANG**            Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)
- 103117
- 103118

- 103119      **LC\_ALL**          If set to a non-empty string value, override the values of all the other internationalization variables.
- 103120

- 103121      **LC\_CTYPE**        Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
- 103122
- 103123

#### 103124 LC\_MESSAGES

103125      Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

103126

103127

- 103128      **LC\_TIME**         Determine the format and contents of date and time strings displayed in the *lp* banner page, if any.
- 103129

- 103130      **LPDEST**          Determine the destination. If the *LPDEST* environment variable is not set, the *PRINTER* environment variable shall be used. The **-d dest** option takes precedence over *LPDEST*. Results are undefined when **-d** is not specified and *LPDEST* contains a value that is not a valid destination name.
- 103131
- 103132
- 103133

- 103134 XSI      **NLSPATH**         Determine the location of messages objects and message catalogs.

- 103135      **PRINTER**         Determine the output device or destination. If the *LPDEST* and *PRINTER* environment variables are not set, an unspecified output device is used. The **-d dest** option and the *LPDEST* environment variable shall take precedence over *PRINTER*. Results are undefined when **-d** is not specified, *LPDEST* is unset, and *PRINTER* contains a value that is not a valid device or destination name.
- 103136
- 103137
- 103138
- 103139

103140 *TZ* Determine the timezone used to calculate date and time strings displayed in the *lp*  
 103141 banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be  
 103142 used.

### 103143 ASYNCHRONOUS EVENTS

103144 Default.

### 103145 STDOUT

103146 The *lp* utility shall write a *request ID* to the standard output, unless *-s* is specified. The format of  
 103147 the message is unspecified. The request ID can be used on systems supporting the historical  
 103148 *cancel* and *lpstat* utilities.

### 103149 STDERR

103150 The standard error shall be used only for diagnostic messages.

### 103151 OUTPUT FILES

103152 None.

### 103153 EXTENDED DESCRIPTION

103154 None.

### 103155 EXIT STATUS

103156 The following exit values shall be returned:

103157 0 All input files were processed successfully.

103158 >0 No output device was available, or an error occurred.

### 103159 CONSEQUENCES OF ERRORS

103160 Default.

### 103161 APPLICATION USAGE

103162 The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's  
 103163 default page size.

103164 A conforming application can use one of the *file* operands only with the *-c* option or if the file is  
 103165 publicly readable and guaranteed to be available at the time of printing. This is because  
 103166 POSIX.1-2024 gives the implementation the freedom to queue up the request for printing at  
 103167 some later time by a different process that might not be able to access the file.

### 103168 EXAMPLES

103169 1. To print file *file*:

103170 `lp -c file`

103171 2. To print multiple files with headers:

103172 `pr file1 file2 | lp`

### 103173 RATIONALE

103174 The *lp* utility was designed to be a basic version of a utility that is already available in many  
 103175 historical implementations. The standard developers considered that it should be implementable  
 103176 simply as:

103177 `cat "$@" > /dev/lp`

103178 after appropriate processing of options, if that is how the implementation chose to do it and if  
 103179 exclusive access could be granted (so that two users did not write to the device simultaneously).  
 103180 Although in the future the standard developers may add other options to this utility, it should  
 103181 always be able to execute with no options or operands and send the standard input to an

103182 unspecified output device.

103183 This volume of POSIX.1-2024 makes no representations concerning the format of the printed  
103184 output, except that it must be “human-readable” and “non-volatile”. Thus, writing by default to  
103185 a disk or tape drive or a display terminal would not qualify. (Such destinations are not  
103186 prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)

103187 This volume of POSIX.1-2024 is worded such that a “print job” consisting of multiple input files,  
103188 possibly in multiple copies, is guaranteed to print so that any one file is not intermixed with  
103189 another, but there is no statement that all the files or copies have to print out together.

103190 The `-c` option may imply a spooling operation, but this is not required. The utility can be  
103191 implemented to wait until the printer is ready and then wait until it is finished. Because of that,  
103192 there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

103193 On some historical systems, the request ID reported on the `STDOUT` can be used to later cancel  
103194 or find the status of a request using utilities not defined in this volume of POSIX.1-2024.

103195 Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality,  
103196 they used different names for the environment variable specifying the destination printer. Since  
103197 the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence  
103198 over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one  
103199 or the other environment variable, the `lp` utility is required to recognize both. If this was not  
103200 done, many applications would send output to unexpected output devices when users moved  
103201 from system to system.

103202 Some have commented that `lp` has far too little functionality to make it worthwhile. Requests  
103203 have proposed additional options or operands or both that added functionality. The requests  
103204 included:

- 103205 • Wording *requiring* the output to be “hardcopy”
- 103206 • A requirement for multiple printers
- 103207 • Options for supporting various page-description languages

103208 Given that a compliant system is not required to even have a printer, placing further restrictions  
103209 upon the behavior of the printer is not useful. Since hardcopy format is so application-  
103210 dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that  
103211 should be required on all compliant systems.

103212 The term *unspecified* is used in this section in lieu of *implementation-defined* as most known  
103213 implementations would not be able to make definitive statements in their conformance  
103214 documents; the existence and usage of printers is very dependent on how the system  
103215 administrator configures each individual system.

103216 Since the default destination, device type, queuing mechanisms, and acceptable forms of input  
103217 are all unspecified, usage guidelines for what a conforming application can do are as follows:

- 103218 • Use the command in a pipeline, or with `-c`, so that there are no permission problems and  
103219 the files can be safely deleted or modified.
- 103220 • Limit output to text files of reasonable line lengths and printable characters and include no  
103221 device-specific formatting information, such as a page description language. The meaning  
103222 of “reasonable” in this context can only be answered as a quality-of-implementation issue,  
103223 but it should be apparent from historical usage patterns in the industry and the locale. The  
103224 `pr` and `fold` utilities can be used to achieve reasonable formatting for the default page size  
103225 of the implementation.

103226 Alternatively, the application can arrange its installation in such a way that it requires the system  
 103227 administrator or operator to provide the appropriate information on *lp* options and environment  
 103228 variable values.

103229 At a minimum, having this utility in this volume of POSIX.1-2024 tells the industry that  
 103230 conforming applications require a means to print output and provides at least a command name  
 103231 and *LPDEST* routing mechanism that can be used for discussions between vendors, application  
 103232 developers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent  
 103233 of the standard developers, even if they cannot mandate that all systems (such as laptops) have  
 103234 printers.

103235 This volume of POSIX.1-2024 does not specify what the ownership of the process performing the  
 103236 writing to the output device may be. If *-c* is not used, it is unspecified whether the process  
 103237 performing the writing to the output device has permission to read *file* if there are any  
 103238 restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the  
 103239 results of deleting *file* before it is printed are unspecified.

#### 103240 FUTURE DIRECTIONS

103241 None.

#### 103242 SEE ALSO

103243 *mailx*

103244 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

#### 103245 CHANGE HISTORY

103246 First released in Issue 2.

#### 103247 Issue 6

103248 The following new requirements on POSIX implementations derive from alignment with the  
 103249 Single UNIX Specification:

- 103250 • In the DESCRIPTION, the requirement to associate a unique request ID, and the normal  
 103251 generation of a banner page is added.
- 103252 • In the OPTIONS section:
  - 103253 — The *-d dest* description is expanded, but references to *lpstat* are removed.
  - 103254 — The *-m*, *-o*, *-s*, *-t*, and *-w* options are added.
- 103255 • In the ENVIRONMENT VARIABLES section, *LC\_TIME* may now affect the execution.
- 103256 • The STDOUT section is added.

103257 The normative text is reworded to avoid use of the term “must” for application requirements.

103258 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

#### 103259 Issue 7

103260 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

#### 103261 Issue 8

103262 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

103263 **NAME**

103264 ls — list directory contents

103265 **SYNOPSIS**

```
103266 XSI  ls [-ikqrs] [-glnO] [-A|-a] [-C|-m|-x|-l] \
103267      [-F|-p] [-H|-L] [-R|-d] [-S|-f|-t] [-c|-u] [file...]
```

103268 **DESCRIPTION**

103269 For each operand that names a file of a type other than directory or symbolic link to a directory,  
 103270 *ls* shall write the name of the file as well as any requested, associated information. For each  
 103271 operand that names a file of type directory, *ls* shall write the names of files contained within the  
 103272 directory as well as any requested, associated information. Filenames beginning with a <period>  
 103273 ('.') and any associated information shall not be written out unless explicitly referenced, the  
 103274 **-A** or **-a** option is supplied, or an implementation-defined condition causes them to be written.  
 103275 If one or more of the **-d**, **-F**, or **-l** options are specified, and neither the **-H** nor the **-L** option is  
 103276 specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the  
 103277 name of the file as well as any requested, associated information. If none of the **-d**, **-F**, or **-l**  
 103278 options are specified, or the **-H** or **-L** options are specified, for each operand that names a file of  
 103279 type symbolic link to a directory, *ls* shall write the names of files contained within the directory  
 103280 as well as any requested, associated information. In each case where the names of files contained  
 103281 within a directory are written, if the directory contains any symbolic links then *ls* shall evaluate  
 103282 the file information and file type to be those of the symbolic link itself, unless the **-L** option is  
 103283 specified.

103284 If no operands are specified, *ls* shall behave as if a single operand of dot ('.') had been  
 103285 specified. If more than one operand is specified, *ls* shall write non-directory operands first; it  
 103286 shall sort directory and non-directory operands separately according to the collating sequence in  
 103287 the current locale.

103288 Whenever *ls* sorts filenames or pathnames according to the collating sequence in the current  
 103289 locale, if this collating sequence does not have a total ordering of all characters (see XBD [Section](#)  
 103290 [7.3.2](#), on page 139), then any filenames or pathnames that collate equally shall be further  
 103291 compared byte-by-byte using the collating sequence for the POSIX locale.

103292 The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an  
 103293 ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic  
 103294 message to standard error and shall either recover its position in the hierarchy or terminate.

103295 **OPTIONS**

103296 The *ls* utility shall conform to XBD [Section 12.2](#) (on page 215).

103297 The following options shall be supported:

103298 **-A** Write out all directory entries, including those whose names begin with a <period>  
 103299 ('.') but excluding the entries dot and dot-dot (if they exist).

103300 **-C** Write multi-text-column output with entries sorted down the columns, according  
 103301 to the collating sequence. The number of text columns and the column separator  
 103302 characters are unspecified, but should be adapted to the nature of the output  
 103303 device. This option disables long format output.

103304 **Note:** Since the output from this option may use separator characters that include  
 103305 characters that might appear in filenames (in addition to the problems related to  
 103306 <newline>s in filenames), **-C** should not be used when filenames might be  
 103307 extracted from the output by a script.

|            |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 103308     | <b>-F</b> | Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Write a <code>&lt;slash&gt;</code> ( <code>'/'</code> ) immediately after each pathname that is a directory, an <code>&lt;asterisk&gt;</code> ( <code>'*'</code> ) after each that is executable, a <code>&lt;vertical-line&gt;</code> ( <code>' '</code> ) after each that is a FIFO, an <code>&lt;equals-sign&gt;</code> ( <code>'='</code> ) after each that is a socket, and a <code>&lt;commercial-at&gt;</code> ( <code>'@'</code> ) after each that is a symbolic link. For other file types, other symbols may be written. |
| 103309     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103310     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103311     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103312     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103313     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103314     | <b>-H</b> | Evaluate the file information and file type for symbolic links specified on the command line to be those of the file referenced by the link, and not the link itself; however, <i>ls</i> shall write the name of the link itself and not the file referenced by the link.                                                                                                                                                                                                                                                                                                                                                                  |
| 103315     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103316     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103317     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103318     | <b>-L</b> | Evaluate the file information and file type for all symbolic links (whether named on the command line or encountered in a file hierarchy) to be those of the file referenced by the link, and not the link itself; however, <i>ls</i> shall write the name of the link itself and not the file referenced by the link. When <b>-L</b> is used with <b>-l</b> , write the contents of symbolic links in the long format (see the STDOUT section).                                                                                                                                                                                           |
| 103319     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103320     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103321     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103322     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103323     | <b>-R</b> | Recursively list subdirectories encountered. Subdirectories with filenames beginning with a <code>&lt;period&gt;</code> ( <code>'.'</code> ) shall be recursively listed if and only if the subdirectory name was included in the filenames listed for the containing directory. When a symbolic link to a directory is encountered, the directory shall not be recursively listed unless the <b>-L</b> option is specified. The use of <b>-R</b> with <b>-d</b> or <b>-f</b> produces unspecified results.                                                                                                                                |
| 103324     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103325     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103326     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103327     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103328     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103329     | <b>-S</b> | Sort with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order). For a symbolic link, the size used as the sort key is that of the symbolic link itself, unless <i>ls</i> is evaluating its file information to be that of the file referenced by the link (see the <b>-H</b> and <b>-L</b> options).                                                                                                                                                                                                                                                 |
| 103330     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103331     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103332     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103333     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103334     | <b>-a</b> | Write out all directory entries, including those whose names begin with a <code>&lt;period&gt;</code> ( <code>'.'</code> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 103335     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103336     | <b>-c</b> | Use time of last modification of the file status information (see XBD <code>&lt;sys/stat.h&gt;</code> ) instead of last modification of the file itself for sorting ( <b>-t</b> ) or writing ( <b>-l</b> ).                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 103337     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103338     | <b>-d</b> | Do not follow symbolic links named as operands unless the <b>-H</b> or <b>-L</b> options are specified. Do not treat directories differently than other types of files. The use of <b>-d</b> with <b>-R</b> or <b>-f</b> produces unspecified results.                                                                                                                                                                                                                                                                                                                                                                                     |
| 103339     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103340     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103341     | <b>-f</b> | List the entries in directory operands in the order they appear in the directory. The behavior for non-directory operands is unspecified. This option shall turn on <b>-a</b> . When <b>-f</b> is specified, any occurrences of the <b>-r</b> , <b>-S</b> , and <b>-t</b> options shall be ignored and any occurrences of the <b>-A</b> , <b>-g</b> , <b>-l</b> , <b>-n</b> , <b>-o</b> , and <b>-s</b> options may be ignored. The use of <b>-f</b> with <b>-R</b> or <b>-d</b> produces unspecified results.                                                                                                                             |
| 103342     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103343     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103344 XSI | <b>-g</b> | Turn on the <b>-l</b> (ell) option, but disable writing the file's owner name or number. Disable the <b>-C</b> , <b>-m</b> , and <b>-x</b> options.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 103345     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103346 XSI |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103347     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103348     | <b>-i</b> | For each file, write the file's file serial number (see <i>stat()</i> in the System Interfaces volume of POSIX.1-2024).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 103349     |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 103350     | <b>-k</b> | Set the block size for the <b>-s</b> option and the per-directory block count written for the <b>-l</b> , <b>-n</b> , <b>-s</b> , <b>-g</b> , and <b>-o</b> options (see the STDOUT section) to 1 024 bytes.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 103351 XSI |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

- 103352        **-l**            (The letter ell.) Do not follow symbolic links named as operands unless the **-H** or  
 103353        **-L** options are specified. Write out in long format (see the STDOUT section).  
 103354        Disable the **-C**, **-m**, and **-x** options.
- 103355        **-m**            Stream output format; list pathnames across the page, separated by a <comma>  
 103356        character followed by a <space> character. Use a <newline> character as the list  
 103357        terminator and after the separator sequence when there is not room on a line for  
 103358        the next list entry. This option disables long format output.
- 103359        **-n**            Turn on the **-l** (ell) option, but when writing the file's owner or group, write the  
 103360        file's numeric UID or GID rather than the user or group name, respectively. Disable  
 103361        the **-C**, **-m**, and **-x** options.
- 103362 XSI        **-o**            Turn on the **-l** (ell) option, but disable writing the file's group name or number.  
 103363        Disable the **-C**, **-m**, and **-x** options.
- 103364        **-p**            Write a <slash> ( '/' ) after each pathname if that file is a directory.
- 103365        **-q**            Force each instance of non-printable filename characters (including <newline>,  
 103366        <tab>, and other control characters) to be written as the <question-mark> ( '?' )  
 103367        character. Implementations may provide this option by default if the output is to a  
 103368        terminal device.
- 103369        **-r**            Reverse the order of the sort to get reverse collating sequence, oldest first, or  
 103370        smallest file size first depending on the other options given.
- 103371        **-s**            Indicate the total number of file system blocks consumed by each file displayed. If  
 103372        the **-k** option is also specified, the block size shall be 1024 bytes; otherwise, the  
 103373        block size is implementation-defined.
- 103374        **-t**            Sort with the primary key being time modified (most recently modified first) and  
 103375        the secondary key being filename in the collating sequence. For a symbolic link,  
 103376        the time used as the sort key is that of the symbolic link itself, unless *ls* is  
 103377        evaluating its file information to be that of the file referenced by the link (see the  
 103378        **-H** and **-L** options).
- 103379        **-u**            Use time of last access (see XBD <sys/stat.h>) instead of last modification of the file  
 103380        for sorting (**-t**) or writing (**-l**).
- 103381        **-x**            The same as **-C**, except that the multi-text-column output is produced with entries  
 103382        sorted across, rather than down, the columns. This option disables long format  
 103383        output.
- 103384        **-1**            (The numeric digit one.) Force output to be one entry per line. This option does  
 103385 XSI        not disable long format output. (Long format output is enabled by **-g**, **-l** (ell), **-n**,  
 103386 XSI        and **-o**; and disabled by **-C**, **-m**, and **-x**.)
- 103387 XSI        If an option that enables long format output (**-g**, **-l** (ell), **-n**, and **-o**) is given with an option that  
 103388        disables long format output (**-C**, **-m**, and **-x**), this shall not be considered an error. The last of  
 103389        these options specified shall determine whether long format output is written.
- 103390        If **-R**, **-d**, or **-f** are specified, the results of specifying these mutually-exclusive options are  
 103391        specified by the descriptions of these options above. If more than one of any of the other options  
 103392        shown in the SYNOPSIS section in mutually-exclusive sets are given, this shall not be considered  
 103393        an error; the last option specified in each set shall determine the output.
- 103394        Note that if **-t** is specified, **-c** and **-u** are not only mutually-exclusive with each other, they are  
 103395        also mutually-exclusive with **-S** when determining sort order. But even if **-S** is specified after all  
 103396        occurrences of **-c**, **-t**, and **-u**, the last use of **-c** or **-u** determines the timestamp printed when



103397 producing long format output.

103398 **OPERANDS**

103399 The following operand shall be supported:

103400 *file* A pathname of a file to be written. If the file specified is not found, a diagnostic  
103401 message shall be output on standard error.

103402 **STDIN**

103403 Not used.

103404 **INPUT FILES**

103405 None.

103406 **ENVIRONMENT VARIABLES**

103407 The following environment variables shall affect the execution of *ls*:

103408 *COLUMNS* Override the system-selected horizontal display line size, used to determine the  
103409 column position width for writing multiple text-column output. See XBD [Chapter](#)  
103410 [8](#) (on page 167) for valid values and results when it is unset or null. The *ls* utility  
103411 shall use this value to calculate how many pathname text columns to write (see  
103412 [-C](#)). The column width chosen to write the names of files in any given directory  
103413 shall be constant. Filenames shall not be truncated to fit into the multiple text-  
103414 column output.

103415 *LANG* Provide a default value for the internationalization variables that are unset or null.  
103416 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
103417 variables used to determine the values of locale categories.)

103418 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
103419 internationalization variables.

103420 *LC\_COLLATE*

103421 Determine the locale for character collation information in determining the  
103422 pathname collation sequence.

103423 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
103424 characters (for example, single-byte as opposed to multi-byte characters in  
103425 arguments) and which characters are defined as printable (character class **print**).

103426 *LC\_MESSAGES*

103427 Determine the locale that should be used to affect the format and contents of  
103428 diagnostic messages written to standard error.

103429 *LC\_TIME* Determine the format and contents for date and time strings written by *ls*.

103430 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

103431 *TZ* Determine the timezone for date and time strings written by *ls*. If *TZ* is unset or  
103432 null, an unspecified default timezone shall be used.

103433 **ASYNCHRONOUS EVENTS**

103434 Default.

103435 **STDOUT**

103436 The default format shall be to list one entry per line to standard output; the exceptions are to  
103437 terminals or when one of the [-C](#), [-m](#), or [-x](#) options is specified. If the output is to a terminal, the  
103438 format is implementation-defined.

103439 In the formats specified below, except where specified otherwise the *<pathname>* field shall

103440 consist of the file's pathname and, if the **-F** or **-p** option is specified, any indicator character  
103441 written after the pathname as described for those options.

103442 When **-m** is specified, the format used for the last element of the list shall be:

103443 "%s\n", <pathname>

103444 The format used for each other element of the list shall be:

103445 "%s,%s", <pathname>, <separator>

103446 where, if there is not room for the next element of the list to fit within the current line length,  
103447 <separator> is a string containing an optional <space> character and a mandatory <newline>  
103448 character; otherwise it is a single <space> character.

103449 If the **-i** option is specified, the file's file serial number (see XBD <sys/stat.h>) shall be written in  
103450 the following format before any other output for the corresponding entry:

103451 %u ", <file serial number>

103452 If the **-l** option is specified, the following information shall be written for files other than  
103453 character special and block special files:

103454 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,  
103455 <owner name>, <group name>, <size>, <date and time>,  
103456 <pathname>

103457 If the **-l** option is specified, the following information shall be written for character special and  
103458 block special files:

103459 "%s %u %s %s %s %s %s\n", <file mode>, <number of links>,  
103460 <owner name>, <group name>, <device info>, <date and time>,  
103461 <pathname>

103462 In both cases if the file is a symbolic link and the **-L** option is also specified, this information  
103463 shall be for the file resolved from the symbolic link, except that the <pathname> field shall  
103464 contain the pathname of the symbolic link itself. If the file is a symbolic link and the **-L** option is  
103465 not specified, this information shall be about the link itself and the <pathname> field shall be one  
103466 of the following forms:

- 103467 • "%s%sΔ->Δ%s", <pathname of link>, <link type indicator>,  
103468 <contents of link>
- 103469 • "%sΔ->Δ%s%s", <pathname of link>, <contents of link>,  
103470 <file type indicator>
- 103471 • "%s%sΔ->Δ%s%s", <pathname of link>, <link type indicator>,  
103472 <contents of link>, <file type indicator>

103473 where <link type indicator> is a <commercial-at> ('@') if the **-F** option is specified, or an empty  
103474 string otherwise and <file type indicator> is the required indicator character, if any, for the file  
103475 resolved from the symbolic link if the **-F** or **-p** option is specified, or an empty string otherwise.  
103476 If pathname resolution fails when following the symbolic link, this shall not be treated as an  
103477 error and the <file type indicator> field shall be an empty string.

103478 XSI The **-n**, **-g**, and **-o** options use the same format as **-l**, but with omitted items and their  
103479 associated <blank> characters. See the OPTIONS section.

103480 In both the preceding **-l** forms, if <owner name> or <group name> cannot be determined, or if **-n**  
103481 is given, they shall be replaced with their associated numeric values using the format %u.

103482 The *<size>* field shall contain the value that would be returned for the file in the *st\_size* field of  
 103483 **struct stat** (see XBD [<sys/stat.h>](#)). Note that for some file types this value is unspecified.

103484 The *<device info>* field shall contain implementation-defined information associated with the  
 103485 device in question.

103486 The *<date and time>* field shall contain the appropriate date and timestamp of when the file was  
 103487 last modified. In the POSIX locale, the field shall be the equivalent of the output of the following  
 103488 *date* command:

103489 `date "+%b %e %H:%M"`

103490 if the file has been modified in the last six months, or:

103491 `date "+%b %e %Y"`

103492 (where two *<space>* characters are used between *%e* and *%Y*) if the file has not been modified in  
 103493 the last six months or if the modification date is in the future, except that, in both cases, the final  
 103494 *<newline>* produced by *date* shall not be included and the output shall be as if the *date*  
 103495 command were executed at the time of the last modification date of the file rather than the  
 103496 current time. When the *LC\_TIME* locale category is not set to the POSIX locale, a different format  
 103497 and order of presentation of this field may be used.

103498 If the pathname was specified as a *file* operand, it shall be written as specified.

103499 XSI The file mode written under the *-l*, *-n*, *-g*, and *-o* options shall consist of the following format:

103500 `"%c%s%s%s%s", <entry type>, <owner permissions>,  
 103501 <group permissions>, <other permissions>,  
 103502 <optional alternate access method flag>`

103503 The *<optional alternate access method flag>* shall be the empty string if there is no alternate or  
 103504 additional access control method associated with the file; otherwise, it shall be a string  
 103505 containing a single printable character that is not a *<blank>*.

103506 The *<entry type>* character shall describe the type of file, as follows:

103507 d Directory.  
 103508 b Block special file.  
 103509 c Character special file.  
 103510 l (ell) Symbolic link.  
 103511 p FIFO.  
 103512 s Socket.  
 103513 – Regular file.

103514 Implementations may add other characters to this list to represent other implementation-defined  
 103515 file types.

103516 The next three fields shall be three characters each:

103517 *<owner permissions>*

103518 Permissions for the file owner class (see XBD [Section 4.7](#), on page 97).

103519 *<group permissions>*

103520 Permissions for the file group class.

- 103521 <other permissions>  
 103522 Permissions for the file other class.
- 103523 Each field shall have three character positions:
- 103524 1. If 'r', the file is readable; if '-', the file is not readable.  
 103525 2. If 'w', the file is writable; if '-', the file is not writable.  
 103526 3. The first of the following that applies:
- 103527 S If in <owner permissions>, the file is not executable and set-user-ID mode is set. If in  
 103528 <group permissions>, the file is not executable and set-group-ID mode is set.
- 103529 s If in <owner permissions>, the file is executable and set-user-ID mode is set. If in  
 103530 <group permissions>, the file is executable and set-group-ID mode is set.
- 103531 XSI T If in <other permissions> and the file is a directory, search permission is not granted to  
 103532 others, and the restricted deletion flag is set.
- 103533 XSI t If in <other permissions> and the file is a directory, search permission is granted to  
 103534 others, and the restricted deletion flag is set.
- 103535 x The file is executable or the directory is searchable.
- 103536 - None of the attributes of 'S', 's', 'T', 't', or 'x' applies.
- 103537 Implementations may add other characters to this list for the third character position.  
 103538 Such additions shall, however, be written in lowercase if the file is executable or  
 103539 searchable, and in uppercase if it is not.
- 103540 XSI If any of the **-l**, **-n**, **-s**, **-g**, or **-o** options is specified, each list of files within a directory shall be  
 103541 preceded by a status line indicating the number of file system blocks occupied by the listed files  
 103542 for that directory in 512-byte units if the **-k** option is not specified, or 1 024-byte units if the **-k**  
 103543 option is specified, rounded up to the next integral number of units, if necessary. In the POSIX  
 103544 locale, the format shall be:
- 103545 "total %u\n", <number of units in the directory>
- 103546 If more than one directory, or a combination of non-directory files and directories are written,  
 103547 either as a result of specifying multiple operands, or the **-R** option, each list of files within a  
 103548 directory shall be preceded by:
- 103549 "\n%s:\n", <directory name>
- 103550 The above string may be omitted for the directory named by the operand if only one operand is  
 103551 present. It may also be omitted for dot ( '.' ) if no operands are present. If this string is the first  
 103552 thing to be written, the first <newline> shall not be written. This output shall precede the  
 103553 number of units in the directory.
- 103554 If the **-s** option is given, each file shall be written with the number of blocks used by the file.  
 103555 XSI Along with **-C**, **-l**, **-m**, or **-x**, the number and a <space> shall precede the filename; with **-l**, **-n**,  
 103556 **-g**, or **-o**, they shall precede each line describing a file.
- 103557 **STDERR**  
 103558 The standard error shall be used only for diagnostic messages.
- 103559 **OUTPUT FILES**  
 103560 None.

103561 **EXTENDED DESCRIPTION**

103562 None.

103563 **EXIT STATUS**

103564 The following exit values shall be returned:

103565 0 Successful completion.

103566 &gt;0 An error occurred.

103567 **CONSEQUENCES OF ERRORS**

103568 Default.

103569 **APPLICATION USAGE**

103570 It is difficult for an application to use every part of the file modes field of *ls -l* in a portable  
 103571 manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as  
 103572 implementations may have extensions. Applications can use this field to pass directly to a user  
 103573 printout or prompt, but actions based on its contents should generally be deferred, instead, to  
 103574 the *test* utility.

103575 The output of *ls* (with the *-l* and related options) contains information that logically could be  
 103576 used by utilities such as *chmod* and *touch* to restore files to a known state. However, this  
 103577 information is presented in a format that cannot be used directly by those utilities or be easily  
 103578 translated into a format that can be used. A character has been added to the end of the  
 103579 permissions string so that applications at least have an indication that they may be working in  
 103580 an area they do not understand instead of assuming that they can translate the permissions  
 103581 string into something that can be used. Future versions or related documents may define one or  
 103582 more specific characters to be used based on different standard additional or alternative access  
 103583 control mechanisms.

103584 As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one  
 103585 of the long listing formats must be used carefully on systems where filenames can contain  
 103586 embedded white space. Systems and system administrators should institute policies and user  
 103587 training to limit the use of such filenames.

103588 The number of disk blocks occupied by the file that it reports varies depending on underlying  
 103589 file system type, block size units reported, and the method of calculating the number of blocks.  
 103590 On some file system types, the number is the actual number of blocks occupied by the file  
 103591 (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the  
 103592 file size (usually making an allowance for indirect blocks, but ignoring holes).

103593 The *total* number provided when using *ls -l* does not necessarily correspond to the space that  
 103594 would be reclaimed if all the listed files were removed, because of hard links (and symbolic links  
 103595 if *-L* is present). The space for each listed file is counted in the total regardless of any  
 103596 relationship between the files.

103597 **EXAMPLES**103598 An example of a small directory tree being fully listed with *ls -laRF a* in the POSIX locale:

```

103599 a:
103600 total 11
103601 drwxr-xr-x  3 fox   prog           64 Jul  4 12:07 ./
103602 drwxrwxrwx  4 fox   prog        3264 Jul  4 12:09 ../
103603 drwxr-xr-x  2 fox   prog           48 Jul  4 12:07 b/
103604 -rwxr--r--  1 fox   prog          572 Jul  4 12:07 foo*

103605 a/b:
103606 total 4

```

```

103607      drwxr-xr-x   2 fox      prog      48 Jul  4 12:07 ./
103608      drwxr-xr-x   3 fox      prog      64 Jul  4 12:07 ../
103609      -rw-r--r--   1 fox      prog      700 Jul  4 12:07 bar

```

103610 where the "a:" line may be omitted by some implementations.

#### 103611 RATIONALE

103612 Some historical implementations of the *ls* utility show all entries in a directory except dot and  
 103613 dot-dot when a superuser invokes *ls* without specifying the `-a` option. When "normal" users  
 103614 invoke *ls* without specifying `-a`, they should not see information about any files with names  
 103615 beginning with a <period> unless they were named as *file* operands.

103616 Implementations are expected to traverse arbitrary depths when processing the `-R` option. The  
 103617 only limitation on depth should be based on running out of physical storage for keeping track of  
 103618 untraversed directories.

103619 The `-1` (one) option was historically found in BSD and BSD-derived implementations only. It is  
 103620 required in this volume of POSIX.1-2024 so that conforming applications might ensure that  
 103621 output is one entry per line, even if the output is to a terminal.

103622 The `-S` option was added in Issue 7, but had been provided by several implementations for  
 103623 many years. The description given in the standard documents historic practice, but does not  
 103624 match much of the documentation that described its behavior. Historical documentation  
 103625 typically described it as something like:

```

103626      -S          Sort by size (largest size first) instead of by name. Special character devices (listed
103627                  last) are sorted by name.

```

103628 even though the file type was never considered when sorting the output. Character special files  
 103629 do typically sort close to the end of the list because their file size on most implementations is  
 103630 zero. But they are sorted alphabetically with any other files that happen to have the same file  
 103631 size (zero), not sorted separately and added to the end.

103632 This volume of POSIX.1-2024 is frequently silent about what happens when mutually-exclusive  
 103633 options are specified. Except for `-R`, `-d`, and `-f`, the *ls* utility is required to accept multiple  
 103634 options from each mutually-exclusive option set without treating them as errors and to use the  
 103635 behavior specified by the last option given in each mutually-exclusive set. Since *ls* is one of the  
 103636 most aliased commands, it is important that the implementation perform intuitively. For  
 103637 example, if the alias were:

```

103638      alias ls="ls -C"

```

103639 and the user typed *ls -1* (one), single-text-column output should result, not an error.

103640 The `-g`, `-l` (ell), `-n`, and `-o` options are not mutually-exclusive options. They all enable long  
 103641 format output. They work together to determine whether the file's owner is written (no if `-g` is  
 103642 present), file's group is written (no if `-o` is present), and if the file's group or owner is written  
 103643 whether it is written as the name (default) or a string representation of the UID or GID number  
 103644 (if `-n` is present). The `-C`, `-m`, `-x`, and `-1` (one) are mutually-exclusive options and the first three  
 103645 of these disable long format output. The `-1` (one) option does not directly change whether or not  
 103646 long format output is enabled, but by overriding `-C`, `-m`, and `-x`, it can re-enable long format  
 103647 output that had been disabled by one of these options.

103648 Earlier versions of this standard did not describe the BSD `-A` option (like `-a`, but dot and dot-dot  
 103649 are not written out). It has been added due to widespread implementation.

103650 Implementations may make `-q` the default for terminals to prevent trojan horse attacks on  
 103651 terminals with special escape sequences. This is not required because:

103652 • Some control characters may be useful on some terminals; for example, a system might  
103653 write them as "\001" or "^A".

103654 • Special behavior for terminals is not relevant to applications portability.

103655 An early proposal specified that the *<optional alternate access method flag>* had to be '+' if there  
103656 was an alternate access method used on the file or *<space>* if there was not. This was changed to  
103657 be *<space>* if there is not and a single printable character if there is. This was done for three  
103658 reasons:

- 103659 1. There are historical implementations using characters other than '+'.
- 103660 2. There are implementations that vary this character used in that position to distinguish  
103661 between various alternate access methods in use.
- 103662 3. The standard developers did not want to preclude future specifications that might need a  
103663 way to specify more than one alternate access method.

103664 Nonetheless, implementations providing a single alternate access method are encouraged to use  
103665 '+'.

103666 Earlier versions of this standard did not have the *-k* option, which meant that the *-s* option  
103667 could not be used portably as its block size was implementation-defined, and the units used to  
103668 specify the number of blocks occupied by files in a directory in an *ls -l* listing were fixed as  
103669 512-byte units. The *-k* option has been added to provide a way for the *-s* option to be used  
103670 portably, and for consistency it also changes the aforementioned units from 512-byte to  
103671 1 024-byte.

103672 The *<date and time>* field in the *-l* format is specified only for the POSIX locale. As noted, the  
103673 format can be different in other locales. No mechanism for defining this is present in this volume  
103674 of POSIX.1-2024, as the appropriate vehicle is a messaging system; that is, the format should be  
103675 specified as a "message".

#### 103676 FUTURE DIRECTIONS

103677 If this utility is directed to display a pathname that contains any bytes that have the encoded  
103678 value of a *<newline>* character when *<newline>* is a terminator or separator in the output  
103679 format being used, implementations are encouraged to treat this as an error. A future version of  
103680 this standard may require implementations to treat this as an error.

103681 Allowing *-f* to ignore the *-A*, *-g*, *-l*, *-n*, *-o*, and *-s* options may be removed in a future version.

#### 103682 SEE ALSO

103683 *chmod*, *find*, *readlink*

103684 XBD [Section 7.3.2](#) (on page 139), [Section 4.7](#) (on page 97), [Chapter 8](#) (on page 167), [Section 12.2](#)  
103685 (on page 215), [<sys/stat.h>](#)

103686 XSH *fstatat()*

#### 103687 CHANGE HISTORY

103688 First released in Issue 2.

#### 103689 Issue 5

103690 A second FUTURE DIRECTION is added.

#### 103691 Issue 6

103692 The following new requirements on POSIX implementations derive from alignment with the  
103693 Single UNIX Specification:

- 103694           • In the **-F** option, other symbols are allowed for other file types.
- 103695           Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.
- 103696           The Open Group Base Resolution bwg2001-010 is applied, adding the **T** and **t** fields as part of
- 103697           the **XSI** option.
- 103698   **Issue 7**
- 103699           Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access
- 103700           method flag in the **STDOUT** section.
- 103701           Austin Group Interpretation 1003.1-2001 #128 is applied, clarifying the **DESCRIPTION** and the
- 103702           definition of the **-R** option.
- 103703           Austin Group Interpretation 1003.1-2001 #129 is applied, clarifying the behavior of *ls* when no
- 103704           operands are specified.
- 103705           Austin Group Interpretation 1003.1-2001 #198 is applied, clarifying the requirements for the **-H**
- 103706           option.
- 103707           SD5-XCU-ERN-50 is applied, adding the **-A** option.
- 103708           SD5-XCU-ERN-97 is applied, updating the **SYNOPSIS**.
- 103709           The **-S** option is added from The Open Group Technical Standard, 2006, Extended API Set
- 103710           Part 1.
- 103711           The **-f**, **-m**, **-n**, **-p**, **-s**, and **-x** options are moved from the **XSI** option to the **Base**.
- 103712           The description of the **-f**, **-s**, and **-t** options are revised and the **-k** option is added.
- 103713           POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0098 [424], XCU/TC1-2008/0099
- 103714           [424], XCU/TC1-2008/0100 [424], XCU/TC1-2008/0101 [424], XCU/TC1-2008/0102 [424],
- 103715           XCU/TC1-2008/0103 [424], XCU/TC1-2008/0104 [424], XCU/TC1-2008/0105 [423,424],
- 103716           XCU/TC1-2008/0106 [424], XCU/TC1-2008/0107 [424], XCU/TC1-2008/0108 [424],
- 103717           XCU/TC1-2008/0109 [424], XCU/TC1-2008/0110 [424], XCU/TC1-2008/0111 [423],
- 103718           XCU/TC1-2008/0112 [117], XCU/TC1-2008/0113 [117], XCU/TC1-2008/0114 [117],
- 103719           XCU/TC1-2008/0115 [424], and XCU/TC1-2008/0116 [424] are applied.
- 103720           POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0115 [963] and XCU/TC2-2008/0116
- 103721           [963] are applied.
- 103722   **Issue 8**
- 103723           Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is
- 103724           directed to display a pathname that contains any bytes that have the encoded value of a
- 103725           <newline> character when <newline> is a terminator or separator in the output format being
- 103726           used.
- 103727           Austin Group Defect 1023 is applied, clarifying the **-R** option with respect to subdirectory
- 103728           filenames beginning with a <period>.
- 103729           Austin Group Defect 1070 is applied, requiring that any filenames or pathnames that collate
- 103730           equally are further compared byte-by-byte using the collating sequence for the POSIX locale.
- 103731           Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 103732           Austin Group Defect 1146 is applied, clarifying the requirements for the status line written by *ls*
- 103733           **-l**.
- 103734           Austin Group Defect 1147 is applied, clarifying the requirements for trailing file type indicators
- 103735           (such as **' / '** for a directory).



- 103736 Austin Group Defect 1148 is applied, clarifying the behavior of the `-S` option for symbolic links.
- 103737 Austin Group Defect 1149 is applied, inserting a comma in the description of the `-r` option.
- 103738 Austin Group Defect 1185 is applied, changing the `COLUMNS` entry in ENVIRONMENT  
103739 VARIABLES.
- 103740 Austin Group Defect 1217 is applied, adding file type indicators for sockets.
- 103741 Austin Group Defect 1261 is applied, changing the STDOUT and EXAMPLES sections in relation  
103742 to the `<directory name>` output.
- 103743 Austin Group Defect 1457 is applied, adding `readlink` to the SEE ALSO section.
- 103744 Austin Group Defect 1703 is applied, changing ```at-sign``` to ```<commercial-at>```.

103745 **NAME**

103746 m4 — macro processor

103747 **SYNOPSIS**103748 m4 [-s] [-D *name*[=*val*]]... [-U *name*]... [*file*...]103749 **DESCRIPTION**103750 The *m4* utility is a macro processor that shall read one or more text files, process them according  
103751 to their included macro statements, and write the results to standard output.103752 **OPTIONS**103753 The *m4* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the order of the **-D**  
103754 and **-U** options shall be significant, and options can be interspersed with operands.

103755 The following options shall be supported:

103756 **-s** Enable line synchronization output for the *c17* preprocessor phase (that is, **#line**  
103757 directives).103758 **-D *name*[=*val*]**103759 Define *name* to *val* or to null if *=val* is omitted.103760 **-U *name*** Undefine *name*.103761 **OPERANDS**

103762 The following operand shall be supported:

103763 *file* A pathname of a text file to be processed. If no *file* is given, or if it is '-', the  
103764 standard input shall be read.103765 **STDIN**103766 The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.103767 **INPUT FILES**103768 The input file named by the *file* operand shall be a text file.103769 **ENVIRONMENT VARIABLES**103770 The following environment variables shall affect the execution of *m4*:103771 **LANG** Provide a default value for the internationalization variables that are unset or null.  
103772 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
103773 variables used to determine the values of locale categories.)103774 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
103775 internationalization variables.103776 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
103777 characters (for example, single-byte as opposed to multi-byte characters in  
103778 arguments and input files).103779 **LC\_MESSAGES**103780 Determine the locale that should be used to affect the format and contents of  
103781 diagnostic messages written to standard error.103782 **XSI** **NLSPATH** Determine the location of messages objects and message catalogs.103783 **ASYNCHRONOUS EVENTS**

103784 Default.

103785 **STDOUT**

103786 The standard output shall be the same as the input files, after being processed for macro  
103787 expansion.

103788 **STDERR**

103789 The standard error shall be used to display strings with the **errprint** macro, macro tracing  
103790 enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for  
103791 diagnostic messages.

103792 **OUTPUT FILES**

103793 None.

103794 **EXTENDED DESCRIPTION**

103795 The *m4* utility shall compare each token from the input against the set of built-in and user-  
103796 defined macros. If the token matches the name of a macro, then the token shall be replaced by  
103797 the macro's defining text, if any, then scanning for tokens shall resume at the start of the macro's  
103798 defining text concatenated with the subsequent input. If a token does not match the name of a  
103799 macro, it shall be written to standard output. Macros may have arguments, in which case the  
103800 arguments shall be substituted into the defining text before it is rescanned.

103801 No special meaning shall be given to characters enclosed between matching left and right  
103802 quoting strings, other than identifying nested quoting while finding the matching right quoting  
103803 string, but the outermost quoting strings shall themselves be discarded. By default, the left  
103804 quoting string consists of a grave accent (backquote) and the right quoting string consists of an  
103805 acute accent (single-quote); see also the **changequote** macro.

103806 Comments are written but not scanned for matching macro names; by default, the begin-  
103807 comment string consists of the <number-sign> character and the end-comment string consists of  
103808 a <newline>. See also the **changecom** and **dnl** macros.

103809 Name tokens shall consist of the longest possible sequence of letters, digits, and underscores,  
103810 where the first character is not a digit. Tokens not of this form shall not be treated as name  
103811 tokens. A macro call is a name token that matches the name of a built-in or user-defined macro.  
103812 Macro calls can have either of the following forms, which shall be distinguished by whether or  
103813 not the macro name is immediately followed by a <left-parenthesis>:

103814 *name*

103815 *name(arg1, arg2, ..., argn)*

103816 The application shall ensure that the <left-parenthesis> immediately follows the name of the  
103817 macro. If a token matching the name of a macro is not followed by a <left-parenthesis>, it shall  
103818 be handled as a use of that macro without arguments.

103819 If a macro name is followed by a <left-parenthesis>, the subsequent text shall be tokenized and  
103820 expanded until a token is encountered that is not a quoted string and whose expansion includes  
103821 a matching unquoted <right-parenthesis>. The expanded text between the <left-parenthesis>  
103822 and the matching unquoted <right-parenthesis> is the macro's argument text. An unquoted  
103823 <comma> character within the macro's argument text shall mark the end of one argument and  
103824 the beginning of the next argument unless the unquoted <comma> is enclosed within a nested  
103825 unquoted <left-parenthesis>, <right-parenthesis> pair. The unquoted <comma> characters that  
103826 separate the arguments, and any unquoted white-space characters at the beginning of each  
103827 argument, shall be discarded. All other characters in the macro's argument text, including any  
103828 white-space characters at the end of an argument and any nested parenthesized text, shall be  
103829 retained. The input text containing the macro name, the following <left-parenthesis>, and all  
103830 tokens up to and including the token whose expansion contained the matching unquoted <right-  
103831 parenthesis> shall be replaced, and tokenization shall resume on the result of performing

103832 argument substitution on the macro's defining text followed by any expanded text that followed  
 103833 the matching unquoted <right-parenthesis>. Otherwise, the macro name was not followed by a  
 103834 <left-parenthesis>, and tokenization shall resume on the result of performing argument  
 103835 substitution with zero arguments on the macro's defining text.

103836 During argument substitution, arguments shall be positionally defined and referenced. The  
 103837 string "\$1" in the defining text shall be replaced by the first argument. Systems shall support at  
 103838 least nine arguments; only the first nine can be referenced, using the strings "\$1" to "\$9",  
 103839 inclusive. The string "\$0" shall be replaced with the name of the macro. The string "\$#" shall  
 103840 be replaced by the number of arguments as a minimal string of decimal digits ('0' if the macro  
 103841 was invoked without being followed by a <left-parenthesis>, otherwise 1 more than the number  
 103842 of unquoted <comma> characters that divided arguments in the macro's argument text). The  
 103843 string "\$\*" shall be replaced by a list of all of the arguments, separated by <comma> characters.  
 103844 The string "\$@" shall be replaced by a list of all of the arguments separated by <comma>  
 103845 characters, and each argument shall be quoted using the current left and right quoting strings.  
 103846 The string "\${" produces unspecified behavior.

103847 If fewer arguments are supplied than are in the macro definition, the omitted arguments are  
 103848 taken to be null. It is not an error if more arguments are supplied than are in the macro  
 103849 definition.

103850 The *m4* utility shall make available the following built-in macros. They can be redefined, but  
 103851 once this is done the original meaning is lost. Their values shall be null unless otherwise stated.  
 103852 In the descriptions below, the term *defining text* refers to the value of the macro: the second  
 103853 argument to the **define** macro, among other things. Except for the first argument to the **eval**  
 103854 macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The  
 103855 string values produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval**  
 103856 built-in macros shall be in the form of a decimal-constant as defined in the C language.

103857 **changecom** The **changecom** macro shall set the begin-comment and end-comment strings.  
 103858 With no arguments, the comment mechanism shall be disabled. With a single non-  
 103859 null argument, that argument shall become the begin-comment and the <newline>  
 103860 shall become the end-comment string. With two non-null arguments, the first  
 103861 argument shall become the begin-comment string and the second argument shall  
 103862 become the end-comment string. The behavior is unspecified if either argument is  
 103863 provided but null, or if either argument includes letters, digits, underscore, or  
 103864 <left-parenthesis>. Systems shall support comment strings of at least five  
 103865 characters.

103866 **changequote** The **changequote** macro shall set the begin-quote and end-quote strings. With no  
 103867 arguments, the quote strings shall be set to the default values (that is, ` `'). The  
 103868 behavior is unspecified if there is a single argument, or if either argument is null or  
 103869 includes letters, digits, underscore, or <left-parenthesis>. With two non-null  
 103870 arguments, the first argument shall become the begin-quote string and the second  
 103871 argument shall become the end-quote string. Systems shall support quote strings  
 103872 of at least five characters.

103873 **decr** The defining text of the **decr** macro shall be its first argument decremented by 1. It  
 103874 shall be an error to specify an argument containing any non-numeric characters.  
 103875 The behavior is unspecified if **decr** is not immediately followed by a <left-  
 103876 parenthesis>.

103877 **define** The second argument shall become the defining text of the macro whose name is  
 103878 the first argument. It is unspecified whether the **define** macro deletes all prior  
 103879 definitions of the macro named by its first argument or preserves all but the

|        |                 |                                                                                                |
|--------|-----------------|------------------------------------------------------------------------------------------------|
| 103880 |                 | current definition of the macro. The behavior is unspecified if <b>define</b> is not           |
| 103881 |                 | immediately followed by a <left-parenthesis>.                                                  |
| 103882 | <b>defn</b>     | The defining text of the <b>defn</b> macro shall be the quoted definition (using the           |
| 103883 |                 | current quoting strings) of its arguments. The behavior is unspecified if <b>defn</b> is not   |
| 103884 |                 | immediately followed by a <left-parenthesis>.                                                  |
| 103885 | <b>divert</b>   | The <i>m4</i> utility maintains nine temporary buffers, numbered 1 to 9, inclusive.            |
| 103886 |                 | When the last of the input has been processed, any output that has been placed in              |
| 103887 |                 | these buffers shall be written to standard output in buffer-numerical order. The               |
| 103888 |                 | <b>divert</b> macro shall divert future output to the buffer specified by its argument.        |
| 103889 |                 | Specifying no argument or an argument of 0 shall resume the normal output                      |
| 103890 |                 | process. Output diverted to a stream with a negative number shall be discarded.                |
| 103891 |                 | Behavior is implementation-defined if a stream number larger than 9 is specified. It           |
| 103892 |                 | shall be an error to specify an argument containing any non-numeric characters.                |
| 103893 | <b>divnum</b>   | The defining text of the <b>divnum</b> macro shall be the number of the current output         |
| 103894 |                 | stream as a string.                                                                            |
| 103895 | <b>dnl</b>      | The <b>dnl</b> macro shall cause <i>m4</i> to discard all input characters up to and including |
| 103896 |                 | the next <newline>.                                                                            |
| 103897 | <b>dumpdef</b>  | The <b>dumpdef</b> macro shall write the defined text to standard error for each of the        |
| 103898 |                 | macros specified as arguments, or, if no arguments are specified, for all macros.              |
| 103899 | <b>errprint</b> | The <b>errprint</b> macro shall write its arguments to standard error. The behavior is         |
| 103900 |                 | unspecified if <b>errprint</b> is not immediately followed by a <left-parenthesis>.            |
| 103901 | <b>eval</b>     | The <b>eval</b> macro shall evaluate its first argument as an arithmetic expression, using     |
| 103902 |                 | signed integer arithmetic with at least 32-bit precision. At least the following C-            |
| 103903 |                 | language operators shall be supported, with precedence, associativity, and                     |
| 103904 |                 | behavior as described in <a href="#">Section 1.1.2.1</a> (on page 2457):                       |
| 103905 |                 | ( )                                                                                            |
| 103906 |                 | unary +                                                                                        |
| 103907 |                 | unary -                                                                                        |
| 103908 |                 | ~                                                                                              |
| 103909 |                 | !                                                                                              |
| 103910 |                 | binary *                                                                                       |
| 103911 |                 | /                                                                                              |
| 103912 |                 | %                                                                                              |
| 103913 |                 | binary +                                                                                       |
| 103914 |                 | binary -                                                                                       |
| 103915 |                 | <<                                                                                             |
| 103916 |                 | >>                                                                                             |
| 103917 |                 | <                                                                                              |
| 103918 |                 | <=                                                                                             |
| 103919 |                 | >                                                                                              |
| 103920 |                 | >=                                                                                             |
| 103921 |                 | ==                                                                                             |
| 103922 |                 | !=                                                                                             |
| 103923 |                 | binary &                                                                                       |
| 103924 |                 | ^                                                                                              |
| 103925 |                 |                                                                                                |
| 103926 |                 | &&                                                                                             |

|        |                |                                                                                                    |
|--------|----------------|----------------------------------------------------------------------------------------------------|
| 103927 |                |                                                                                                    |
| 103928 |                | Systems shall support octal and hexadecimal numbers as in the ISO C standard.                      |
| 103929 |                | The second argument, if specified, shall set the radix for the result; if the argument             |
| 103930 |                | is blank or unspecified, the default is 10. Behavior is unspecified if the radix falls             |
| 103931 |                | outside the range 2 to 36, inclusive. The third argument, if specified, sets the                   |
| 103932 |                | minimum number of digits in the result. Behavior is unspecified if the third                       |
| 103933 |                | argument is less than zero. It shall be an error to specify the second or third                    |
| 103934 |                | argument containing any non-numeric characters. The behavior is unspecified if                     |
| 103935 |                | <b>eval</b> is not immediately followed by a <left-parenthesis>.                                   |
| 103936 | <b>ifdef</b>   | If the first argument to the <b>ifdef</b> macro is defined, the defining text shall be the         |
| 103937 |                | second argument. Otherwise, the defining text shall be the third argument, if                      |
| 103938 |                | specified, or the null string, if not. The behavior is unspecified if <b>ifdef</b> is not          |
| 103939 |                | immediately followed by a <left-parenthesis>.                                                      |
| 103940 | <b>ifelse</b>  | The <b>ifelse</b> macro takes three or more arguments. If the first two arguments                  |
| 103941 |                | compare as equal strings, the defining text shall be the third argument. If the first              |
| 103942 |                | two arguments do not compare as equal strings and there are three arguments, the                   |
| 103943 |                | defining text shall be null. If the first two arguments do not compare as equal                    |
| 103944 |                | strings and there are four or five arguments, the defining text shall be the fourth                |
| 103945 |                | argument. If the first two arguments do not compare as equal strings and there are                 |
| 103946 |                | six or more arguments, the first three arguments shall be discarded and processing                 |
| 103947 |                | shall restart with the remaining arguments. The behavior is unspecified if <b>ifelse</b> is        |
| 103948 |                | not immediately followed by a <left-parenthesis>.                                                  |
| 103949 | <b>include</b> | The defining text for the <b>include</b> macro shall be the contents of the file named by          |
| 103950 |                | the first argument. It shall be an error if the file cannot be read. The behavior is               |
| 103951 |                | unspecified if <b>include</b> is not immediately followed by a <left-parenthesis>.                 |
| 103952 | <b>incr</b>    | The defining text of the <b>incr</b> macro shall be its first argument incremented by 1. It        |
| 103953 |                | shall be an error to specify an argument containing any non-numeric characters.                    |
| 103954 |                | The behavior is unspecified if <b>incr</b> is not immediately followed by a <left-                 |
| 103955 |                | parenthesis>.                                                                                      |
| 103956 | <b>index</b>   | The defining text of the <b>index</b> macro shall be the first character position (as a            |
| 103957 |                | string) in the first argument where a string matching the second argument begins                   |
| 103958 |                | (zero origin), or -1 if the second argument does not occur. The behavior is                        |
| 103959 |                | unspecified if <b>index</b> is not immediately followed by a <left-parenthesis>.                   |
| 103960 | <b>len</b>     | The defining text of the <b>len</b> macro shall be the length (as a string) of the first           |
| 103961 |                | argument. The behavior is unspecified if <b>len</b> is not immediately followed by a               |
| 103962 |                | <left-parenthesis>.                                                                                |
| 103963 | <b>m4exit</b>  | Exit from the <i>m4</i> utility. If the first argument is specified, it shall be the exit code. If |
| 103964 |                | no argument is specified, the exit code shall be zero. It shall be an error to specify             |
| 103965 |                | an argument containing any non-numeric characters. If the first argument is zero                   |
| 103966 |                | or no argument is specified, and an error has previously occurred (for example, a                  |
| 103967 |                | <i>file</i> operand that could not be opened), the exit status shall be non-zero.                  |
| 103968 | <b>m4wrap</b>  | The first argument shall be processed when EOF is reached. If the <b>m4wrap</b> macro              |
| 103969 |                | is used multiple times, the arguments specified shall be processed in the order in                 |
| 103970 |                | which the <b>m4wrap</b> macros were processed. The behavior is unspecified if <b>m4wrap</b>        |
| 103971 |                | is not immediately followed by a <left-parenthesis>.                                               |

|        |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 103972 | <b>mkstemp</b>  | The defining text shall be as if it were the resulting pathname after a successful call to the <i>mkstemp</i> ( <i>)</i> function defined in the System Interfaces volume of POSIX.1-2024 called with the first argument to the macro invocation. If a file is created, that file shall be closed. If a file could not be created, the <i>m4</i> utility shall write a diagnostic message to standard error and shall continue processing input but its final exit status shall be non-zero; the defining text of the macro shall be the empty string. The behavior is unspecified if <b>mkstemp</b> is not immediately followed by a <left-parenthesis>.                                                 |
| 103973 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103974 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103975 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103976 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103977 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103978 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103979 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103980 | <b>popdef</b>   | The <b>popdef</b> macro shall delete the current definition of its arguments, replacing that definition with the previous one. If there is no previous definition, the macro is undefined. The behavior is unspecified if <b>popdef</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                              |
| 103981 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103982 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103983 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103984 | <b>pushdef</b>  | The <b>pushdef</b> macro shall be equivalent to the <b>define</b> macro with the exception that it shall preserve any current definition for future retrieval using the <b>popdef</b> macro. The behavior is unspecified if <b>pushdef</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103985 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103986 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103987 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103988 | <b>shift</b>    | The defining text for the <b>shift</b> macro shall be a comma-separated list of its arguments except the first one. Each argument shall be quoted using the current quoting strings. The behavior is unspecified if <b>shift</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 103989 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103990 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103991 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103992 | <b>sinclude</b> | The <b>sinclude</b> macro shall be equivalent to the <b>include</b> macro, except that it shall not be an error if the file is inaccessible. The behavior is unspecified if <b>sinclude</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 103993 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103994 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103995 | <b>substr</b>   | The defining text for the <b>substr</b> macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. The behavior is unspecified if <b>substr</b> is not immediately followed by a <left-parenthesis>. |
| 103996 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103997 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103998 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 103999 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104000 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104001 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104002 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104003 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104004 | <b>syscmd</b>   | The <b>syscmd</b> macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. The string result shall not be rescanned for macros while setting the defining text. No output redirection shall be performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the <b>sysval</b> macro. The behavior is unspecified if <b>syscmd</b> is not immediately followed by a <left-parenthesis>.                                                                                                                                                                                                        |
| 104005 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104006 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104007 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104008 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104009 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104010 | <b>sysval</b>   | The defining text of the <b>sysval</b> macro shall be the exit value of the utility last invoked by the <b>syscmd</b> macro (as a string).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 104011 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104012 | <b>traceon</b>  | The <b>traceon</b> macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 104013 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104014 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104015 | <b>traceoff</b> | The <b>traceoff</b> macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 104016 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

104017       **translit**     The defining text of the **translit** macro shall be the first argument with every  
 104018                   character that occurs in the second argument replaced with the corresponding  
 104019                   character from the third argument. If no replacement character is specified for  
 104020                   some source character because the second argument is longer than the third  
 104021                   argument, that character shall be deleted from the first argument in **translit**'s  
 104022                   defining text. The behavior is unspecified if the '-' character appears within the  
 104023                   second or third argument anywhere besides the first or last character. The behavior  
 104024                   is unspecified if the same character appears more than once in the second  
 104025                   argument. The behavior is unspecified if **translit** is not immediately followed by a  
 104026                   <left-parenthesis>.

104027       **undefine**    The **undefine** macro shall delete all definitions (including those preserved using  
 104028                   the **pushdef** macro) of the macros named by its arguments. The behavior is  
 104029                   unspecified if **undefine** is not immediately followed by a <left-parenthesis>.

104030       **undivert**    The **undivert** macro shall cause immediate output of any text in temporary buffers  
 104031                   named as arguments, or all temporary buffers if no arguments are specified.  
 104032                   Buffers can be undiverted into other temporary buffers. Undiverting shall discard  
 104033                   the contents of the temporary buffer. The behavior is unspecified if an argument  
 104034                   contains any non-numeric characters.

#### 104035 EXIT STATUS

104036       The following exit values shall be returned:

104037       0    Successful completion.

104038       >0   An error occurred

104039       If the **m4exit** macro is used, the exit value can be specified by the input file.

#### 104040 CONSEQUENCES OF ERRORS

104041       Default.

#### 104042 APPLICATION USAGE

104043       The **defn** macro is useful for renaming macros, especially built-ins.

104044       Since **eval** defers to the ISO C standard, some operations have undefined behavior. In some  
 104045       implementations, division or remainder by zero cause a fatal signal, even if the division occurs  
 104046       on the short-circuited branch of "&&" or "||". Any operation that overflows in signed  
 104047       arithmetic produces undefined behavior. Likewise, using the **shift** operators with a shift amount  
 104048       that is not positive and smaller than the precision is undefined, as is shifting a negative number  
 104049       to the right. Historically, not all implementations obeyed C-language precedence rules: '~' and  
 104050       '!' were lower than '=='; '==' and '!=' were not lower than '<'; and '|' was not lower  
 104051       than '^'; the liberal use of "()" can force the desired precedence even with these non-  
 104052       compliant implementations. Furthermore, some traditional implementations treated '^' as an  
 104053       exponentiation operator, although most implementations now use "\*\*\*" as an extension for this  
 104054       purpose.

104055       When a macro has been multiply defined via the **pushdef** macro, it is unspecified whether the  
 104056       **define** macro will alter only the most recent definition (as though by **popdef** and **pushdef**), or  
 104057       replace the entire stack of definitions with a single definition (as though by **undefine** and  
 104058       **pushdef**). An application desiring particular behavior for the **define** macro in this case can  
 104059       redefine it accordingly.



104060 **EXAMPLES**

104061 If the file **m4src** contains the lines:

```
104062 The value of `VER' is "VER".
104063 ifdef(`VER', ``VER' is defined to be VER., VER is not defined.)
104064 ifelse(VER, 1, ``VER' is `VER'.)
104065 ifelse(VER, 2, ``VER' is `VER'., ``VER' is not 2.)
104066 end
```

104067 then the command

```
104068 m4 m4src
```

104069 or the command:

```
104070 m4 -U VER m4src
```

104071 produces the output:

```
104072 The value of VER is "VER".
104073 VER is not defined.
104074 VER is not 2.
104075 end
```

104076 The command:

```
104077 m4 -D VER m4src
```

104078 produces the output:

```
104079 The value of VER is ".
104080 VER is defined to be .
104081 VER is not 2.
104082 end
```

104083 The command:

```
104084 m4 -D VER=1 m4src
```

104085 produces the output:

```
104086 The value of VER is "1".
104087 VER is defined to be 1.
104088 VER is 1.
104089 VER is not 2.
104090 end
```

104091 The command:

```
104092 m4 -D VER=2 m4src
```

104093 produces the output:

```
104094 The value of VER is "2".
104095 VER is defined to be 2.
104096 VER is 2.
104097 end
```

104098 In the following six examples, an additional line is evaluated after this prologue of three  
104099 definitions:

```

104100 define(`macro', `argument 2 is :`$2':, called with $# arguments')dnl
104101 define(`argument$a', `Arguments')dnl
104102 define(`a', `.`)dnl

```

104103 1. The additional line:

```

104104 macro`'a
104105 produces:
104106 argument 2 is ::, called with 0 arguments.

```

104107 Explanation: *macro* is called with 0 arguments (as shown by the \$# substitution), the substitution of \$2 is the empty string, and the empty quoted string after the expansion text prevents concatenation with the subsequent 'a', which in turn lets macro *a* expand to the final '.'

104108

104109

104110

104111 2. The additional line:

```

104112 macro()a
104113 produces:
104114 argument 2 is ::, called with 1 Arguments

```

104115 Explanation: *macro* is called with one (empty string) argument; then the defining text ending in "arguments" is concatenated with the subsequent 'a' to form the next macro name *argument\$a* which is expanded into *Arguments* before the final output.

104116

104117

104118 3. The additional line:

```

104119 macro( 1, ( ,2,) , `3')
104120 produces:
104121 argument 2 is :( ,2,) :, called with 3 arguments

```

104122 Explanation: Leading (but not internal or trailing) space is removed before the argument substituted for \$2, and the unquoted commas embedded in parentheses do not delineate arguments.

104123

104124

104125 4. The additional line:

```

104126 macro( `1', `mac2(,`2',)', `3')
104127 produces:
104128 argument 2 is :mac2(,`2',):, called with 3 arguments

```

104129 Explanation: Regardless of whether *mac2* is a defined macro, quoting in the macro call prevents interpretation of "mac2" during argument collection, and the quoting in the defining text of *macro* prevents interpretation of "mac2" in the substitution of \$2 during rescan of the output of *macro*.

104130

104131

104132

104133 5. The additional line:

```

104134 undefine(`mac2')macro( 1, mac2(,2,), 3)
104135 produces:
104136 argument 2 is :mac2(,2,):, called with 3 arguments

```

104137 Explanation: *mac2* is not a macro name when scanned during argument collection, so it and the subsequent parenthesized text are used literally.

104138

104139 6. The additional line:  
 104140 `define(`mac2', `hi $@')macro( 1, mac2( ,2, ), 3)`  
 104141 produces:  
 104142 `argument 2 is :hi :, called with 5 arguments`  
 104143 Explanation: *mac2* is a macro name, so collecting the arguments to *macro* requires  
 104144 scanning the output of *mac2(,2,)* (the text `hi `',`2',`'` after substitution of `$@`); this  
 104145 output contains unquoted commas causing additional arguments to be visible to *macro*.

#### 104146 RATIONALE

104147 Historic System V-based behavior treated "`#{`" in a macro definition as two literal characters.  
 104148 However, this sequence is left unspecified so that implementations may offer extensions such as  
 104149 "`#{11}`" meaning the eleventh argument. Macros can still be defined with appropriate uses of  
 104150 nested quoting to result in a literal "`#{`" in the output after rescanning removes the nested  
 104151 quotes.

104152 In the **translit** built-in, historic System V-based behavior treated `'-'` as a literal; GNU behavior  
 104153 treats it as a range. This version of the standard allows either behavior.

104154 Earlier versions of this standard allowed the exit status to be either zero or non-zero when  
 104155 `m4exit(0)` is called after an error has occurred. Exiting with zero status is now disallowed as  
 104156 this hides the fact that an error occurred from shell scripts that check the exit status of *m4*.

#### 104157 FUTURE DIRECTIONS

104158 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 104159 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
 104160 format being used, implementations are encouraged to treat this as an error. A future version of  
 104161 this standard may require implementations to treat this as an error.

#### 104162 SEE ALSO

104163 [c17](#)  
 104164 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 104165 CHANGE HISTORY

104166 First released in Issue 2.

#### 104167 Issue 5

104168 The phrase "the defined text for macros written by the **dumpdef** macro" is added to the  
 104169 description of **STDERR**, and the description of **dumpdef** is updated to indicate that output is  
 104170 written to standard error. The description of **eval** is updated to indicate that the list of excluded  
 104171 C operators excludes unary `'&'` and `'.'`. In the description of **ifdef**, the phrase "and it is not  
 104172 defined to be zero" is deleted.

#### 104173 Issue 6

104174 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a `'&'` character in the  
 104175 excepted list.

104176 The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion  
 104177 buffers.

104178 The normative text is reworded to avoid use of the term "must" for application requirements.

104179 The Open Group Base Resolution bwg2000-006 is applied.

104180 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/31 is applied, replacing the EXAMPLES  
 104181 section.

104182 **Issue 7**

104183 Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro  
104184 obsolescent and adding a new **mkstemp** macro.

104185 Austin Group Interpretation 1003.1-2001 #207 is applied, clarifying the handling of white-space  
104186 characters that precede or trail any macro arguments.

104187 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
104188 apply (options can be interspersed with operands).

104189 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104190 SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED  
104191 DESCRIPTION.

104192 SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED  
104193 DESCRIPTION.

104194 SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED  
104195 DESCRIPTION.

104196 SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string "\$ {"  
104197 produces unspecified behavior.

104198 SD5-XCU-ERN-112 is applied, updating the **changequote** macro.

104199 SD5-XCU-ERN-118 is applied, clarifying the definition of the **define** macro in the EXTENDED  
104200 DESCRIPTION and APPLICATION USAGE sections.

104201 SD5-XCU-ERN-119 is applied, clarifying the definition of the **translit** macro in the EXTENDED  
104202 DESCRIPTION and RATIONALE sections.

104203 SD5-XCU-ERN-130, SD5-XCU-ERN-131, and SD5-XCU-ERN-137 are applied.

104204 The *m4* utility is moved from the XSI option to the Base.

104205 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0117 [241], XCU/TC1-2008/0118  
104206 [242,431], XCU/TC1-2008/0119 [242,431], and XCU/TC1-2008/0120 [325,430] are applied.

104207 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0117 [964], XCU/TC2-2008/0118 [970],  
104208 and XCU/TC2-2008/0119 [964] are applied.

104209 **Issue 8**

104210 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
104211 directed to display a pathname that contains any bytes that have the encoded value of a  
104212 <newline> character when <newline> is a terminator or separator in the output format being  
104213 used.

104214 Austin Group Defect 984 is applied, requiring that the exit status of *m4* is non-zero when **m4exit**  
104215 is called with a first argument of zero or with no arguments after an error has occurred.

104216 Austin Group Defect 1072 is applied, clarifying the handling of macro arguments.

104217 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

104218 Austin Group Defect 1330 is applied, removing the obsolescent **maketemp** macro.

104219 Austin Group Defect 1514 is applied, changing the RATIONALE section to use the same  
104220 terminology as the normative text to which it refers.

104221 Austin Group Defect 1570 is applied, removing extra spacing in "==".

104222 Austin Group Defect 1658 is applied, changing ``whitespace characters'' to ``white-space

104223

characters”.

104224

Austin Group Defect 1730 is applied, adding square brackets around *file...* in the SYNOPSIS.

104225

104226 **NAME**

104227 mailx — process messages

104228 **SYNOPSIS**104229 **Send Mode**104230 mailx [-E] [-s *subject*] *address*...104231 **Receive Mode**

104232 UP mailx -e

104233 mailx [-HiNn] [-F] [-u *user*]104234 mailx -f [-HiNn] [-F] [*file*]104235 **DESCRIPTION**

104236 The *mailx* utility provides a message sending and receiving facility. It has two major modes,  
 104237 selected by the options used: Send Mode and Receive Mode.

104238 On systems that do not support the User Portability Utilities option, an application using *mailx*  
 104239 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first  
 104240 character of one or more lines is <tilde> ('~'), all characters in the input message shall appear in  
 104241 the delivered message, but additional characters may be inserted in the message before it is  
 104242 retrieved.

104243 UP On systems supporting the User Portability Utilities option, mail-receiving capabilities and other  
 104244 interactive features, Receive Mode, described below, also shall be enabled.

104245 **Send Mode**

104246 Send Mode can be used by applications or users to send messages from the text in standard  
 104247 input. The message shall be passed to the mail delivery software. The mail delivery software  
 104248 shall process passed messages according to the rules of IETF RFC 5322.

104249 UP **Receive Mode**

104250 Receive Mode is more oriented towards interactive users. Mail can be read and sent in this  
 104251 interactive mode.

104252 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to  
 104253 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the  
 104254 message as it is entered.

104255 Incoming mail shall be stored in one or more unspecified locations for each user, collectively  
 104256 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system  
 104257 mailbox shall be the default place to find new mail. As messages are read, they shall be marked  
 104258 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is  
 104259 called the **mbox** and is normally located in the directory referred to by the *HOME* environment  
 104260 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).  
 104261 Messages shall remain in this file until explicitly removed. When the *-f* option is used to read  
 104262 mail messages from secondary files, messages shall be retained in those files unless specifically  
 104263 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred  
 104264 to in this section as simply “mailboxes”, unless more specific identification is required.

104265 **OPTIONS**

- 104266 The *mailx* utility shall conform to XBD [Section 12.2](#) (on page 215).
- 104267 The following options shall be supported. (Only the **-E** and **-s subject** options are required on all  
104268 systems. The other options are required only on systems supporting the User Portability Utilities  
104269 option.)
- 104270 **-E** Discard messages with an empty message body.
- 104271 UP **-e** Test for the presence of mail in the system mailbox. The *mailx* utility shall write  
104272 nothing and exit with a successful return code if there is mail to read.
- 104273 UP **-f** Read messages from the file named by the *file* operand instead of the system  
104274 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from **mbox**  
104275 instead of the system mailbox.
- 104276 UP **-F** Record the message in a file named after the first recipient. The name is the login-  
104277 name portion of the address found first in the **To** field in the message header.  
104278 Overrides the **record** variable, if set (see [Internal Variables in mailx](#), on page 3109).
- 104279 UP **-H** Write a header summary only.
- 104280 UP **-i** Ignore interrupts. (See also **ignore**.)
- 104281 UP **-n** Do not initialize from the system default start-up file. See the EXTENDED  
104282 DESCRIPTION section.
- 104283 UP **-N** Do not write an initial header summary.
- 104284 **-s subject** Set the **Subject** header field to *subject*. All characters in the *subject* string shall  
104285 appear in the delivered message. The results are unspecified if *subject* is longer  
104286 than {LINE\_MAX} – 10 bytes or contains a <newline>.
- 104287 UP **-u user** Read the system mailbox of the login name *user*. This shall only be successful if  
104288 the invoking user has appropriate privileges to read the system mailbox of that  
104289 user.

104290 **OPERANDS**

- 104291 The following operands shall be supported:
- 104292 *address* Addressee of message. When **-n** is specified and no user start-up files are accessed  
104293 (see the EXTENDED DESCRIPTION section), the user or application shall ensure  
104294 this is an address to pass to the mail delivery system. Any system or user start-up  
104295 files may enable aliases (see **alias** under [Commands in mailx](#), on page 3112) that  
104296 may modify the form of *address* before it is passed to the mail delivery system.
- 104297 UP *file* A pathname of a file to be read instead of the system mailbox when **-f** is specified.  
104298 The meaning of the *file* operand shall be affected by the contents of the **folder**  
104299 internal variable; see [Internal Variables in mailx](#) (on page 3109).

104300 **STDIN**

- 104301 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the  
104302 UP message to be delivered to the specified addresses. When in Receive Mode, user commands  
104303 shall be accepted from *stdin*. If the User Portability Utilities option is not supported, standard  
104304 input lines beginning with a <tilde> ('~') character produce unspecified results.
- 104305 UP If the User Portability Utilities option is supported, then in both Send and Receive Modes,  
104306 standard input lines beginning with the escape character (usually <tilde> ('~')) shall affect  
104307 processing as described in [Command Escapes in mailx](#) (on page 3122).

104308 **INPUT FILES**

104309 When *mailx* is used as described by this volume of POSIX.1-2024, the *file* operand (see the `-f`  
 104310 option) and the **mbox** shall be text files containing mail messages, formatted as described in the  
 104311 OUTPUT FILES section. The nature of the system mailbox is unspecified; it need not be a file.

104312 **ENVIRONMENT VARIABLES**

104313 UP Some of the functionality described in this section shall be provided on implementations that  
 104314 support the User Portability Utilities option as described in the text, and is not further shaded  
 104315 for this option.

104316 The following environment variables shall affect the execution of *mailx*:

104317 **DEAD** Determine the pathname of the file in which to save partial messages in case of  
 104318 interrupts or delivery errors. The default shall be **dead.letter** in the directory  
 104319 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is  
 104320 unspecified if the User Portability Utilities option is not supported and *DEAD* is  
 104321 not defined with the value **/dev/null**.

104322 **EDITOR** Determine the name of a utility to invoke when the **edit** (see [Commands in mailx](#),  
 104323 on page 3112) or **~e** (see [Command Escapes in mailx](#), on page 3122) command is  
 104324 XSI used. The default editor is unspecified. On XSI-conformant systems it is *ed*. The  
 104325 effects of this variable are unspecified if the User Portability Utilities option is not  
 104326 supported.

104327 **HOME** Determine the pathname of the user's home directory.

104328 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 104329 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 104330 variables used to determine the values of locale categories.)

104331 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 104332 internationalization variables.

104333 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 104334 characters (for example, single-byte as opposed to multi-byte characters in  
 104335 arguments and input files) and the handling of case-insensitive address and header  
 104336 field name comparisons.

104337 **LC\_TIME** This variable may determine the format and contents of the date and time strings  
 104338 written by *mailx*. This volume of POSIX.1-2024 specifies the effects of this variable  
 104339 only for systems supporting the User Portability Utilities option.

104340 **LC\_MESSAGES**

104341 Determine the locale that should be used to affect the format and contents of  
 104342 diagnostic messages written to standard error and informative messages written to  
 104343 standard output.

104344 **LISTER** Determine a string representing the command for writing the contents of the  
 104345 **folder** directory to standard output when the **folders** command is given (see  
 104346 **folders** in [Commands in mailx](#), on page 3112). Any string acceptable as a  
 104347 *command\_string* operand to the *sh -c* command shall be valid. If this variable is null  
 104348 or not set, the output command shall be *ls*. The effects of this variable are  
 104349 unspecified if the User Portability Utilities option is not supported.

104350 **MAILRC** Determine the pathname of the user start-up file. The default shall be **.mailrc** in the  
 104351 directory referred to by the *HOME* environment variable. The behavior of *mailx* is  
 104352 unspecified if the User Portability Utilities option is not supported and *MAILRC* is  
 104353 not defined with the value **/dev/null**.



|        |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------|-----|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 104354 |     | <i>MBOX</i>                   | Determine a pathname of the file to save messages from the system mailbox that have been read. The <b>exit</b> command shall override this function, as shall saving the message explicitly in another file. The default shall be <b>mbox</b> in the directory named by the <i>HOME</i> variable. The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 104355 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104356 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104357 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104358 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104359 | XSI | <i>NLSPATH</i>                | Determine the location of messages objects and message catalogs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 104360 |     | <i>PAGER</i>                  | Determine a string representing an output filtering or pagination command for writing the output to the terminal. Any string acceptable as a <i>command_string</i> operand to the <i>sh -c</i> command shall be valid. When standard output is a terminal device, the message output shall be piped through the command if the <i>mailx</i> internal variable <b>crf</b> is set to a value less than the total number of lines in the message; see <a href="#">Internal Variables in mailx</a> (on page 3109). When standard output is not a terminal device, it is unspecified whether the message output is written directly to standard output or is subject to pagination. If the <i>PAGER</i> variable is null or not set, the paginator shall be either <i>more</i> or another paginator utility documented in the system documentation. The effects of this variable are unspecified if the User Portability Utilities option is not supported. |
| 104361 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104362 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104363 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104364 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104365 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104366 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104367 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104368 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104369 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104370 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104371 |     | <i>SHELL</i>                  | Determine the name of a preferred command interpreter. The default shall be <i>sh</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104372 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104373 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104374 |     | <i>TERM</i>                   | If the internal variable <b>screen</b> is not specified, determine the name of the terminal type to indicate in an unspecified manner the number of lines in a screenful of header summaries. If <i>TERM</i> is not set or is set to null, an unspecified default terminal type shall be used and the value of a screenful is unspecified. The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 104375 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104376 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104377 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104378 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104379 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104380 |     | <i>TZ</i>                     | This variable may determine the timezone used to calculate date and time strings written by <i>mailx</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 104381 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104382 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104383 |     | <i>VISUAL</i>                 | Determine a pathname of a utility to invoke when the <b>visual</b> command (see <a href="#">Commands in mailx</a> , on page 3112) or <b>~v</b> command-escape (see <a href="#">Command Escapes in mailx</a> , on page 3122) is used. If this variable is null or not set, the full-screen editor shall be <i>vi</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 104384 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104385 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104386 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104387 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104388 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104389 |     |                               | When <i>mailx</i> is in Send Mode and standard input is not a terminal, it shall take the standard action for all signals.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 104390 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104391 | UP  | In <b>Receive Mode, or in</b> | Send Mode when standard input is a terminal, if a SIGINT signal is received:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104392 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104393 | UP  | 1.                            | If in command mode, the current command, if there is one, shall be aborted, and a command-mode prompt shall be written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 104394 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104395 |     | 2.                            | If in input mode:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 104396 | UP  | a.                            | If <b>ignore</b> is set, <i>mailx</i> shall write "@\n", discard the current input line, and continue processing, bypassing the message-abort mechanism described in item 2b.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 104397 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104398 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

- 104399 UP            b. If the interrupt was received while sending mail, either when in Receive Mode or  
 104400            in Send Mode, a message shall be written, and another subsequent interrupt, with  
 104401            no other intervening characters typed, shall be required to abort the mail message.  
 104402 UP            If in Receive Mode and another interrupt is received, a command-mode prompt  
 104403            shall be written. If in Send Mode and another interrupt is received, *mailx* shall  
 104404            terminate with a non-zero status.

104405            In both cases listed in item b, if the message is not empty:

- 104406 UP            i. If **save** is enabled and the file named by *DEAD* can be created, the message  
 104407            shall be written to the file named by *DEAD*. If the file exists, the message  
 104408            shall be written to replace the contents of the file.
- 104409 UP            ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the  
 104410            message shall not be saved.

104411            The *mailx* utility shall take the standard action for all other signals.

#### 104412 **STDOUT**

104413            In command and input modes, all output, including prompts and messages, shall be written to  
 104414            standard output.

#### 104415 **STDERR**

104416            The standard error shall be used only for diagnostic messages.

#### 104417 **OUTPUT FILES**

104418            Various *mailx* commands and command escapes can create or add to files, including the **mbox**,  
 104419            the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of  
 104420            POSIX.1-2024, these files shall be text files, formatted as follows:

104421            line beginning with **From<space>**  
 104422            [one or more *header fields*; see [Commands in mailx](#) (on page 3112)]  
 104423            *empty line*  
 104424            [zero or more *body lines*  
 104425            *empty line*]  
 104426            [line beginning with **From<space> . . . ]**

104427            where each message begins with the **From <space>** line shown, preceded by the beginning of  
 104428            the file or an empty line. (The **From <space>** line is considered to be part of the message header,  
 104429            but not one of the header fields referred to in [Commands in mailx](#) (on page 3112); thus, it shall  
 104430            not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the  
 104431            **From <space>** line and any additional header lines are unspecified, except that none shall be  
 104432            empty. The format of a message body line is also unspecified, except that no line following an  
 104433            empty line shall start with **From <space>**; *mailx* shall modify any such user-entered message  
 104434            body lines (following an empty line and beginning with **From <space>**) by adding one or more  
 104435            characters to precede the 'F'; it may add these characters to **From <space>** lines that are not  
 104436            preceded by an empty line.

104437            When a message from the system mailbox or entered by the user is not a text file, it is  
 104438            implementation-defined how such a message is stored in files written by *mailx*.

#### 104439 **EXTENDED DESCRIPTION**

104440 UP            The functionality in the entire EXTENDED DESCRIPTION section shall be provided on  
 104441            implementations supporting the User Portability Utilities option. The functionality described in  
 104442            this section shall be provided on implementations that support the User Portability Utilities  
 104443            option (and the rest of this section is not further shaded for this option).

104444 The *mailx* utility need not support for all character encodings in all circumstances. For example,  
 104445 inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data need not  
 104446 be portable to non-internationalized systems, and so on. Under these circumstances, it is  
 104447 recommended that only characters defined in the ISO/IEC 646:1991 standard International  
 104448 Reference Version (equivalent to ASCII) 7-bit range of characters be used.

104449 When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of  
 104450 header-summary lines (if `-N` was not specified and there are messages, see below), followed by  
 104451 a prompt indicating that *mailx* can accept regular commands (see [Commands in mailx](#), on page  
 104452 3112); this is termed *command mode*. The page of header-summary lines shall contain the first  
 104453 new message if there are new messages, or the first unread message if there are unread  
 104454 messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and  
 104455 standard input is a terminal, if no subject is specified on the command line and the `asksub`  
 104456 variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input  
 104457 mode. This input mode shall also be entered when using one of the Receive Mode synopsis  
 104458 forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or  
 104459 **mail** commands and standard input is a terminal. When the message is typed and the end of the  
 104460 message is encountered, the message shall be passed to the mail delivery software. Commands  
 104461 can be entered by beginning a line with the escape character (by default, `<tilde>` ('~')) followed  
 104462 by a single command letter and optional arguments. See [Commands in mailx](#) (on page 3112) for  
 104463 a summary of these commands. It is unspecified what effect these commands will have if  
 104464 standard input is not a terminal when a message is entered using either the Send Mode  
 104465 synopsis, or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

104466 **Note:** For notational convenience, this section uses the default escape character, `<tilde>`, in all  
 104467 references and examples.

104468 At any time, the behavior of *mailx* shall be governed by a set of environmental and internal  
 104469 variables. These are flags and valued parameters that can be set and cleared via the *mailx set*  
 104470 and *unset* commands.

104471 Regular commands are of the form:

104472 `[command] [msglist] [argument ...]`

104473 If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands  
 104474 shall be recognized by the escape character, and lines not treated as commands shall be taken as  
 104475 input for the message.

104476 In command mode, each message shall be assigned a sequential number, starting with 1.

104477 All messages have a state that shall affect how they are displayed in the header summary and  
 104478 how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a  
 104479 *current* message, which shall be marked by a '>' at the beginning of a line in the header  
 104480 summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current  
 104481 message shall be the first new message, if there is a new message, or the first unread message if  
 104482 there is an unread message, or the first message if there are any messages, or unspecified if there  
 104483 are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*)  
 104484 or an optional single message (*message*) on which to operate shall leave the current message set  
 104485 to the highest-numbered message of the messages specified, unless the command deletes  
 104486 messages, in which case the current message shall be set to the first undeleted message (that is, a  
 104487 message not in the deleted state) after the highest-numbered message deleted by the command,  
 104488 if one exists, or the first undeleted message before the highest-numbered message deleted by the  
 104489 command, if one exists, or to an unspecified value if there are no remaining undeleted messages.  
 104490 All messages shall be in one of the following states:

|        |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 104491 | <i>new</i>       | The message is present in the system mailbox and has not been viewed by the user or moved to any other state. Messages in state <i>new</i> when <i>mailx</i> quits shall be retained in the system mailbox.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 104492 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104493 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104494 | <i>unread</i>    | The message has been present in the system mailbox for more than one invocation of <i>mailx</i> and has not been viewed by the user or moved to any other state. Messages in state <i>unread</i> when <i>mailx</i> quits shall be retained in the system mailbox.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 104495 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104496 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104497 | <i>read</i>      | The message has been processed by one of the following commands: <b>~f</b> , <b>~m</b> , <b>~F</b> , <b>~M</b> , <b>copy</b> , <b>mbox</b> , <b>next</b> , <b>pipe</b> , <b>print</b> , <b>Print</b> , <b>top</b> , <b>type</b> , <b>Type</b> , <b>undelete</b> . The <b>dp</b> and <b>dt</b> commands shall also cause the message they write, if any, to be marked as <i>read</i> . If the <b>autoprnt</b> variable is set, the <b>delete</b> command shall also cause the message it writes, if any, to be marked as <i>read</i> . Messages that are in the system mailbox and in state <i>read</i> when <i>mailx</i> quits shall be saved in the <b>mbox</b> , unless the internal variable <b>hold</b> was set. Messages that are in the <b>mbox</b> or in a secondary mailbox and in state <i>read</i> when <i>mailx</i> quits shall be retained in their current location. |
| 104498 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104499 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104500 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104501 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104502 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104503 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104504 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104505 | <i>deleted</i>   | The message has been processed by one of the following commands: <b>delete</b> , <b>dp</b> , <b>dt</b> . Messages in state <i>deleted</i> when <i>mailx</i> quits shall be deleted. Deleted messages shall be ignored until <i>mailx</i> quits or changes mailboxes or they are specified to the undelete command; for example, the message specification <i>/string</i> shall only search the <b>Subject</b> header fields of messages that have not yet been deleted, unless the command operating on the list of messages is <b>undelete</b> . No deleted message or deleted message header shall be displayed by any <i>mailx</i> command other than <b>undelete</b> .                                                                                                                                                                                                        |
| 104506 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104507 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104508 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104509 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104510 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104511 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104512 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104513 | <i>preserved</i> | The message has been processed by a <b>preserve</b> command. When <i>mailx</i> quits, the message shall be retained in its current location.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 104514 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104515 | <i>saved</i>     | The message has been processed by one of the following commands: <b>save</b> or <b>write</b> . If the current mailbox is the system mailbox, and the internal variable <b>keepsave</b> is set, messages in the state <i>saved</i> shall be saved to the file designated by the <b>MBOX</b> variable (see the ENVIRONMENT VARIABLES section). If the current mailbox is the system mailbox, messages in the state <i>saved</i> shall be deleted from the current mailbox, when the <b>quit</b> or <b>file</b> command is used to exit the current mailbox.                                                                                                                                                                                                                                                                                                                         |
| 104516 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104517 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104518 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104519 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104520 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104521 |                  | The header-summary line for each message shall indicate the state of the message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 104522 |                  | Many commands take an optional list of messages ( <i>msglist</i> ) on which to operate, which defaults to the current message. A <i>msglist</i> is a list of message specifications separated by <blank> characters, which can include:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 104523 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104524 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104525 | <i>n</i>         | Message number <i>n</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 104526 | <b>+</b>         | The next undeleted message, or the next deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 104527 | <b>-</b>         | The next previous undeleted message, or the next previous deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104528 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 104529 | <b>.</b>         | The current message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 104530 | <b>^</b>         | The first undeleted message, or the first deleted message for the <b>undelete</b> command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 104531 | <b>\$</b>        | The last message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 104532 | <b>*</b>         | All messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

- 104533        *n-m*     An inclusive range of message numbers.
- 104534        *address*   All messages from *address*; any address as shown in a header summary shall be  
104535                          matchable in this form.
- 104536        */string*   All messages with *string* in the **Subject** header field (case ignored).
- 104537        *:c*        All messages of type *c*, where *c* shall be one of:
- 104538                          *d*     Deleted messages.
- 104539                          *n*     New messages.
- 104540                          *o*     Old messages (any not in state *read* or *new*).
- 104541                          *r*     Read messages.
- 104542                          *u*     Unread messages.

104543        Other commands take an optional message (*message*) on which to operate, which defaults to the  
104544                          current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more  
104545                          than one message is specified, only the first shall be operated on.

104546        Other arguments are usually arbitrary strings whose usage depends on the command involved.

### 104547        **Start-Up in mailx**

104548        At start-up time, *mailx* shall take the following steps in sequence:

- 104549                          1. Establish all variables at their stated default values.
- 104550                          2. Process command line options, overriding corresponding default values.
- 104551                          3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables  
104552                          that are present in the environment, overriding the corresponding default values.
- 104553                          4. Read *mailx* commands from an unspecified system start-up file, unless the **-n** option is  
104554                          given, to initialize any internal *mailx* variables and aliases.
- 104555                          5. Process the user start-up file of *mailx* commands named in the user *MAILRC* variable.

104556        Most regular *mailx* commands are valid inside start-up files, the most common use being to set  
104557                          up initial display options and alias lists. The following commands shall be invalid in a start-up  
104558                          file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **Save**, **shell**, **visual**, **Copy**, **followup**, and  
104559                          **Followup**. Any errors in a start-up file shall either cause *mailx* to terminate with a diagnostic  
104560                          message and a non-zero status or to continue after writing a diagnostic message, ignoring the  
104561                          remainder of the lines in the file.

104562        A blank line in a start-up file shall be ignored.

### 104563        **Internal Variables in mailx**

104564        The following variables are internal *mailx* variables. Each internal variable can be set via the  
104565                          *mailx set* command at any time. The **unset** and **set noname** commands can be used to erase  
104566                          variables.

104567        In the following list, variables shown as:

104568        *variable*

104569        represent Boolean values. Variables shown as:

104570        *variable=value*

- 104571 shall be assigned string or numeric values. For string values, the rules in [Commands in mailx](#)  
104572 (on page 3112) concerning filenames and quoting shall also apply.
- 104573 The defaults specified here may be changed by the unspecified system start-up file unless the  
104574 user specifies the **-n** option.
- 104575 **allnet** All network names whose login name components match shall be treated as  
104576 identical. This shall cause the *msglist* message specifications to behave similarly.  
104577 The default shall be **noallnet**. See also the **alternates** command and the **metoo**  
104578 variable.
- 104579 **append** Append messages to the end of the **mbox** file upon termination instead of placing  
104580 them at the beginning. The default shall be **noappend**. This variable shall not  
104581 affect the **save** command when saving to **mbox**.
- 104582 **ask, asksub** Prompt for a value for the **Subject** header field on outgoing mail if one is not  
104583 specified on the command line with the **-s** option. The **ask** and **asksub** forms are  
104584 synonyms; the system shall refer to **asksub** and **noasksub** in its messages, but shall  
104585 accept **ask** and **noask** as user input to mean **asksub** and **noasksub**. It shall not be  
104586 possible to set both **ask** and **noasksub**, or **noask** and **asksub**. The default shall be  
104587 **asksub**, but no prompting shall be done if standard input is not a terminal.
- 104588 **askbcc** Prompt for the blind copy list. The default shall be **noaskbcc**.
- 104589 **askcc** Prompt for the copy list. The default shall be **noaskcc**.
- 104590 **autoprint** Enable automatic writing of messages after **delete** and **undelete** commands. The  
104591 default shall be **noautoprint**.
- 104592 **bang** Enable the special-case treatment of <exclamation-mark> characters ('!') in !  
104593 commands and *~!command* escapes; see the **Invoke Shell Command** command and  
104594 [Command Escapes in mailx](#) (on page 3122). The default shall be **nobang**, disabling  
104595 the expansion of '!' in the *command* argument to the ! command and the  
104596 *~!command* escape.
- 104597 **cmd=command**  
104598 Set the default command to be invoked by the **pipe** command. The default shall be  
104599 **nocmd**.
- 104600 **crt=number** Paginate message output as described for the *PAGER* variable. The default shall be  
104601 **nocrt**, disabling this pagination. If it is set to null, the value used is  
104602 implementation-defined.
- 104603 XSI **debug** Enable verbose diagnostics for debugging. Messages are not delivered. The  
104604 default shall be **nodebug**.
- 104605 **dot** When **dot** is set, a <period> on a line by itself during message input from a  
104606 terminal shall also signify end-of-file (in addition to normal end-of-file). The  
104607 default shall be **nodot**. If **ignoreeof** is set (see below), a setting of **nodot** shall be  
104608 ignored and <period> and the *~.* command escape are the only methods to  
104609 terminate input mode.
- 104610 **escape=c** Set the command escape character to be the character 'c'. By default, the  
104611 command escape character shall be <tilde>. If **escape** is unset, <tilde> shall be  
104612 used; if it is set to null, command escaping shall be disabled.
- 104613 **flipr** Reverse the meanings of the **R** and **r** commands. The default shall be **noflipr**.

|        |                                     |                                                                                                            |
|--------|-------------------------------------|------------------------------------------------------------------------------------------------------------|
| 104614 | <b>folder</b> = <i>directory</i>    |                                                                                                            |
| 104615 |                                     | The default directory for saving mail files. User-specified filenames beginning with                       |
| 104616 |                                     | a <plus-sign> ('+') shall be expanded by preceding the filename with this                                  |
| 104617 |                                     | directory name to obtain the real pathname. If <i>directory</i> does not start with a                      |
| 104618 |                                     | <slash> ('/'), the contents of <i>HOME</i> shall be prefixed to it. The default shall be                   |
| 104619 |                                     | <b>nofolder</b> . If <b>folder</b> is unset, user-specified filenames beginning with '+' shall             |
| 104620 |                                     | refer to files in the current directory that begin with the literal '+' character. See                     |
| 104621 |                                     | also <b>outfolder</b> below. The <b>folder</b> value need not affect the processing of the files           |
| 104622 |                                     | named in <i>MBOX</i> and <i>DEAD</i> .                                                                     |
| 104623 | <b>header</b>                       | Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The                       |
| 104624 |                                     | default shall be <b>header</b> .                                                                           |
| 104625 | <b>hold</b>                         | Disable message moving of read messages from the system mailbox to the <b>mbox</b>                         |
| 104626 |                                     | save file upon normal program termination or folder change. This automatic mail                            |
| 104627 |                                     | management is complemented with the commands <b>hold</b> (and <b>preserve</b> ), <b>mbox</b> ,             |
| 104628 |                                     | and <b>touch</b> , which partially override the <b>hold</b> variable. The default shall be <b>nohold</b> . |
| 104629 | <b>ignore</b>                       | Ignore interrupts while entering messages. The default shall be <b>noignore</b> .                          |
| 104630 | <b>ignoreeof</b>                    | Ignore normal end-of-file during message input. Input can be terminated only by                            |
| 104631 |                                     | entering a <period> ('.') on a line by itself or by the ~. command escape. The                             |
| 104632 |                                     | default shall be <b>noignoreeof</b> . See also <b>dot</b> above.                                           |
| 104633 | <b>indentprefix</b> = <i>string</i> |                                                                                                            |
| 104634 |                                     | A string that shall be added as a prefix to each line that is inserted into the message                    |
| 104635 |                                     | by the ~m command escape. This variable shall default to one <tab>.                                        |
| 104636 | <b>keep</b>                         | When a system mailbox, secondary mailbox, or <b>mbox</b> is empty, truncate it to zero                     |
| 104637 |                                     | length instead of removing it. The default shall be <b>nokeep</b> .                                        |
| 104638 | <b>keepsave</b>                     | Keep the messages that have been saved from the system mailbox into other files                            |
| 104639 |                                     | in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default                 |
| 104640 |                                     | shall be <b>nokeepsave</b> .                                                                               |
| 104641 | <b>metoo</b>                        | Suppress the deletion of the user's login name from the recipient list when                                |
| 104642 |                                     | replying to a message or sending to a group. The default shall be <b>nometoo</b> .                         |
| 104643 | <b>onehop</b>                       | When responding to a message that was originally sent to several recipients, the                           |
| 104644 |                                     | other recipient addresses are normally forced to be relative to the originating                            |
| 104645 |                                     | author's machine for the response. This flag disables alteration of the recipients'                        |
| 104646 |                                     | addresses, improving efficiency in a network where all machines can send directly                          |
| 104647 |                                     | to all other machines (that is, one hop away). The default shall be <b>noonehop</b> .                      |
| 104648 | <b>outfolder</b>                    | Cause the files used to record outgoing messages to be located in the directory                            |
| 104649 |                                     | specified by the <b>folder</b> variable unless the pathname is absolute. The default shall                 |
| 104650 |                                     | be <b>nooutfolder</b> . See the <b>record</b> variable.                                                    |
| 104651 | <b>page</b>                         | Insert a <form-feed> after each message sent through the pipe created by the <b>pipe</b>                   |
| 104652 |                                     | command. The default shall be <b>nopage</b> .                                                              |
| 104653 | <b>prompt</b> = <i>string</i>       |                                                                                                            |
| 104654 |                                     | Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if <b>noprompt</b> is set, no   |
| 104655 |                                     | prompting shall occur. The default shall be to prompt with the string "? ".                                |
| 104656 | <b>quiet</b>                        | Refrain from writing the opening message and version when entering <i>mailx</i> . The                      |
| 104657 |                                     | default shall be <b>noquiet</b> .                                                                          |

- 104658        **record=***file*    Record all outgoing mail in the file with the pathname *file*. The default shall be  
104659        **norecord**. See also **outfolder** above.
- 104660        **save**            Enable saving of messages in the dead-letter file on interrupt or delivery error. See  
104661        the variable *DEAD* for the location of the dead-letter file. The default shall be **save**.
- 104662        **screen=***number*  
104663                Set the number of lines in a screenful of headers for the **headers** and **z** commands.  
104664                If **screen** is not specified, a value based on the terminal type identified by the  
104665                *TERM* environment variable, the window size, the baud rate, or some combination  
104666                of these shall be used. The default shall be **noscreen**.
- 104667        **sendwait**        Wait for the background mailer to finish before returning. The default shall be  
104668        **nosendwait**.
- 104669        **showto**         When the sender of the message was the user who is invoking *mailx*, write the  
104670        information from the **To** field instead of the **From** field in the header summary. The  
104671        default shall be **noshowto**.
- 104672        **sign=***string*    Set the variable inserted into the text of a message when the **~a** command escape is  
104673        given. The default shall be **nosign**. The character sequences '\t' and '\n' shall  
104674        be recognized in the variable as <tab> and <newline> characters, respectively. (See  
104675        also **~i** in [Command Escapes in mailx](#) (on page 3122).)
- 104676        **Sign=***string*    Set the variable inserted into the text of a message when the **~A** command escape is  
104677        given. The default shall be **noSign**. The character sequences '\t' and '\n' shall  
104678        be recognized in the variable as <tab> and <newline> characters, respectively.
- 104679        **toplines=***number*  
104680                Set the number of lines of the message to write with the **top** command. The default  
104681                shall be 5.

### 104682        **Commands in mailx**

104683        The following *mailx* commands shall be provided. In the following list, *header* refers to lines from  
104684        the message header, as shown in the OUTPUT FILES section. *Header field* refers to a line or set of  
104685        lines within the header that begins with one or more non-white-space characters immediately  
104686        followed by a <colon> and white space, and continuing up to and including a <newline> that  
104687        immediately precedes either the next line beginning with a non-white-space character or an  
104688        empty line. *Field name* refers to the portion of a header field prior to the first <colon>.

104689        For each of the commands listed below, the command can be entered as the abbreviation (those  
104690        characters in the Synopsis command word preceding the '['), the full command (all characters  
104691        shown for the command word, omitting the '[' and ']'), or any truncation of the full  
104692        command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the  
104693        Synopsis) can be entered as **ex**, **exi**, or **exit**.

104694        The arguments to commands can be quoted, using the following methods:

- 104695        • An argument can be enclosed between paired double-quotes (" ") or single-quotes (' ');  
104696        any white space, shell word expansion, or <backslash> characters within the quotes shall  
104697        be treated literally as part of the argument. A double-quote shall be treated literally within  
104698        single-quotes and *vice versa*. These special properties of the <quotation-mark> characters  
104699        shall occur only when they are paired at the beginning and end of the argument.
- 104700        • A <backslash> outside of the enclosing quotes shall be discarded and the following  
104701        character treated literally as part of the argument.



- 104702           • An unquoted <backslash> at the end of a command line shall be discarded and the next  
104703           line shall continue the command.

104704           Filenames, where expected, shall be subjected to the following transformations, in sequence:

- 104705           • If the filename begins with an unquoted <plus-sign>, and the **folder** variable is defined  
104706           (see the **folder** variable), the <plus-sign> shall be replaced by the value of the **folder**  
104707           variable followed by a <slash>. If the **folder** variable is unset or is set to null, the filename  
104708           shall be unchanged.
- 104709           • Shell word expansions shall be applied to the filename (see [Section 2.6](#), on page 2483). If  
104710           more than a single pathname results from this expansion and the command is expecting  
104711           one file, the effects are unspecified.

### 104712           **Declare Aliases**

104713           *Synopsis:*     a[lias] [alias [address...]]  
104714                   g[roup] [alias [address...]]

104715           Add the given addresses to the alias specified by *alias*. The names shall be substituted when  
104716           *alias* is used as a recipient address specified by the user in an outgoing message (that is, other  
104717           recipients addressed indirectly through the **reply** command shall not be substituted in this  
104718           manner). Mail address alias substitution shall apply only when the alias string is used as a full  
104719           address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution.  
104720           Recursive expansion of an alias group member can be prevented by prefixing it with an  
104721           unquoted <backslash>. If no arguments are given, write a listing of the current aliases to  
104722           standard output. If only an *alias* argument is given, write a listing of the specified alias to  
104723           standard output. These listings need not reflect the same order of addresses that were entered.

### 104724           **Declare Alternatives**

104725           *Synopsis:*     alt[ernates] name...

104726           Declare a list of alternative addresses for the address consisting of the user's login name. When  
104727           responding to a message, these alternative addresses shall be removed from the list of recipients.  
104728           The comparison of addresses shall be performed in a case-insensitive manner. With no  
104729           arguments, **alternates** shall write the current list of alternative addresses.

### 104730           **Change Current Directory**

104731           *Synopsis:*     cd [directory]  
104732                   ch[dir] [directory]

104733           Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

### 104734           **Copy Messages**

104735           *Synopsis:*     c[opy] [file]  
104736                   c[opy] [msglist] file  
104737                   C[opy] [msglist]

104738           Copy messages to the file named by the pathname *file* without marking the messages as saved.  
104739           Otherwise, it shall be equivalent to the **save** command.

104740           In the capitalized form, save the specified messages in a file whose name is derived from the  
104741           author of the message to be saved, without marking the messages as saved. Otherwise, it shall  
104742           be equivalent to the **Save** command.

104743 **Delete Messages**104744 *Synopsis:* d[*delete*] [*msglist*]

104745 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see  
 104746 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there  
 104747 are messages remaining after the **delete** command, the current message shall be written as  
 104748 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be  
 104749 written.

104750 **Discard Header Fields**

104751 *Synopsis:* di[*scard*] [*field-name...*]  
 104752 ig[*nore*] [*field-name...*]

104753 Suppress header fields with the specified field names when writing messages. Specified *field-*  
 104754 *name* arguments shall be added to the list of suppressed field names. Examples of field names to  
 104755 ignore are **status** and **cc**. The header fields shall be included when the message is saved. The  
 104756 **Print** and **Type** commands shall override this command. The comparison of field names shall be  
 104757 performed in a case-insensitive manner. If no arguments are specified, write a list of the  
 104758 currently suppressed field names to standard output; the listing need not reflect the same order  
 104759 of field names that were entered.

104760 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

104761 **Delete Messages and Display**

104762 *Synopsis:* dp [*msglist*]  
 104763 dt [*msglist*]

104764 Delete the specified messages as described for the **delete** command, except that the **autoprint**  
 104765 variable shall have no effect, and the current message shall be written only if it was set to a  
 104766 message after the last message deleted by the command. Otherwise, an informational message  
 104767 to the effect that there are no further messages in the mailbox shall be written, followed by the  
 104768 *mailx* prompt.

104769 **Echo a String**

104770 *Synopsis:* ec[*ho*] *string* ...

104771 Echo the given strings, equivalent to the shell *echo* utility.

104772 **Edit Messages**

104773 *Synopsis:* e[*dit*] [*msglist*]

104774 Edit the given messages. The messages shall be placed in a temporary file and the utility named  
 104775 by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is  
 104776 unspecified.

104777 The **edit** command does not modify the contents of those messages in the mailbox.

104778 **Exit**

104779 *Synopsis:*    ex[it]  
104780                   x[it]

104781 Exit from *mailx* without performing automatic message moving, or any other management tasks.  
104782 See also **quit**.

104783 **Change Folder**

104784 *Synopsis:*    fi[le] [*file*]  
104785                   fold[er] [*file*]

104786 If no argument is given, write the name and status of the current mailbox. Otherwise, close the  
104787 current file of messages after performing actions as specified for the **quit** command (except for  
104788 terminating *mailx*) and then read in the file named by the pathname *file*. The behavior is  
104789 unspecified if *file* is not a valid **mbox**.

104790 Several unquoted special characters shall be recognized when used as *file* names, with the  
104791 following substitutions:

- 104792 %           The system mailbox for the invoking user.  
104793 %*user*    The system mailbox for *user*.  
104794 #           The previous file.  
104795 &          The current **mbox**.  
104796 +*file*    The named file in the **folder** directory. (See the **folder** variable.)  
104797 The default file shall be the current mailbox.

104798 **Display List of Folders**

104799 *Synopsis:*    folders

104800 Write the names of the files in the directory set by the **folder** variable. The command specified  
104801 by the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES  
104802 section).

104803 **Follow Up Specified Messages**

104804 *Synopsis:*    fo[llowup] [*message*]  
104805                   F[ollowup] [*msglist*]

104806 The **followup** and **Followup** commands shall be equivalent to **reply** and **Reply**, respectively,  
104807 except that:

- 104808       • They shall ignore the **record** variable.  
104809       • The **followup** command shall record the response in a file whose name is derived from the  
104810       author of the *message*.  
104811       • The **Followup** command shall record the response in a file whose name is derived from the  
104812       author of the first message in the *msglist*.

104813 See also the **save** and **copy** commands and **outfolder**.

104814 **Display Header Summary for Specified Messages**104815 *Synopsis:* f[rom] [msglist]

104816 Write the header summary for the specified messages.

104817 **Display Header Summaries**104818 *Synopsis:* h[eaders] [message]

104819 Write the page of header summaries that includes the message specified. If the *message*  
 104820 argument is not specified, the current message shall not change. However, if the *message*  
 104821 argument is specified, the current message shall become the message that appears at the top of  
 104822 the page of header summaries that includes the message specified. The **screen** variable sets the  
 104823 number of header summaries per page. See also the **z** command.

104824 **Help**104825 *Synopsis:* hel[p]

104826 ?

104827 Write a summary of commands.

104828 **Hold Messages**104829 *Synopsis:* ho[ld] [msglist]

104830 pre[serve] [msglist]

104831 Allowed only in the system mailbox. Mark the messages in *msglist* to be preserved, as if the **hold**  
 104832 variable were set, upon normal termination or when the folder is changed. This shall override  
 104833 any commands that might previously have marked the messages to be deleted, and only the  
 104834 **delete**, **dp**, or **dt**, as well as the **mbox** and **touch** commands, shall remove the *preserve* mark of a  
 104835 message.

104836 **Execute Commands Conditionally**104837 *Synopsis:* i[f] s|r104838 *mail-commands*

104839 el[se]

104840 *mail-commands*

104841 en[dif]

104842 Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an  
 104843 **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be  
 104844 executed only in Receive Mode.

104845 **List Available Commands**104846 *Synopsis:* l[ist]

104847 Write a list of all commands available. No explanation shall be given.

104848 **Mail a Message**104849 *Synopsis:* m[ail] address...

104850 Mail a message to the specified addresses or aliases.

104851 **Direct Messages to mbox**104852 *Synopsis:* mb[ox] [msglist]

104853 Allowed only in the system mailbox. Arrange for the given messages to end up in the secondary  
104854 mailbox, overriding a possibly set **hold** variable, upon normal termination or when the folder is  
104855 changed. Overrides a former **hold** or **preserve** request. See *MBOX* in the ENVIRONMENT  
104856 VARIABLES section. See also the **exit** and **quit** commands.

104857 **Process Next Specified Message**104858 *Synopsis:* n[ext] [message]

104859 If the current message has not been written (for example, by the **print** command) since *mailx*  
104860 started or since any other message was the current message, behave as if the **print** command  
104861 was entered. Otherwise, if there is an undeleted message after the current message, make it the  
104862 current message and behave as if the **print** command was entered. Otherwise, an informational  
104863 message to the effect that there are no further messages in the mailbox shall be written, followed  
104864 by the *mailx* prompt. Should the current message location be the result of an immediately  
104865 preceding **hold**, **mbox**, **preserve**, or **touch** command, **next** shall act as if the current message has  
104866 already been written.

104867 **Pipe Message**104868 *Synopsis:* pi[pe] [[msglist] command]  
104869 | [[msglist] command]

104870 Pipe the messages through the given *command* by invoking the command interpreter specified  
104871 by *SHELL* with three arguments: "-c", "--", and *command*. (See also *sh -c*.) The application  
104872 shall ensure that the command is given as a single argument. Quoting, described previously, can  
104873 be used to accomplish this. If no arguments are given, the current message shall be piped  
104874 through the command specified by the value of the **cmd** variable. If the **page** variable is set, a  
104875 <form-feed> shall be inserted after each message.

104876 **Display Message with Header**104877 *Synopsis:* P[rint] [msglist]  
104878 T[ype] [msglist]

104879 Write the specified messages, including all header fields, to standard output. This command  
104880 shall override suppression of header fields by the **discard**, **ignore**, and **retain** commands. If **crt** is  
104881 set, the output shall be paginated as described for the *PAGER* variable.

104882      **Display Message**

104883      *Synopsis:*    p[rint] [msglist]  
 104884                    t[type] [msglist]

104885      Write the specified messages to standard output. If **crt** is set, the output shall be paginated as  
 104886      described for the *PAGER* variable.

104887      **Quit**

104888      *Synopsis:*    q[uit]  
 104889                    end-of-file

104890      Terminate *mailx* normally, performing automatic message moving as specified in the description  
 104891      of the variable **hold**, deleting messages that have been explicitly saved (unless **keepsave** is set),  
 104892      discarding messages that have been deleted, and saving all remaining messages in the mailbox.

104893      **Reply to a Message or a Message List**

104894      *Synopsis:*    r[eply] [message]  
 104895                    r[espond] [message]  
 104896                    R[eply] [msglist]  
 104897                    R[espond] [msglist]

104898      Mail a reply message to one or more addresses taken from certain header fields in the specified  
 104899      message or message list. If the **flipr** variable is unset, these commands shall behave as described  
 104900      below. If the **flipr** variable is set, commands in the lowercase form shall behave as described  
 104901      below for commands in the capitalized form, and *vice versa*; the synopsis forms shown above  
 104902      shall also be swapped accordingly.

104903      The recipients of the reply message shall be determined by first constructing an initial list of  
 104904      recipients and then modifying it to form the list that is in effect when *mailx* enters input mode.

104905      In the capitalized form, the initial list of recipients shall be taken from the header of each  
 104906      message in the *msglist* as follows:

- 104907            • If the header contains a **Reply-To** field, the address or addresses in that field shall be  
 104908            added to the list.
- 104909            • Otherwise, the address or addresses in the **From** field of the header shall be added to the  
 104910            list.

104911      In the lowercase form, the initial list of recipients shall be taken from the header of the *message* as  
 104912      follows:

- 104913            • If the header does not contain a **Reply-To** field, all of the addresses in the **From**, **To**, and **Cc**  
 104914            fields shall be included in the list.
- 104915            • Otherwise, it is implementation-defined whether all of the addresses in the **Reply-To**, **To**,  
 104916            and **Cc** fields are included in the list or only the address or addresses in the **Reply-To** field.

104917      The initial list of recipients shall be marked for placement in the header fields of the reply  
 104918      message as follows. Recipient addresses taken from a **From** or **Reply-To** header field shall be  
 104919      marked for placement in the **To** field of the reply message. Recipient addresses taken from a **Cc**  
 104920      header field shall be marked for placement in the **Cc** field of the reply message. Recipient  
 104921      addresses taken from a **To** header field shall be marked for placement in either the **To** or the **Cc**  
 104922      field of the reply message. Implementations shall provide a way to place them in the **To** field.  
 104923      Implementations may, but need not, provide an implementation-defined way to place them in  
 104924      the **Cc** field.

104925 The modifications applied to the initial list of recipients shall be as follows:

- 104926 • If the **metoo** variable is unset, addresses consisting of the login name of the user and any
- 104927 alternative addresses declared using the **alternates** command shall be removed from the
- 104928 list.
- 104929 • The set of recipients marked for placement in the **To** header field of the reply message shall
- 104930 have duplicates within that set removed.
- 104931 • The set of recipients marked for placement in the **Cc** header field of the reply message shall
- 104932 have duplicates within that set removed and may have recipients that are also marked for
- 104933 placement in the **To** field removed.

104934 The values for the **To** and **Cc** header fields of the reply message shall be constructed from the

104935 addresses in the modified list of recipients that are marked for placement in each of those fields.

104936 The value for the **Subject** header field of the reply message shall be formed from the value of the

104937 **Subject** header field of the *message* or the first message in *msglist* by prefixing it with

104938 **Re:**<space>, unless it already begins with that string.

104939 The values of the **To**, **Cc**, and **Subject** header fields set as described above can be modified by

104940 the user after *mailx* enters input mode through the use of the **~t**, **~c**, **~s**, and **~h** command escapes.

104941 If **record** is set to a pathname, the response shall be saved at the end of that file.

#### 104942 **Retain Header Fields**

104943 *Synopsis:*     **ret**[ain] [*field-name...*]

104944 Retain header fields with the specified field names when writing messages. This command shall

104945 override all **discard** and **ignore** commands. The comparison of field names shall be performed

104946 in a case-insensitive manner. If no arguments are specified, write a list of the currently retained

104947 field names to standard output; the listing need not reflect the same order of field names that

104948 were entered.

#### 104949 **Save Messages**

104950 *Synopsis:*     **s**[ave] [*file*]

104951               **s**[ave] [*msglist*] *file*

104952               **S**[ave] [*msglist*]

104953 Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file*

104954 argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be

104955 appended to the file. The message shall be put in the state *saved*, and shall behave as specified in

104956 the description of the *saved* state when the current mailbox is exited by the **quit** or **file**

104957 command.

104958 In the capitalized form, save the specified messages in a file whose name is derived from the

104959 author of the first message. The name of the file shall be taken to be the author's name with all

104960 network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and

104961 **outfolder** variable.

**104962 Set Variables**

104963 *Synopsis:*    `se[t] [name=[string]] ... [name=number ...] [noname ...]`

104964 Define one or more variables called *name*. The variable can be given a null, string, or numeric  
104965 value. Quoting and <backslash>-escapes can occur anywhere in *string*, as described previously,  
104966 as if the *string* portion of the argument were the entire argument. The forms *name* and *name=*  
104967 shall be equivalent to *name=""* for variables that take string values. The **set** command without  
104968 arguments shall write a list of all defined variables and their values. The **no name** form shall be  
104969 equivalent to **unset name**.

**104970 Invoke a Shell**

104971 *Synopsis:*    `sh[ell]`

104972 Invoke an interactive command interpreter (see also *SHELL*).

**104973 Display Message Size**

104974 *Synopsis:*    `si[ze] [msglist]`

104975 Write the size in bytes of each of the specified messages.

**104976 Read mailx Commands From a File**

104977 *Synopsis:*    `so[urce] file`

104978 Read and execute commands from the file named by the pathname *file* and return to command  
104979 mode.

**104980 Display Beginning of Messages**

104981 *Synopsis:*    `to[p] [msglist]`

104982 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken  
104983 as the number of lines to write. The default shall be 5.

**104984 Touch Messages**

104985 *Synopsis:*    `tou[ch] [msglist]`

104986 Allowed only in the system mailbox. Touch the specified messages. Unless overridden by the  
104987 **hold** variable, any message in *msglist* that is not specifically deleted nor saved in a file shall be  
104988 placed in the **mbox** upon normal termination or when the folder is changed. Overrides a former  
104989 **hold** or **preserve** request.

**104990 Delete Aliases**

104991 *Synopsis:*    `una[lias] [alias] ...`

104992 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.



**104993 Undelete Messages**

104994 *Synopsis:* u[ndelete] [*msglist*]

104995 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last  
104996 message of those restored shall be written. If *msglist* is not specified, the message shall be  
104997 selected as follows:

- 104998 • If there are any deleted messages that follow the current message, the first of these shall be  
104999 chosen.
- 105000 • Otherwise, the last deleted message that also precedes the current message shall be chosen.

**105001 Unset Variables**

105002 *Synopsis:* uns[et] *name*...

105003 Cause the specified variables to be erased.

**105004 Edit Message with Full-Screen Editor**

105005 *Synopsis:* v[isual] [*msglist*]

105006 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,  
105007 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The  
105008 default editor shall be *vi*.

105009 The **visual** command does not modify the contents of those messages in the mailbox.

**105010 Write Messages to a File**

105011 *Synopsis:* w[rite] [*msglist*] *file*

105012 Write the given messages to the file specified by the pathname *file*, minus the message header.  
105013 Otherwise, it shall be equivalent to the **save** command.

**105014 Scroll Header Display**

105015 *Synopsis:* z[+|-]

105016 Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if  
105017 '-' is specified) one screenful. The number of header summaries written shall be set by the  
105018 **screen** variable.

**105019 Invoke Shell Command**

105020 *Synopsis:* !*command*

105021 Invoke the command interpreter specified by *SHELL* with three arguments: "-c", "--", and  
105022 *command*. (See also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in  
105023 *command* shall be replaced with the command executed by the previous ! command or ~!  
105024 command escape.

105025 **Null Command**105026 *Synopsis:* # *comment*105027 This null command (comment) shall be ignored by *mailx*.105028 **Display Current Message Number**105029 *Synopsis:* =

105030 Write the current message number.

105031 **Command Escapes in mailx**

105032 The following commands can be entered only from input mode, by beginning a line with the  
 105033 escape character (by default, <tilde> ('~')). See the **escape** variable description for changing  
 105034 this special character. The format for the commands shall be:

105035 &lt;escape-character&gt;&lt;command-char&gt;&lt;separator&gt; [&lt;arguments&gt;]

105036 where the &lt;separator&gt; can be zero or more &lt;blank&gt; characters.

105037 In the following descriptions, the application shall ensure that the argument *command* (but not  
 105038 *mailx-command*) is a shell command string. Any string acceptable to the command interpreter  
 105039 specified by the *SHELL* variable when it is invoked as *SHELL -c command\_string* shall be valid.  
 105040 The command can be presented as multiple arguments (that is, quoting is not required).

105041 Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send  
 105042 Mode and produce unspecified results.

105043 **~! command** Invoke the command interpreter specified by *SHELL* with three arguments: "-c",  
 105044 "--", and *command*; and then return to input mode. If the **bang** variable is set,  
 105045 each unescaped occurrence of '!' in *command* shall be replaced with the command  
 105046 executed by the previous ! command or ~! command escape.

105047 **~.** Simulate end-of-file (terminate message input).

105048 **~: mailx-command, ~\_ mailx-command**  
 105049 Perform the command-level request.

105050 **~?** Write a summary of command escapes.105051 **~A** This shall be equivalent to **~i Sign**.105052 **~a** This shall be equivalent to **~i sign**.105053 **~b name...** Add the *names* to the blind carbon copy (**Bcc**) list.105054 **~c name...** Add the *names* to the carbon copy (**Cc**) list.105055 **~d** Read in the dead-letter file. See *DEAD* for a description of this file.

105056 **~e** Invoke the editor, as specified by the *EDITOR* environment variable, on the partial  
 105057 message.

105058 **~f [msglist]** Forward the specified messages. The specified messages, including their headers,  
 105059 shall be inserted into the current message without alteration. The header fields  
 105060 included in each header shall be affected by the **discard**, **ignore**, and **retain**  
 105061 commands.

- 105062        **~F** [*msglist*] This shall be the equivalent of the **~f** command escape, except that all header fields  
105063 shall be included in the message headers, regardless of previous **discard**, **ignore**,  
105064 and **retain** commands.
- 105065        **~h**            If standard input is a terminal, prompt for values for the **Subject**, **To**, **Cc**, and **Bcc**  
105066 header fields. Other implementation-defined header fields may also be presented  
105067 for editing. If the header field is written with an initial value, it can be edited as if it  
105068 had just been typed.
- 105069        **~i** *string*      Insert the value of the named variable, followed by a <newline>, into the text of  
105070 the message. If the string is unset or null, the message shall not be changed.
- 105071        **~m** [*msglist*] Insert the specified messages, including their headers, into the current message,  
105072 prefixing non-empty lines with the string in the **indentprefix** variable. The header  
105073 fields included in each header shall be affected by the **discard**, **ignore**, and **retain**  
105074 commands.
- 105075        **~M** [*msglist*] This shall be the equivalent of the **~m** command escape, except that all header  
105076 fields shall be included in the message headers, regardless of previous **discard**,  
105077 **ignore**, and **retain** commands.
- 105078        **~p**            Write the message being entered. If the message is longer than **crt** lines (see  
105079 [Internal Variables in mailx](#), on page 3109), the output shall be paginated as  
105080 described for the **PAGER** variable.
- 105081        **~q**            Quit (see the **quit** command) from input mode by simulating an interrupt. If the  
105082 body of the message is not empty, the partial message shall be saved in the dead-  
105083 letter file. See **DEAD** for a description of this file.
- 105084        **~r** *file*, **~<** *file*, **~r !command**, **~< !command**  
105085                    Read in the file specified by the pathname *file*. If the argument begins with an  
105086 <exclamation-mark> ('!'), the rest of the string shall be taken as an arbitrary  
105087 system command; the command interpreter specified by **SHELL** shall be invoked  
105088 with three arguments: "-c", "--", and *command*. The standard output of  
105089 *command* shall be inserted into the message.
- 105090        **~s** *string*      Set the value for the **Subject** header field to *string*.
- 105091        **~t** *name...*    Add the given *names* to the **To** list.
- 105092        **~v**            Invoke the full-screen editor, as specified by the **VISUAL** environment variable, on  
105093 the partial message.
- 105094        **~w** *file*        Write the partial message, without the header, onto the file named by the  
105095 pathname *file*. The file shall be created or the message shall be appended to it if  
105096 the file exists.
- 105097        **~x**            Exit as with **~q**, except the message shall not be saved in the dead-letter file.
- 105098        **~|** *command*    Pipe the body of the message through the given *command* by invoking the  
105099 command interpreter specified by **SHELL** with three arguments: "-c", "--", and  
105100 *command*. If the *command* returns a successful exit status, the standard output of  
105101 the command shall replace the message. Otherwise, the message shall remain  
105102 unchanged. If the *command* fails, an error message giving the exit status shall be  
105103 written.

105104 **EXIT STATUS**

105105 UP When the `-e` option is specified, the following exit values are returned:

105106 0 Mail was found.

105107 >0 Mail was not found or an error occurred.

105108 Otherwise, the following exit values are returned:

105109 0 Successful completion; note that this status implies that any messages that *mailx* was  
 105110 instructed to send were all successfully either *sent* or discarded (see `-E`), but it gives no  
 105111 assurances that any of them were actually *delivered*.

105112 >0 An error occurred.

105113 **CONSEQUENCES OF ERRORS**

105114 UP When in `input mode (Receive Mode)` or Send Mode:

105115 • If an error is encountered processing an input line beginning with a `<tilde>` (`'~'`)  
 105116 UP character, (see [Command Escapes in mailx](#), on page 3122), a diagnostic message shall be  
 105117 written to standard error, and the message being composed may be modified, but this  
 105118 condition shall not prevent the message from being sent.

105119 • Other errors shall prevent the sending of the message.

105120 UP When in command mode:

105121 • Default.

105122 **APPLICATION USAGE**

105123 Delivery of messages to remote systems requires the existence of communication paths to such  
 105124 systems. These need not exist.

105125 Input lines are limited to `{LINE_MAX}` bytes, but mailers between systems may impose more  
 105126 severe line-length restrictions. This volume of POSIX.1-2024 does not place any restrictions on  
 105127 the length of messages handled by *mailx*, and for delivery of local messages the only limitations  
 105128 should be the normal problems of available disk space for the target mail file. When sending  
 105129 messages to external machines, applications are advised to limit messages to less than 100 000  
 105130 bytes because some mail gateways impose message-length restrictions.

105131 The format of the system mailbox is intentionally unspecified. Not all systems implement  
 105132 system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some  
 105133 system mailboxes may be multiple files, others records in a database. The internal format of the  
 105134 messages themselves is specified with the historical format from Version 7, but only after the  
 105135 messages have been saved in some file other than the system mailbox. This was done so that  
 105136 many historical applications expecting text-file mailboxes are not broken.

105137 Some new formats for messages can be expected in the future, probably including binary data,  
 105138 bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from  
 105139 handling such messages, but it must store them as text files in secondary mailboxes (unless some  
 105140 extension, such as a variable or command line option, is used to change the stored format). Its  
 105141 method of doing so is implementation-defined and might include translating the data into text  
 105142 file-compatible or readable form or omitting certain portions of the message from the stored  
 105143 output.

105144 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**  
 105145 command discards all header fields except those explicitly retained. The **discard** command

105146 keeps all header fields except those explicitly discarded. If field names exist on the retained field  
 105147 names list, **discard** and **ignore** commands are ignored.

#### 105148 EXAMPLES

105149 None.

#### 105150 RATIONALE

105151 The standard developers felt strongly that a method for applications to send messages to specific  
 105152 users was necessary. The obvious example is a batch utility, running non-interactively, that  
 105153 wishes to communicate errors or results to a user. However, the actual format, delivery  
 105154 mechanism, and method of reading the message are clearly beyond the scope of this volume of  
 105155 POSIX.1-2024.

105156 The intent of this command is to provide a simple, portable interface for sending messages non-  
 105157 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that  
 105158 implementations explicitly denote the sender and recipient in the body of the delivered message.  
 105159 Further specification of formats for either the message envelope or the message itself were  
 105160 deliberately not made, as the industry is in the midst of changing from the current standards to a  
 105161 more internationalized standard and it is probably incorrect, at this time, to require either one.

105162 Implementations are encouraged to conform to the various delivery mechanisms described in  
 105163 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request  
 105164 for Comment (RFC) documents RFC 819, RFC 920, RFC 921, RFC 1123, and RFC 5322.

105165 Many historical systems modified each body line that started with **From** by prefixing the 'F'  
 105166 with '>'. It is unnecessary, but allowed, to do that when the string does not follow a blank line  
 105167 because it cannot be confused with the next header.

105168 The **edit** and **visual** commands merely edit the specified messages in a temporary file. They do  
 105169 not modify the contents of those messages in the mailbox; such a capability could be added as an  
 105170 extension, such as by using different command names.

105171 The restriction on the value for a **Subject** header field being {LINE\_MAX}-10 bytes is based on  
 105172 the historical format that consumes 10 bytes for **Subject:** and the trailing <newline>. Many  
 105173 historical mailers that a message may encounter on other systems are not able to handle lines  
 105174 that long, however.

105175 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient  
 105176 justification to exclude this utility from this volume of POSIX.1-2024. It is also arguable that it is,  
 105177 in fact, testable, but that the tests themselves are not portable.

105178 When *mailx* is being used by an application that wishes to receive the results as if none of the  
 105179 User Portability Utilities option features were supported, the *DEAD* environment variable must  
 105180 be set to */dev/null*. Otherwise, it may be subject to the file creations described in *mailx*  
 105181 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to  
 105182 */dev/null*, historical versions of *mailx* and *Mail* read initialization commands from a file before  
 105183 processing begins. Since the initialization that a user specifies could alter the contents of  
 105184 messages an application is trying to send, such applications must set *MAILRC* to */dev/null*.

105185 The description of *LC\_TIME* uses “may affect” because many historical implementations do not  
 105186 or cannot manipulate the date and time strings in the incoming mail headers. Some header fields  
 105187 found in incoming mail do not have enough information to determine the timezone in which the  
 105188 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal  
 105189 form that then is parsed by routines like *strftime()* that can take *LC\_TIME* settings into account.  
 105190 Changing all these times to a user-specified format is allowed, but not required.

105191 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System

105192 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by  
 105193 declaring that *cat* is the paginator, would not meet with the intended meaning of this  
 105194 description. However, any “portable user” would have to set *PAGER* explicitly to get his or her  
 105195 preferred paginator on all systems. The paginator choice was made partially unspecified, unlike  
 105196 the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common  
 105197 theme of user input, whereas editors differ dramatically.

105198 Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to  
 105199 be format details and were omitted.

105200 A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them  
 105201 were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an  
 105202 appropriate marketing distinction between systems.

105203 In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file*  
 105204 option-argument to **-f**. By making *file* an operand, the guidelines are satisfied and users remain  
 105205 portable. However, it does force implementations to support usage such as:

```
105206 mailx -fin mymail.box
```

105207 The **no name** method of unsetting variables is not present in all historical systems, but it is in  
 105208 System V and provides a logical set of commands corresponding to the format of the display of  
 105209 options from the *mailx set* command without arguments.

105210 The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the  
 105211 same feature. They are synonyms in this volume of POSIX.1-2024.

105212 The *mailx echo* command was not documented in the BSD version and has been omitted here  
 105213 because it is not obviously useful for interactive users.

105214 The default prompt on the System V *mailx* is a <question-mark>, on BSD *Mail* an <ampersand>.  
 105215 Since this volume of POSIX.1-2024 chose the *mailx* name, it kept the System V default, assuming  
 105216 that BSD users would not have difficulty with this minor incompatibility (that they can  
 105217 override).

105218 The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again,  
 105219 since this volume of POSIX.1-2024 chose the *mailx* name, it kept the System V default, but allows  
 105220 the SunOS user to achieve the desired results using **flipr**, an internal variable in System V *mailx*,  
 105221 although it has not been documented in the SVID.

105222 The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command  
 105223 escapes were adopted from 4.3 BSD *Mail*.

105224 The **version** command was not included because no sufficiently general specification of the  
 105225 version information could be devised that would still be useful to a portable user. This  
 105226 command name should be used by suppliers who wish to provide version information about the  
 105227 *mailx* command.

105228 The “implementation-specific (unspecified) system start-up file” historically has been named  
 105229 **/etc/mailx.rc**, but this specific name and location are not required.

105230 The intent of the wording for the **next** command is that if any command has already displayed  
 105231 the current message it should display a following message, but, otherwise, it should display the  
 105232 current message. Consider the command sequence:

```
105233 next 3
105234 delete 3
105235 next
```

105236 where the **autoprint** option was not set. The normative text specifies that the second **next**  
 105237 command should display a message following the third message, because even though the  
 105238 current message has not been displayed since it was set by the **delete** command, it has been  
 105239 displayed since the current message was anything other than message number 3. This does not  
 105240 always match historical practice in some implementations, where the command file address  
 105241 followed by **next** (or the default command) would skip the message for which the user had  
 105242 searched.

#### 105243 FUTURE DIRECTIONS

105244 If this utility is directed to create a new directory entry that contains any bytes that have the  
 105245 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 105246 error. A future version of this standard may require implementations to treat this as an error.

#### 105247 SEE ALSO

105248 [Chapter 2](#) (on page 2472), *ed*, *ls*, *more*, *vi*  
 105249 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 105250 CHANGE HISTORY

105251 First released in Issue 2.

#### 105252 Issue 5

105253 The description of the *EDITOR* environment variable is changed to indicate that *ed* is the default  
 105254 editor if this variable is not set. In previous issues, this default was not stated explicitly at this  
 105255 point but was implied further down in the text.

105256 The FUTURE DIRECTIONS section is added.

#### 105257 Issue 6

105258 The following new requirements on POSIX implementations derive from alignment with the  
 105259 Single UNIX Specification:

- 105260 • The **-F** option is added.
- 105261 • The **allnet**, **debug**, and **sendwait** internal variables are added.
- 105262 • The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.

105263 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “*HOME*  
 105264 directory” is replaced by “directory referred to by the *HOME* environment variable”.

105265 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which includes various  
 105266 clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993  
 105267 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,  
 105268 #103, #106, #108, #114, #115, #122, and #129.

105269 The normative text is reworded to avoid use of the term “must” for application requirements.

105270 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

105271 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/32 is applied, applying a change to the  
 105272 EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was  
 105273 overlooked in the first version of this standard.

105274 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/17 is applied, updating the EXTENDED  
 105275 DESCRIPTION (Internal Variables in *mailx*). The system start-up file is changed from  
 105276 “implementation-defined” to “unspecified” for consistency with other text in the EXTENDED  
 105277 DESCRIPTION.

105278 **Issue 7**

105279 Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC\_TIME*  
105280 environment variable.

105281 Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the **next**  
105282 command.

105283 SD5-XCU-ERN-69 is applied.

105284 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105285 Shading to indicate support for the User Portability Utilities option is added.

105286 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0120 [855] and XCU/TC2-2008/0121  
105287 [619] are applied.

105288 **Issue 8**

105289 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
105290 filenames containing any bytes that have the encoded value of a <newline> character.

105291 Austin Group Defect 956 is applied, clarifying how the *PAGER* environment variable and the **cr**  
105292 internal variable affect pagination.

105293 Austin Group Defects 990 and 991 are applied, changing the description of the **mbox** command.

105294 Austin Group Defect 999 is applied, adding **Save** to the list of commands that are invalid in a  
105295 start-up file.

105296 Austin Group Defect 1034 is applied, clarifying that `~.` can be used to terminate input mode  
105297 when **ignoreeof** is set.

105298 Austin Group Defect 1109 is applied, changing the description of the **bang** internal variable.

105299 Austin Group Defect 1113 is applied, changing the description of the *read* state.

105300 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

105301 Austin Group Defect 1176 is applied, changing ``option-argument'' to ``operand''.

105302 Austin Group Defect 1306 is applied, changing the description of the **folder** internal variable.

105303 Austin Group Defect 1367 is applied, adding the **-E** option.

105304 Austin Group Defect 1401 is applied, changing the requirements for the **reply**, **Reply**, **followup**,  
105305 and **Followup** commands.

105306 Austin Group Defect 1405 is applied, changing the terminology related to mail messages to  
105307 match IETF RFC 5322.

105308 Austin Group Defect 1408 is applied, changing the description of Send Mode.

105309 Austin Group Defect 1507 is applied, changing the EXIT STATUS section.

105310 Austin Group Defect 1528 is applied, adding a "--" argument to be passed between "-c" and  
105311 *command* when executing shell commands.

105312 Austin Group Defect 1634 is applied, clarifying handling of the system mailbox.

105313 Austin Group Defect 1685 is applied, updating RFC references.

105314 Austin Group Defect 1725 is applied, clarifying that the default for the **screen** internal variable is  
105315 **noscreen**.

105316 Austin Group Defect 1743 is applied, changing the descriptions of the **metoo** internal variable



105317

and the **alternates** command.

105318

Austin Group Defect 1747 is applied, changing the description of the **alias** command.

105319 **NAME**105320 make — maintain, update, and regenerate files (**DEVELOPMENT**)105321 **SYNOPSIS**

```
105322 SD make [-einpqrst] [-f makefile]... [-j maxjobs] [-k|-S]
105323      [macro[:[:]]=value...] [target_name...]
```

105324 **DESCRIPTION**

105325 The *make* utility shall update files that are derived from other files. A typical case is one where  
 105326 object files are derived from the corresponding source files. The *make* utility examines time  
 105327 relationships and shall update those derived files (called targets) that have modified times  
 105328 earlier than the modified times of the files (called prerequisites) from which they are derived. A  
 105329 description file (makefile) contains a description of the relationships between files, and the  
 105330 commands that need to be executed to update the targets to reflect changes in their  
 105331 prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and  
 105332 optional commands to be executed when a prerequisite is newer than the target. There are two  
 105333 kinds of rule:

- 105334 1. *Inference rules*, which have one target name with at least one <period> ('.') and no  
 105335 <slash> ('/')
- 105336 2. *Target rules*, which can have more than one target name

105337 In addition, *make* shall have a collection of built-in macros and inference rules that infer  
 105338 prerequisite relationships to simplify maintenance of programs.

105339 To receive exactly the behavior described in this section, a portable makefile shall:

- 105340 • Include the special target **.POSIX**
- 105341 • Omit any special target reserved for implementations (a leading period followed by  
 105342 uppercase letters) that has not been specified by this section

105343 The behavior of *make* is unspecified if either or both of these conditions are not met.

105344 **OPTIONS**

105345 The *make* utility shall conform to XBD [Section 12.2](#) (on page 215), except for Guideline 9.

105346 The following options shall be supported:

- 105347 **-e** Cause environment variables, including those with null values, to override macro  
 105348 assignments within makefiles.
- 105349 **-f *makefile*** Specify a different makefile. The argument *makefile* is a pathname of a description  
 105350 file, which is also referred to as the *makefile*. A pathname of '-' shall denote the  
 105351 standard input. There can be multiple instances of this option, and they shall be  
 105352 processed in the order specified. The effect of specifying the same option-argument  
 105353 more than once is unspecified.
- 105354 **-i** Ignore error codes returned by invoked commands. This mode shall be the same as  
 105355 if the special target **.IGNORE** were specified without prerequisites.
- 105356 **-j *maxjobs*** Set the maximum number of targets that can be updated concurrently. If this  
 105357 option is specified multiple times, the last value of *maxjobs* specified shall take  
 105358 precedence. If this option is not specified, or if *maxjobs* is 1, only one target shall be  
 105359 updated at a time (no parallelization). If the value of *maxjobs* is non-positive, the  
 105360 behavior is unspecified. When *maxjobs* is greater than 1, *make* shall create a pool of  
 105361 up to *maxjobs* - 1 tokens. (Note that implementations are not required to create a  
 105362 pool of exactly *maxjobs* - 1 tokens. For example, an implementation could limit the

- 105363 pool size based on the number of processors available.) If the size of the token pool  
 105364 would be 0, *make* need not implement a token pool.
- 105365 When all of the following are true:
- 105366 • There is a target with commands that is not up-to-date
  - 105367 • The target's prerequisites (if any) are up-to-date
  - 105368 • *make* is not waiting to bring the target up-to-date (see **.WAIT**)
  - 105369 • *make* is currently bringing a different target with commands up-to-date
  - 105370 • *make* is not currently bringing *maxjobs* targets up-to-date in parallel
  - 105371 • The special target **.NOTPARALLEL** is not specified
  - 105372 • The token pool is not empty
- 105373 then *make* may attempt to remove one token from the pool. If a token is  
 105374 successfully removed, it shall attempt to bring this target up-to-date in parallel,  
 105375 and after this processing completes shall return the token to the pool. When *make* is  
 105376 bringing a target without commands up-to-date, it need not remove a token from  
 105377 the pool.
- 105378 If a rule invokes a sub-*make* either via the **MAKE** macro or via a command line that  
 105379 begins with ' + ', the sub-*make* is the same implementation as the *make* that invoked  
 105380 the sub-*make*, and the **-j** option is passed to the sub-*make* via the **MAKEFLAGS**  
 105381 environment variable with the same *maxjobs* value and is not overridden by a  
 105382 *maxjobs* value from another source (even if it has the same value), the sub-*make*  
 105383 shall use the same token pool as its invoking *make* rather than create a new token  
 105384 pool. Otherwise, it is unspecified whether the sub-*make* uses the same token pool  
 105385 as its invoking *make* or creates a new token pool. If a rule executes multiple sub-  
 105386 *make* processes asynchronously the behavior is unspecified.
- 105387 **-k** Continue to update other targets that do not depend on the current target if a non-  
 105388 ignored error occurs while executing the commands to bring a target up-to-date.
- 105389 **-n** Write commands that would be executed on standard output, but do not execute  
 105390 them. However, lines with a <plus-sign> ('+') prefix, lines that expand the **MAKE**  
 105391 macro, and lines being processed in order to create an include file or to bring it up-  
 105392 to-date (see **Include Lines** in the EXTENDED DESCRIPTION section) shall be  
 105393 executed. In this mode, lines with a <commercial-at> ('@') character prefix shall  
 105394 be written to standard output.
- 105395 **-p** Write to standard output the complete set of macro definitions and target  
 105396 descriptions. The output format is unspecified.
- 105397 **-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit  
 105398 value of 1. Targets shall not be updated if this option is specified. However, a  
 105399 makefile command line (associated with the targets) with a <plus-sign> ('+')  
 105400 prefix shall be executed and it is unspecified whether command lines that do not  
 105401 have a <plus-sign> prefix and either expand the **MAKE** macro or are being  
 105402 processed in order to create an include file or to bring it up-to-date (see **Include**  
 105403 **Lines** in the EXTENDED DESCRIPTION section) are executed.
- 105404 **-r** Clear the suffix list and do not use the built-in rules.

- 105405        **-S**            Terminate *make* if an error occurs while executing the commands to bring a target  
105406                   up-to-date. This shall be the default and the opposite of **-k**.
- 105407        **-s**            Do not write makefile *execution lines* (see [Makefile Execution](#), on page 3136) or  
105408                   touch messages (see **-t**) to standard output before executing. This mode shall be  
105409                   the same as if the special target **.SILENT** were specified without prerequisites.
- 105410        **-t**            Update the modification time of each target as though a *touch target* had been  
105411                   executed. Targets that have prerequisites but no commands (see [Target Rules](#), on  
105412                   page 3137), or that are already up-to-date, shall not be touched in this manner.  
105413                   Write messages to standard output for each target file indicating the name of the  
105414                   file and that it was touched. Normally, the *makefile* command lines associated with  
105415                   each target are not executed. However, a command line with a <plus-sign> ('+')  
105416                   prefix shall be executed and it is unspecified whether command lines that do not  
105417                   have a <plus-sign> prefix and either expand the *MAKE* macro or are being  
105418                   processed in order to create an include file or to bring it up-to-date (see **Include**  
105419                   **Lines** in the EXTENDED DESCRIPTION section) are executed.
- 105420            Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any  
105421            options specified on the *make* utility command line. If the **-k** and **-S** options are both specified  
105422            on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option  
105423            specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment  
105424            variable, the result is undefined.

#### 105425 OPERANDS

- 105426            The following operands shall be supported:
- 105427            *target\_name*        Target names, as defined in the EXTENDED DESCRIPTION section. If no target  
105428                   is specified, while *make* is processing the makefiles, the first target that *make*  
105429                   encounters that is not a special target or an inference rule shall be used.
- 105430            *macro=value*  
105431            *macro::=value*  
105432            *macro:::=value*        Delayed and immediate expansion macro definitions, as defined in [Macros](#) (on  
105433                   page 3139).
- 105434            Delayed and immediate expansion macro definitions can be intermixed, and shall be processed  
105435            in the order specified. If any macro definition appears after a *target\_name* operand on the *make*  
105436            utility command line, the results are unspecified.

#### 105437 STDIN

- 105438            The standard input shall be used only if the *makefile* option-argument is '-'. See the INPUT  
105439            FILES section.

#### 105440 INPUT FILES

- 105441            The input file, otherwise known as the makefile, is a text file containing rules, macro definitions,  
105442            include lines, and comments. See the EXTENDED DESCRIPTION section.

#### 105443 ENVIRONMENT VARIABLES

- 105444            The following environment variables shall affect the execution of *make*:
- 105445            **LANG**            Provide a default value for the internationalization variables that are unset or null.  
105446                   (See [XBD Section 8.2](#) (on page 169) for the precedence of internationalization  
105447                   variables used to determine the values of locale categories.)
- 105448            **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
105449                   internationalization variables.

- 105450 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 105451 characters (for example, single-byte as opposed to multi-byte characters in  
 105452 arguments and input files).
- 105453 **LC\_MESSAGES**  
 105454 Determine the locale that should be used to affect the format and contents of  
 105455 diagnostic messages written to standard error.
- 105456 **MAKEFLAGS**  
 105457 This variable shall be interpreted as a character string representing a series of  
 105458 option characters to be used as the default options. The implementation shall  
 105459 accept both of the following formats (but need not accept them when intermixed):
- 105460 • The characters are option letters without the leading <hyphen-minus>  
 105461 characters or <blank> separation used on a *make* utility command line.
  - 105462 • The characters are formatted in a manner similar to the use of the *make* utility  
 105463 in shell commands: options are preceded by <hyphen-minus> characters and  
 105464 <blank>-separated as described in XBD [Section 12.2](#) (on page 215). The  
 105465 *macro=value* macro definition operands can also be included. The difference  
 105466 between the contents of *MAKEFLAGS* and the use of the *make* utility in shell  
 105467 commands is that the contents of the variable shall not be subjected to the  
 105468 word expansions (see [Section 2.6](#), on page 2483) associated with parsing shell  
 105469 command lines.
- 105470 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 105471 XSI **PROJECTDIR**  
 105472 Provide a directory to be used to search for SCCS files not found in the current  
 105473 directory. In all of the following cases, the search for SCCS files is made in the  
 105474 directory **SCCS** in the identified directory. If the value of *PROJECTDIR* begins with  
 105475 a <slash>, it shall be considered an absolute pathname; otherwise, the value of  
 105476 *PROJECTDIR* is treated as a user name and that user's initial working directory  
 105477 shall be examined for a subdirectory **src** or **source**. If such a directory is found, it  
 105478 shall be used. Otherwise, the value is used as a relative pathname.
- 105479 If *PROJECTDIR* is not set or has a null value, the search for SCCS files shall be  
 105480 made in the directory **SCCS** in the current directory.
- 105481 The setting of *PROJECTDIR* affects all files listed in the remainder of this utility  
 105482 description for files with a component named **SCCS**.
- 105483 The value of the *SHELL* environment variable shall not be used as a macro and shall not be  
 105484 modified by defining the *SHELL* macro in a makefile or on the command line. All other  
 105485 environment variables, including those with null values, shall be used as macros, as defined in  
 105486 [Macros](#) (on page 3139).
- 105487 **ASYNCHRONOUS EVENTS**  
 105488 For **SIGHUP**, **SIGINT**, **SIGQUIT**, and **SIGTERM** signals, if the signal was not inherited as  
 105489 ignored, none of the **-n**, **-p**, or **-q** options was specified, *make* is currently processing a target or  
 105490 inference rule, and the current target is neither a directory nor a prerequisite of the special  
 105491 targets **.PHONY** or **.PRECIOUS**:
- 105492 • The *make* utility shall catch the signal and, if the time of last data modification of the  
 105493 current target has changed since *make* began processing the rule to bring that target up to  
 105494 date, remove that target; it may also remove that target if the time of last data modification  
 105495 has not changed. Any targets removed in this manner shall be reported in diagnostic or

105496 informational messages of unspecified format, written to standard error.

105497 • If *make* writes a diagnostic message to standard error, it shall exit with a status that  
105498 indicates an error occurred; otherwise, it shall set the signal to default and re-signal itself.

105499 In all other circumstances, *make* shall take the standard action for all signals; see [Section 1.4](#) (on  
105500 page 2462).

#### 105501 **STDOUT**

105502 If *make* is invoked without any work needing to be done, it may write a message to standard  
105503 output indicating that no action was taken. Otherwise, the *make* utility shall write all commands  
105504 to be executed (and the filenames of files touched for the `-t` option in a message of unspecified  
105505 format) to standard output unless the `-s` option was specified, the command is prefixed with a  
105506 `<commercial-at>` ('@'), or the special target `.SILENT` has either the current target as a  
105507 prerequisite or has no prerequisites.

#### 105508 **STDERR**

105509 The standard error shall be used for diagnostic messages and may be used for informational  
105510 messages about target removals (see ASYNCHRONOUS EVENTS).

#### 105511 **OUTPUT FILES**

105512 Files can be created when the `-t` option is present. Additional files can also be created by the  
105513 utilities invoked by *make*.

#### 105514 **EXTENDED DESCRIPTION**

105515 The *make* utility attempts to perform the actions, specified in one or more makefiles, required to  
105516 ensure that specified targets are up-to-date. By default, the following files shall be tried in  
105517 sequence: `./makefile` and `./Makefile`. If neither `./makefile` nor `./Makefile` is found, other  
105518 XSI implementation-defined files may also be tried. On XSI-conformant systems, the additional  
105519 files `./s.makefile`, `SCCS/s.makefile`, `./s.Makefile`, and `SCCS/s.Makefile` shall also be tried. The  
105520 `-f` option shall direct *make* to ignore any of these default files and use the specified option-  
105521 argument as a makefile instead. If this option-argument is `'-'`, standard input shall be used.

105522 The term *makefile* is used to refer to any makefile contents provided by the user, whether in  
105523 `./makefile` or its variants, or specified by the `-f` option.

105524 A target shall be considered up-to-date if it exists and is newer than all of its prerequisites, or if it  
105525 has already been made up-to-date by the current invocation of *make* (regardless of the target's  
105526 existence or age), except that targets that are made up-to-date in order for them to be processed  
105527 as include line pathnames (see **Include Lines** below) need not be considered up-to-date during  
105528 later processing. A target may also be considered up-to-date if it exists, is the same age as one or  
105529 more of its prerequisites, and is newer than the remaining prerequisites (if any). The *make* utility  
105530 shall treat all prerequisites as targets themselves and recursively ensure that they are up-to-date,  
105531 processing them in the order in which they appear in the rule. The *make* utility shall use the  
105532 modification times of files to determine whether the corresponding targets are out-of-date.

105533 To ensure that a target is up-to-date, *make* shall ensure that all of the prerequisites of the target  
105534 are up-to-date, then check to see if the target itself is up-to-date. If the target is not up-to-date,  
105535 the target shall be made up-to-date by executing the rule's commands (if any). If the target does  
105536 not exist after the target has been successfully made up-to-date, the target shall be treated as  
105537 being newer than any target for which it is a prerequisite.

105538 If a target exists and there is neither a target rule nor an inference rule for the target, the target  
105539 shall be considered up-to-date. It shall be an error if *make* attempts to ensure that a target is up-  
105540 to-date but the target does not exist and there is neither a target rule nor an inference rule for the  
105541 target.

105542 **Makefile Syntax**

105543 A makefile can contain rules, macro definitions (see [Macros](#), on page 3139), include lines, and  
 105544 comments. There are two kinds of rules: *target rules*, including special targets (see [Target Rules](#),  
 105545 on page 3137), and *inference rules* (see [Inference Rules](#), on page 3143). The *make* utility shall  
 105546 contain a set of built-in inference rules. If the `-r` option is present, the built-in rules shall not be  
 105547 used and the suffix list shall be cleared. Additional rules of both kinds can be specified in a  
 105548 makefile. If a rule is defined more than once, the value of the rule shall be that of the last one  
 105549 specified. Macros can also be defined more than once, and the value of the macro is specified in  
 105550 [Macros](#) (on page 3139). There are three kinds of comments: blank lines, empty lines, and a  
 105551 `<number-sign>` ('#') and all following characters up to the first unescaped `<newline>`  
 105552 character. Blank lines, empty lines, and lines with `<number-sign>` ('#') as the first character on  
 105553 the line are also known as comment lines.

105554 Target and inference rules can contain *command lines*. Command lines can have a prefix that  
 105555 shall be removed before execution (see [Makefile Execution](#), on page 3136).

105556 When an escaped `<newline>` (one preceded by a `<backslash>`) is found anywhere in the  
 105557 makefile except in a command line after macro expansion, an include line, or a line immediately  
 105558 preceding an include line, it shall be replaced, along with any leading white space on the next  
 105559 line, with a single `<space>`. After all macro expansion is complete, when an escaped `<newline>`  
 105560 is found in a command line in a makefile, the command line that is executed shall contain the  
 105561 `<backslash>`, the `<newline>`, and the next line, except that the first character of the next line shall  
 105562 not be included if it is a `<tab>`. When an escaped `<newline>` is found in an include line or in a  
 105563 line immediately preceding an include line, the behavior is unspecified.

105564 **Include Lines**

105565 If the word **include**, optionally prefixed with a `<hyphen-minus>` character, appears at the  
 105566 beginning of a line and is followed by one or more `<blank>` characters, the string formed by the  
 105567 remainder of the line shall be processed as follows to produce one or more pathnames:

- 105568 • The trailing `<newline>`, any `<blank>` characters immediately preceding a comment, and  
 105569 any comment shall be discarded. If the resulting string contains any double-quote  
 105570 characters ('"') the behavior is unspecified.
- 105571 • The resulting string shall be processed for macro expansion (see [Macros](#), on page 3139).
- 105572 • Any `<blank>` characters that appear after the first non-`<blank>` shall be used as separators  
 105573 to divide the macro-expanded string into fields. It is unspecified whether pathname  
 105574 expansion (see [Section 2.14](#), on page 2523) is also performed.
- 105575 • If the processing of separators and optional pathname expansion results in zero non-empty  
 105576 fields, the behavior is unspecified. If it results in at least one non-empty field, these fields  
 105577 are taken as pathnames.

105578 For each pathname so identified, in the order specified:

- 105579 • If the pathname does not begin with a `'/'`, it shall be treated as relative to the current  
 105580 working directory of the process, not relative to the directory containing the makefile.
- 105581 • The *make* utility shall use one of the following two methods to attempt to create the file or  
 105582 bring it up-to-date:
  - 105583 1. The “immediate remaking” method
    - 105584 If *make* uses this method, any target rules or inference rules for the pathname that  
 105585 were parsed before the include line was parsed shall be used to attempt to create the

- 105586 file or to bring it up-to-date before opening the file.
- 105587 2. The “delayed remaking” method
- 105588 If *make* uses this method, no attempt shall be made to create the file or bring it up-
- 105589 to-date until after the makefile(s) have been read. During processing of the include
- 105590 line, *make* shall read the current contents of the file, if it exists, or treat it as an empty
- 105591 file if it does not exist. Once the makefile(s) have been read, *make* shall use any
- 105592 applicable target rule or inference rule for the pathname, regardless of whether it is
- 105593 parsed before or after the include line, when creating the file or bringing it up-to-
- 105594 date. Additionally in this case, the new contents of the file, if it is successfully
- 105595 created or updated, shall be used when processing rules for the following targets
- 105596 after the makefile(s) have been read:
- 105597 • The *target\_name* operands, if any.
  - 105598 • The first target *make* encounters that is not a special target or an inference rule,
  - 105599 if no *target\_name* operands are specified.
  - 105600 • All targets that are prerequisites, directly or recursively, of the above targets.
- 105601 If the pathname is relative, the file does not exist, and an attempt to create it using a rule
- 105602 has not been made and will not be made, it is unspecified whether additional directories
- 105603 are searched for an existing file of the same relative pathname.
- 105604 If, after proceeding as described above, the file still cannot be opened:
- 105605 • If the word **include** was prefixed with a <hyphen-minus> character, the file shall be
  - 105606 ignored.
  - 105607 • Otherwise, an error shall occur.
- 105608 • The contents of the file specified by the pathname shall be read and processed as if they
  - 105609 appeared in the makefile in place of the include line. If the file ends with an escaped
  - 105610 <newline> the behavior is unspecified.
  - 105611 • The file may itself contain further include lines. Implementations shall support nesting of
  - 105612 include files up to a depth of at least 16.

### 105613 Makefile Execution

- 105614 Makefile command lines shall be processed one at a time.
- 105615 Makefile command lines can have one or more of the following prefixes: a <hyphen-minus>
- 105616 ('-'), a <commercial-at> ('@'), or a <plus-sign> ('+'). These shall modify the way in which
- 105617 *make* processes the command.
- 105618 – If the command prefix contains a <hyphen-minus>, or the `-i` option is present, or the special
  - 105619 target **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any
  - 105620 error found while executing the command shall be ignored.
  - 105621 @ If the command prefix contains a <commercial-at> and the *make* utility command line `-n`
  - 105622 option is not specified, or the `-s` option is present, or the special target **.SILENT** has either
  - 105623 the current target as a prerequisite or has no prerequisites, the command shall not be
  - 105624 written to standard output before it is executed.
  - 105625 + If the command prefix contains a <plus-sign>, the command shall be executed even if `-n`,
  - 105626 `-q`, or `-t` is specified.
- 105627 An *execution line* is built from the command line by removing any prefix characters. Except as



105628 described under the <commercial-at> ('@') prefix, the execution line shall be written to the  
 105629 standard output, optionally preceded by a <tab>. The execution line shall then be executed by a  
 105630 shell as if it were passed as the argument to the *system()* interface, except that if errors are not  
 105631 being ignored then the shell *-e* option shall also be in effect. If errors are being ignored for the  
 105632 command (as a result of the *-i* option, a '-' command prefix, or a **.IGNORE** special target), the  
 105633 shell *-e* option shall not be in effect. The environment for the command being executed shall  
 105634 contain all of the variables in the environment of *make*.

105635 By default, when *make* receives a non-zero status from the execution of a command, it shall  
 105636 terminate with an error message to standard error.

## 105637 Target Rules

105638 Target rules are formatted as follows:

```
105639 target [target...]: [prerequisite...][;command]
105640 [<tab>command
105641 <tab>command
105642 ...]
```

105643 Target entries are specified by a <blank>-separated, non-null list of targets, then a <colon>, then  
 105644 a <blank>-separated, possibly empty list of prerequisites. Text following a <semicolon>, if any,  
 105645 and all following lines that begin with a <tab>, are makefile command lines to be executed to  
 105646 update the target. The first non-empty line that does not begin with a <tab> or '#' shall begin a  
 105647 new entry. Any comment line may begin a new entry.

105648 Applications shall select target names from the set of characters consisting solely of slashes,  
 105649 hyphens, periods, underscores, digits, and alphabets from the portable character set (see XBD  
 105650 Section 6.1, on page 117). Implementations may allow other characters in target names as  
 105651 extensions. The interpretation of targets containing the characters '%' and '"' is  
 105652 implementation-defined.

105653 A target that has prerequisites, but does not have any commands, can be used to add to the  
 105654 prerequisite list for that target. Only one target rule for any given target can contain commands.

105655 Lines that begin with one of the following are called *special targets* and control the operation of  
 105656 *make*:

105657 **.DEFAULT** If the makefile contains this special target, the application shall ensure that it is  
 105658 specified with commands, but without prerequisites. The commands shall be used  
 105659 by *make* if there are no other rules available to build a target.

105660 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors  
 105661 from commands associated with them to be ignored in the same manner as  
 105662 specified by the *-i* option. Subsequent occurrences of **.IGNORE** shall add to the  
 105663 list of targets ignoring command errors. If no prerequisites are specified, *make* shall  
 105664 behave as if the *-i* option had been specified and errors from all commands  
 105665 associated with all targets shall be ignored.

## 105666 .NOTPARALLEL

105667 The application shall ensure that this special target is specified without  
 105668 prerequisites or commands. When specified, *make* shall update one target at a time,  
 105669 regardless of whether the *-j maxjobs* option is specified. If the *-j maxjobs* option  
 105670 is specified, the option shall continue to be passed unchanged to sub-*make*  
 105671 invocations via *MAKEFLAGS*.

- 105672       **.PHONY**     Prerequisites of this special target are targets themselves; these targets (known as  
105673                   *phony targets*) shall be considered always out-of-date when the *make* utility begins  
105674                   executing. If a phony target's commands are executed, that phony target shall then  
105675                   be considered up-to-date until the execution of *make* completes. Subsequent  
105676                   occurrences of **.PHONY** shall add to the list of phony targets. A **.PHONY** special  
105677                   target with no prerequisites shall be ignored. If the `-t` option is specified, phony  
105678                   targets shall not be touched. Phony targets shall not be removed if *make* receives  
105679                   one of the asynchronous events explicitly described in the ASYNCHRONOUS  
105680                   EVENTS section.
- 105681       **.POSIX**     The application shall ensure that this special target is specified without  
105682                   prerequisites or commands. If it appears as the first non-comment line in the  
105683                   makefile, *make* shall process the makefile as specified by this section; otherwise, the  
105684                   behavior of *make* is unspecified.
- 105685       **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the  
105686                   asynchronous events explicitly described in the ASYNCHRONOUS EVENTS  
105687                   section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious  
105688                   files. If no prerequisites are specified, all targets in the makefile shall be treated as  
105689                   if specified with **.PRECIOUS**.
- 105690 XSI     **.SCCS\_GET** The application shall ensure that this special target is specified without  
105691                   prerequisites. If this special target is included in a makefile, the commands  
105692                   specified with this target shall replace the default commands associated with this  
105693                   special target (see [Default Rules](#), on page 3146). The commands specified with this  
105694                   target are used to get all SCCS files that are not found in the current directory.
- 105695                   When source files are named in a list of prerequisites, *make* shall treat them just like  
105696                   any other target. Because the source file is presumed to be present in the directory,  
105697                   there is no need to add an entry for it to the makefile. When a target has no  
105698                   prerequisites, but is present in the directory, *make* shall assume that that file is up-  
105699                   to-date. If, however, an SCCS file named **SCCS/s.source\_file** is found for a target  
105700                   *source\_file*, *make* compares the timestamp of the target file with that of the  
105701                   **SCCS/s.source\_file** to ensure the target is up-to-date. If the target is missing, or if  
105702                   the SCCS file is newer, *make* shall automatically issue the commands specified for  
105703                   the **.SCCS\_GET** special target to retrieve the most recent version. However, if the  
105704                   target is writable by anyone, *make* shall not retrieve a new version.
- 105705       **.SILENT**   Prerequisites of this special target are targets themselves; this shall cause  
105706                   commands associated with them not to be written to the standard output before  
105707                   they are executed. Subsequent occurrences of **.SILENT** shall add to the list of  
105708                   targets with silent commands. If no prerequisites are specified, *make* shall behave  
105709                   as if the `-s` option had been specified and no commands or touch messages  
105710                   associated with any target shall be written to standard output.
- 105711       **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are  
105712                   used in conjunction with the inference rules (see [Inference Rules](#), on page 3143). If  
105713                   **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be  
105714                   cleared.
- 105715       **.WAIT**     The application shall ensure that this special target, if specified as a target, is  
105716                   specified without prerequisites or commands. When **.WAIT** appears as a target, it  
105717                   shall have no effect. When **.WAIT** appears in a target rule as a prerequisite, it shall  
105718                   not itself be treated as a prerequisite; however, *make* shall not recursively process  
105719                   the prerequisites (if any) to the right of the **.WAIT** until the prerequisites (if any) to

105720 the left of it have been brought up-to-date. Implementations may also enforce the  
 105721 same ordering between the affected prerequisites while processing other target  
 105722 rules that have some or all of the same affected prerequisites.

105723 The special targets **.IGNORE**, **.NOTPARALLEL**, **.PHONY**, **.POSIX**, **.PRECIOUS**, **.SILENT**,  
 105724 **.SUFFIXES**, and **.WAIT** shall be specified without commands.

105725 Targets and prerequisites consisting of a leading <period> followed by the uppercase letters  
 105726 "POSIX" and then any other characters are reserved for future standardization. Targets and  
 105727 prerequisites consisting of a leading <period> followed by one or more uppercase letters, that  
 105728 are not described above, are reserved for implementation extensions.

## 105729 **Macros**

105730 A macro can be one of two flavors, *delayed-expansion* or *immediate-expansion*.

105731 The following form defines a delayed-expansion macro (replacing any previous definition of the  
 105732 macro named by *string1*):

```
105733 string1 = [string2]
```

105734 The following form defines an immediate-expansion macro (replacing any previous definition of  
 105735 the macro named by *string1*):

```
105736 string1 ::= [string2]
```

105737 The following form defines a delayed-expansion macro (replacing any previous definition of the  
 105738 macro named by *string1*):

```
105739 string1 ::= [string2]
```

105740 by immediately expanding macros in *string2*, if any, before assigning the value.

105741 The following form defines a delayed-expansion macro (replacing any previous definition of the  
 105742 macro named by *string1*):

```
105743 string1 != [string2]
```

105744 by immediately expanding macros in *string2*, if any, and then executing the result as a shell  
 105745 command as if it were passed as the argument to the *system()* interface. The *make* utility shall  
 105746 capture the standard output from the shell execution and shall remove all white space at the  
 105747 beginning, remove a single trailing <newline> character (if there is one), and then replace all  
 105748 remaining <newline> characters with <space> characters to produce the value assigned to the  
 105749 macro named by *string1*. It shall not be an error if the shell command has non-zero exit status.

105750 The following form defines a delayed-expansion macro, but only if the macro named by *string1*  
 105751 is not already defined:

```
105752 string1 ?= [string2]
```

105753 The following form (the *append* form) appends additional text to the value of a macro:

```
105754 string1 += [string2]
```

105755 When using the append form:

- 105756 • If the macro named by *string1* does not exist, this form shall be equivalent to the delayed-  
 105757 expansion form

```
105758 string1 = [string2]
```

105759 • If the macro named by *string1* exists and is an immediate-expansion macro, then a <space>  
 105760 or <tab> character followed by the evaluation of *string2* shall be appended to the value  
 105761 currently assigned to the macro named by *string1*.

105762 • If the macro named by *string1* exists and is a delayed-expansion macro, then a <space> or  
 105763 <tab> character followed by the unevaluated *string2* shall be appended to the value  
 105764 currently assigned to the macro named by *string1*.

105765 In all cases the value of *string1* is defined as all characters from the first non-<blank> character to  
 105766 the last non-<blank> character, inclusive, before the =, ::=, :::=, !=, ?=, or +=. Portable  
 105767 applications shall ensure that a <blank> precedes the ::=, :::=, !=, ?=, or += in those forms to  
 105768 avoid any parsing ambiguity with implementations that permit <colon>, <exclamation-mark>,  
 105769 <question-mark>, or <plus-sign> in macro names as extensions. The value of *string2* is defined  
 105770 as all characters from the first non-<blank> character, if any, after the <equals-sign>, up to but  
 105771 not including a comment character (' # ') or an unescaped <newline>.

105772 Portable applications shall select macro names from the set of characters consisting solely of  
 105773 characters from the portable filename character set. Implementations may allow other characters  
 105774 in macro names as extensions; however, a macro name shall not contain an <equals-sign>,  
 105775 <blank>, or control character.

105776 Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro  
 105777 expansions in *string2* of macro definition lines shall be performed according to the form of  
 105778 macro definition used. In immediate-expansion forms (including appending to an existing  
 105779 immediate-expansion macro), they shall be expanded in the macro definition line and the result  
 105780 of the expansion shall not be scanned for further macros when the macro identified by *string1* is  
 105781 expanded. In delayed-expansion forms (including appending to an existing delayed-expansion  
 105782 macro, and conditional assignment to a macro not previously existing), they shall not be  
 105783 expanded in the macro definition line; they shall be expanded when the macro identified by  
 105784 *string1* is expanded, and the result of the expansion shall be scanned for further macros.  
 105785 Implementations shall support at least 100 levels of indirection.

105786 Macros can appear anywhere in the makefile. Macro expansions using the forms  $\$(string1)$  or  
 105787  $\${string1}$  shall be replaced by *string2*, as follows:

- 105788 • Macros in target lines shall be evaluated when the target line is read.
- 105789 • Macros in makefile command lines shall be evaluated when the command is executed.
- 105790 • Macros in the string before the <equals-sign> in a macro definition shall be evaluated  
 105791 when the macro assignment is made.
- 105792 • Immediate-expansion macros shall be evaluated immediately when the macro assignment  
 105793 is made, and this value shall be used as the replacement until the immediate-expansion  
 105794 macro is redefined.
- 105795 • Delayed-expansion macros after the <equals-sign> in macro definitions other than the  
 105796 ::=, !=, and += forms, and after the <equals-sign> in += form macro definitions where  
 105797 the macro named by *string1* exists and is a delayed-expansion macro, shall only be  
 105798 evaluated when the defined macro is expanded.

105799 The parentheses or braces are optional if *string1* is a single character. The string "\$\$" shall be  
 105800 replaced by the single character '\$', except during the immediate expansion performed for the  
 105801 ::= operator, where it shall be left unmodified. If *string1* in a macro expansion contains a  
 105802 macro expansion, that inner macro expansion shall be performed first and the result substituted  
 105803 into *string1* to produce the macro name used for the outer macro expansion.

105804 Macro expansions using the forms  $\$(string1:subst1=[subst2])$  or  $\${string1:subst1=[subst2]}$  can be

105805 used to replace all occurrences of *subst1* with *subst2* when the macro substitution is performed.  
 105806 The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word in *string1*  
 105807 (where a *word*, in this context, is defined to be a string delimited by the beginning of the value, a  
 105808 <blank>, or a <newline>). If *string1* in a macro expansion contains a macro expansion, that inner  
 105809 macro expansion shall be performed as described above and the result substituted into *string1*  
 105810 to produce the macro name used for the outer macro expansion.

105811 Macro expansions using the forms  $\$(string1:[op]\%[os]=[np][\%][ns])$  or  
 105812  $\${string1:[op]\%[os]=[np][\%][ns]}$  are called pattern macro expansions, where *op* is the old prefix,  
 105813 *os* is the old suffix, *np* is the new prefix and *ns* is the new suffix. Any item inside square brackets  
 105814 is optional. With this form, when the macro *string1* is expanded each white-space-separated  
 105815 word that completely matches the  $[op]\%[os]$  pattern on the left-hand side of the <equals-sign>  
 105816 ('='), where the <percent> ('%') character matches zero or more characters, shall be replaced  
 105817 by the right-hand side of the <equals-sign> and shall then be further modified according to the  
 105818 use of <percent> characters as described below. Any words that do not match shall be  
 105819 unmodified in the expansion.

105820 If more than one <percent> character appears on the left-hand side of the <equals-sign> ('='),  
 105821 the second and subsequent <percent> characters shall be treated as literal characters in *os*.

105822 If no <percent> character appears on the right-hand side of the <equals-sign>, no further  
 105823 modification of the word shall be performed. If a single <percent> character appears on the  
 105824 right-hand side, the <percent> character in the word shall be replaced with the characters  
 105825 matched by the <percent> on the left-hand side. If more than one <percent> character appears  
 105826 on the right-hand side, it is unspecified whether the first <percent> character in the word is  
 105827 replaced with the characters matched by the <percent> on the left-hand side and all remaining  
 105828 <percent> characters are left unchanged, or each <percent> character is replaced with the  
 105829 characters matched by the <percent> on the left-hand side.

105830 In both macro expansion forms, any macro expansions on the right-hand side of the <colon>  
 105831 shall be recursively expanded before further examination. If this results in more than one  
 105832 <equals-sign> after the <colon>, the first one shall be the separator.

105833 In all forms of macro expansion, if the value of the macro named by *string1* is an empty string, or  
 105834 if the macro named by *string1* does not exist, the final result shall be an empty string.

105835 **Note:** It is not safe to assume that a macro which has not intentionally been set to a specific value will  
 105836 not exist. See APPLICATION USAGE for more information.

105837 Macro definitions shall be taken from the following sources, in the following logical order,  
 105838 before the makefile(s) are read.

- 105839 1. Macros specified on the *make* utility command line, in the order specified on the  
 105840 command line. It is unspecified whether the internal macros defined in [Internal Macros](#)  
 105841 (on page 3144) are accepted from this source.
- 105842 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the  
 105843 environment variable. It is unspecified whether the internal macros defined in [Internal](#)  
 105844 [Macros](#) (on page 3144) are accepted from this source.
- 105845 3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and  
 105846 including the variables with null values.
- 105847 4. Macros defined in the inference rules built into *make*.

105848 Macro definitions from these sources shall not override macro definitions from a lower-  
 105849 numbered source. Macro definitions from a single source (for example, the *make* utility  
 105850 command line, the *MAKEFLAGS* environment variable, or the other environment variables)

105851 shall override previous macro definitions from the same source.

105852 Macros defined in the makefile(s) shall override macro definitions that occur before them in the  
105853 makefile(s) and macro definitions from source 4. If the `-e` option is not specified, macros defined  
105854 in the makefile(s) shall override macro definitions from source 3. Macros defined in the  
105855 makefile(s) shall not override macro definitions from source 1 or source 2.

105856 Before the makefile(s) are read, all of the *make* utility command line options (except `-f` and `-p`)  
105857 and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not  
105858 already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted  
105859 in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance  
105860 of the *make* command, the original macro's value is recovered. Other implementation-defined  
105861 options and macros, with the exception of the *CURDIR* macro, may also be added to the  
105862 *MAKEFLAGS* macro. If this modifies the value of the *MAKEFLAGS* macro, or, if the  
105863 *MAKEFLAGS* macro is modified at any subsequent time, the *MAKEFLAGS* environment variable  
105864 shall be modified to match the new value of the *MAKEFLAGS* macro. The result of setting  
105865 *MAKEFLAGS* in the Makefile is unspecified.

105866 Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the  
105867 *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other  
105868 implementation-defined variables may also be added to the environment of *make*. Macros  
105869 defined by the *MAKEFLAGS* environment variable and macros defined in the makefile(s) shall  
105870 not be added to the environment of *make* if they are not already in its environment. With the  
105871 exception of *SHELL* (see below), it is unspecified whether macros defined in these ways update  
105872 the value of an environment variable that already exists in the environment of *make*.

105873 The *MAKE* macro shall be treated specially. If *MAKE* is not defined in the environment, the  
105874 *MAKE* macro shall be provided by *make* and set to the value of *argv*[0] passed to *main*() (or  
105875 equivalent, if *make* is not a C program). If this value contains at least one `</code>slash>` and is a relative  
105876 pathname, *make* shall convert it to an absolute pathname. If *MAKE* is defined in the makefile or  
105877 is specified on the command line, it shall replace the original value of the *MAKE* macro.

105878 The *SHELL* macro shall be treated specially. It shall be provided by *make* and set to the pathname  
105879 of the shell command language interpreter (see *sh*). The *SHELL* environment variable shall not  
105880 affect the value of the *SHELL* macro. If *SHELL* is defined in the makefile or is specified on the  
105881 command line, it shall replace the original value of the *SHELL* macro, but shall not affect the  
105882 *SHELL* environment variable. Other effects of defining *SHELL* in the makefile or on the  
105883 command line are implementation-defined.

105884 The *CURDIR* macro shall be treated specially. It shall be provided by *make* and set to an absolute  
105885 pathname of the current working directory when *make* is executed. The value shall be the same  
105886 as the pathname that would be output by the *pwd* utility with either the `-L` or `-P` option; if they  
105887 differ, it is unspecified which value is used. The *CURDIR* environment variable shall not affect  
105888 the value of the *CURDIR* macro unless the `-e` option is specified. If the `-e` option is not specified,  
105889 there is a *CURDIR* environment variable set, and its value is different from the *CURDIR* macro  
105890 value, the environment variable value shall be set to the macro value. If *CURDIR* is defined in  
105891 the makefile, present in the *MAKEFLAGS* environment variable, or specified on the command  
105892 line, it shall replace the original value of the *CURDIR* macro in accordance with the logical order  
105893 described above, but shall not cause *make* to change its current working directory.

105894 **Inference Rules**

105895 Inference rules are formatted as follows:

```

105896 target:
105897 <tab>command
105898 [<tab>command]
105899 ...
105900 line that does not begin with <tab> or #

```

105901 The application shall ensure that the *target* portion is a valid target name (see [Target Rules](#), on  
 105902 page 3137) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as  
 105903 prerequisites of the **.SUFFIXES** special target and *s1* and *s2* do not contain any *<slash>* or  
 105904 *<period>* characters.) If there is only one *<period>* in the target, it is a single-suffix inference  
 105905 rule. Targets with two periods are double-suffix inference rules. Inference rules can have only  
 105906 one target before the *<colon>*.

105907 The application shall ensure that the makefile does not specify prerequisites for inference rules;  
 105908 no characters other than white space shall follow the *<colon>* in the first line, except when  
 105909 creating the *empty rule*, described below. Prerequisites are inferred, as described below.

105910 Inference rules can be redefined. A target that matches an existing inference rule shall overwrite  
 105911 the old inference rule. An empty rule can be created with a command consisting of simply a  
 105912 *<semicolon>* (that is, the rule still exists and is found during inference rule search, but since it is  
 105913 empty, execution has no effect). The empty rule can also be formatted as follows:

```

105914 rule: ;

```

105915 where zero or more *<blank>* characters separate the *<colon>* and *<semicolon>*.

105916 The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be  
 105917 made up-to-date. A list of inference rules defines the commands to be executed. By default, *make*  
 105918 contains a built-in set of inference rules. Additional rules can be specified in the makefile.

105919 The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by  
 105920 the inference rules. The order in which the suffixes are specified defines the order in which the  
 105921 inference rules for the suffixes are used. New suffixes shall be appended to the current list by  
 105922 specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites  
 105923 shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is  
 105924 required to change the order of the suffixes.

105925 Normally, the user would provide an inference rule for each suffix. The inference rule to update  
 105926 a target with a suffix *.s1* from a prerequisite with a suffix *.s2* is specified as a target *.s2.s1*. The  
 105927 internal macros provide the means to specify general inference rules (see [Internal Macros](#), on  
 105928 page 3144).

105929 When no target rule with commands is found to update a target, the inference rules shall be  
 105930 checked. The suffix of the target (*.s1*) to be built shall be compared to the list of suffixes specified  
 105931 by the **.SUFFIXES** special targets. If the *.s1* suffix is found in **.SUFFIXES**, the inference rules  
 105932 shall be searched in the order defined for the first *.s2.s1* rule whose prerequisite file (*\$.s2*) exists.  
 105933 If the target is out-of-date with respect to this prerequisite, the commands for that inference rule  
 105934 shall be executed. Prerequisites added by target rules without commands shall not affect the  
 105935 selection of the applicable inference rule.

105936 If the target to be built does not contain a suffix and there is no rule for the target, the single-  
 105937 suffix inference rules shall be checked. The single-suffix inference rules define how to build a  
 105938 target if a file is found with a name that matches the target name with one of the single suffixes

105939 appended. A rule with one suffix *.s2* is the definition of how to build *target* from *target.s2*. The  
 105940 other suffix (*.s1*) is treated as null.

105941 XSI A <tilde> ('~') in the above rules refers to an SCCS file in the current directory. Thus, the rule  
 105942 *.c.o* would transform an SCCS C-language source file into an object file (*.o*). Because the *s.* of  
 105943 the SCCS files is a prefix, it is incompatible with *make's* suffix point of view. Hence, the '~' is a  
 105944 way of changing any file reference into an SCCS file reference.

### 105945 Libraries

105946 If a target or prerequisite contains parentheses, it shall be treated as a member of an archive  
 105947 library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o*  
 105948 to the member name. The application shall ensure that the member is an object file with the *.o*  
 105949 suffix. The modification time of the expression is the modification time for the member as kept  
 105950 in the archive library; see *ar*. The *.a* suffix shall refer to an archive library. The *.s2.a* rule shall be  
 105951 used to update a member in the library from a file with a suffix *.s2*.

### 105952 Internal Macros

105953 The *make* utility shall maintain a set of internal macros that can be used in the commands of  
 105954 target and inference rules, as described below. In order to clearly define the meaning of these  
 105955 macros, some clarification of the terms *target rule*, *inference rule*, *target*, and *prerequisite* is  
 105956 necessary.

105957 Target rules are specified by the user in a makefile for a particular target. Inference rules are  
 105958 user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites  
 105959 are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those  
 105960 prerequisites that are generated when inference rules are used. Inference rules are applied to  
 105961 implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in  
 105962 the makefile. Target rules are applied to targets specified in the makefile.

105963 Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)  
 105964 shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon  
 105965 recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be  
 105966 processed recursively until a target is found that has no prerequisites, or further recursion would  
 105967 require applying two inference rules one immediately after the other, at which point the  
 105968 recursion shall stop. As an extension, implementations may continue recursion when two or  
 105969 more successive inference rules need to be applied; however, if there are multiple different  
 105970 chains of such rules that could be used to create the target, it is unspecified which chain is used.  
 105971 The recursion shall then back up, updating each target as it goes.

105972 In the definitions that follow, the word *target* refers to one of:

- 105973 • A target specified in the makefile
- 105974 • An explicit prerequisite specified in the makefile that becomes the target when *make*  
 105975 processes it during recursion
- 105976 • An implicit prerequisite that becomes a target when *make* processes it during recursion

105977 In the definitions that follow, the word *prerequisite* refers to one of:

- 105978 • An explicit prerequisite specified in the makefile for a particular target
- 105979 • An implicit prerequisite generated as a result of locating an appropriate inference rule and  
 105980 corresponding file that matches the suffix of the target

105981 The internal macros are:



- 105982        **\$@**        The **\$@** macro shall evaluate to the full target name of the current target, or the archive  
105983 filename part of a library archive target. It shall be evaluated for both target and  
105984 inference rules.
- 105985                For example, in the **.c.a** inference rule, **\$@** represents the out-of-date **.a** file to be built.  
105986 Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-  
105987 date **lib.a**.
- 105988        **\$%**        The **\$%** macro shall be evaluated only when the current target is an archive library  
105989 member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and  
105990 **\$%** shall evaluate to *member.o*. The **\$%** macro shall be evaluated for both target and  
105991 inference rules.
- 105992                For example, in a makefile target rule to build **lib.a(file.o)**, **\$%** represents **file.o**, as  
105993 opposed to **\$@**, which represents **lib.a**.
- 105994        **\$^**        The **\$^** macro shall evaluate to the list of prerequisites for the current target, with any  
105995 duplicates (except the first) removed. It shall be evaluated for both target and inference  
105996 rules. If the list of prerequisites of the target contains any **.WAIT** special targets, the  
105997 results of expanding **\$^** are unspecified.
- 105998                For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and  
105999 regardless of which prerequisites *prog* is out-of-date with respect to, **\$^** represents  
106000 **file1.o**, **file2.o**, and **file3.o**.
- 106001        **\$+**        The **\$+** macro shall be equivalent to **\$^**, except that duplicates shall not be removed; all  
106002 prerequisites shall appear in the order they were listed in the makefile.
- 106003        **\$?**        The **\$?** macro shall evaluate to the list of prerequisites that are newer than the current  
106004 target. It shall be evaluated for both target and inference rules.
- 106005                For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and  
106006 where *prog* is not out-of-date with respect to **file1.o**, but is out-of-date with respect to  
106007 **file2.o** and **file3.o**, **\$?** represents **file2.o** and **file3.o**.
- 106008        **\$<**        In an inference rule, the **\$<** macro shall evaluate to the filename whose existence  
106009 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the **\$<**  
106010 macro shall evaluate to the current target name. The meaning of the **\$<** macro is  
106011 otherwise unspecified.
- 106012                For example, in the **.c.a** inference rule, **\$<** represents the prerequisite **.c** file.
- 106013        **\$\***        The **\$\*** macro shall evaluate to the current target name with its suffix deleted. It shall be  
106014 evaluated at least for inference rules.
- 106015                For example, in the **.c.a** inference rule, **\$.o** represents the out-of-date **.o** file that  
106016 corresponds to the prerequisite **.c** file.
- 106017        Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended  
106018 to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part*  
106019 for 'F'. The directory part is the path prefix of the file without a trailing **<slash>**; for the current  
106020 directory, the directory part is **'.'**. When the **\$?** macro contains more than one prerequisite  
106021 filename, the **\$(?D)** and **\$(?F)** (or **\${?D}** and **\${?F}**) macros expand to a list of directory name parts  
106022 and filename parts respectively.
- 106023        For the target *lib(member.o)* and the **.s2.a** rule, the internal macros shall be defined as:

```

106024    $<    member.s2
106025    $*    member
106026    $@    lib
106027    $^    member.s2
106028    $?    member.s2
106029    $%    member.o

```

### 106030 **Default Rules**

106031 The default rules and macro values for *make* shall achieve results that are the same as if the  
 106032 following were used, except that where a result includes the literal value of a macro, this value  
 106033 may differ. Implementations that do not support the C-Language Development Utilities option  
 106034 may omit **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l**  
 106035 inference rules. Implementations may provide additional macros and rules.

106036 *SPECIAL TARGETS*

```

106037 XSI .SCCS_GET:
106038     sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@

```

```

106039 XSI .SUFFIXES: .o .c .y .l .a .sh .c~ .y~ .l~ .sh~

```

### 106040 **MACROS**

```

106041 AR=ar
106042 ARFLAGS=-rv
106043 YACC=yacc
106044 YFLAGS=
106045 LEX=lex
106046 LFLAGS=
106047 LDFLAGS=
106048 CC=c17
106049 CFLAGS=-O 1
106050 XSI GET=get
106051 GFLAGS=
106052 SCCSFLAGS=
106053 SCCSGETFLAGS=-s

```

### 106054 *SINGLE-SUFFIX RULES*

```

106055 .c:
106056     $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<

```

```

106057 .sh:
106058     cp $< $@
106059     chmod a+x $@

```

```

106060 XSI .c~:
106061     $(GET) $(GFLAGS) -p $< > $*.c
106062     $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
106063 .sh~:
106064     $(GET) $(GFLAGS) -p $< > $*.sh

```

```

106065         cp $*.sh $@
106066         chmod a+x $@

106067     DOUBLE-SUFFIX RULES

106068     .c.o:
106069         $(CC) $(CFLAGS) -c $<

106070     .y.o:
106071         $(YACC) $(YFLAGS) $<
106072         $(CC) $(CFLAGS) -c y.tab.c
106073         rm -f y.tab.c
106074         mv y.tab.o $@

106075     .l.o:
106076         $(LEX) $(LFLAGS) $<
106077         $(CC) $(CFLAGS) -c lex.yy.c
106078         rm -f lex.yy.c
106079         mv lex.yy.o $@

106080     .y.c:
106081         $(YACC) $(YFLAGS) $<
106082         mv y.tab.c $@

106083     .l.c:
106084         $(LEX) $(LFLAGS) $<
106085         mv lex.yy.c $@

106086 XSI .c~.o:
106087     $(GET) $(GFLAGS) -p $< > $*.c
106088     $(CC) $(CFLAGS) -c $*.c

106089     .y~.o:
106090     $(GET) $(GFLAGS) -p $< > $*.y
106091     $(YACC) $(YFLAGS) $*.y
106092     $(CC) $(CFLAGS) -c y.tab.c
106093     rm -f y.tab.c
106094     mv y.tab.o $@

106095     .l~.o:
106096     $(GET) $(GFLAGS) -p $< > $*.l
106097     $(LEX) $(LFLAGS) $*.l
106098     $(CC) $(CFLAGS) -c lex.yy.c
106099     rm -f lex.yy.c
106100     mv lex.yy.o $@

106101     .y~.c:
106102     $(GET) $(GFLAGS) -p $< > $*.y
106103     $(YACC) $(YFLAGS) $*.y
106104     mv y.tab.c $@

106105     .l~.c:
106106     $(GET) $(GFLAGS) -p $< > $*.l
106107     $(LEX) $(LFLAGS) $*.l
106108     mv lex.yy.c $@

```

```

106109     .c.o:
106110         $(CC) -c $(CFLAGS) $<
106111         $(AR) $(ARFLAGS) $@ $*.o
106112         rm -f $*.o

```

#### 106113 EXIT STATUS

106114 When the `-q` option is specified, the *make* utility shall exit with one of the following values:

- 106115 0 All specified targets were already up-to-date.
- 106116 1 One or more targets were not up-to-date.
- 106117 >1 An error occurred.

106118 When the `-q` option is not specified, the *make* utility shall exit with one of the following values:

- 106119 0 All specified targets were already up-to-date, or all commands executed to bring targets up-to-date either exited with status 0 or had a non-zero exit status that was specified (via the `-i` option, the special target `.IGNORE`, or a '-' command prefix) to be ignored.
- 106120
- 106121
- 106122 >0 An error occurred, or at least one command executed to bring a target up-to-date exited with a non-zero exit status that was not specified to be ignored.
- 106123

#### 106124 CONSEQUENCES OF ERRORS

106125 Default.

#### 106126 APPLICATION USAGE

106127 If there is a source file (such as `./source.c`) and there are two SCCS files corresponding to it  
 106128 (`./s.source.c` and `./SCCS/s.source.c`), on XSI-conformant systems *make* uses the SCCS file in the  
 106129 current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*,  
 106130 *get*, and so on) or the *scs* utility for all source files in a given directory. If both forms are used for  
 106131 a given source file, future developers are very likely to be confused.

106132 It is incumbent upon portable makefiles to specify the `.POSIX` special target in order to  
 106133 guarantee that they are not affected by local extensions.

106134 The `-k` and `-S` options are both present so that the relationship between the command line, the  
 106135 `MAKEFLAGS` variable, and the makefile can be controlled precisely. If the `k` flag is passed in  
 106136 `MAKEFLAGS` and a command is of the form:

```
106137 $(MAKE) -S f.o
```

106138 then the default behavior is restored for the child *make*.

106139 When the `-n` option is specified, it is always added to `MAKEFLAGS`. This allows a recursive  
 106140 *make -n target* to be used to see all of the actions that would be taken to update *target*.

106141 Because of widespread historical practice, interpreting a `<number-sign>` ('#') inside a variable  
 106142 as the start of a comment has the unfortunate side-effect of making it impossible to place a  
 106143 `<number-sign>` in a variable, thus forbidding something like:

```
106144 CFLAGS = -D "COMMENT_CHAR='#'"
```

106145 Many historical *make* utilities stop chaining together inference rules when an intermediate target  
 106146 is nonexistent. For example, it might be possible for a *make* to determine that both `.y.c` and `.c.o`  
 106147 could be used to convert a `.y` to a `.o`. Instead, in this case, *make* requires the use of a `.y.o` rule.

106148 The standard set of default rules uses only features provided by other parts of this volume of  
 106149 POSIX.1-2024. They include rules for optional utilities in this volume of POSIX.1-2024, but only  
 106150 rules pertaining to utilities that are provided are needed in an implementation's default set.

106151 Although *make* expands macros that do not exist to an empty string, it is not safe to assume that  
 106152 a macro which has not intentionally been set to a specific value will expand to an empty string  
 106153 for everyone who uses the makefile. There are two reasons for this:

- 106154 1. When another user executes *make*, they might happen to have an environment variable of  
 106155 the same name (which they have set for some unrelated purpose) with a non-empty  
 106156 value.
- 106157 2. A different implementation of *make* (or a later version of the same implementation) might  
 106158 have a non-empty value for the macro in its default set.

106159 This is one aspect of a more general problem, which is that any macro that is not one for which  
 106160 this standard requires a default value, and is not explicitly set either in the makefile or on the  
 106161 *make* command line, can have an unexpected value (or unexpectedly not exist) when the  
 106162 makefile is used by a different user or with a different *make* implementation.

106163 For this reason, it is recommended that makefile authors do not design makefile schemes in  
 106164 which values for non-standard macros are obtained from the user's environment variables. Safer  
 106165 methods of allowing users to configure macro values include:

- 106166 • Setting the macros to default values in a *make* include file where the user can edit the  
 106167 values.
- 106168 • Executing *make* from one or more wrapper scripts which set macro values on the command  
 106169 line (and which do not obtain those values from environment variables).

106170 Makefile authors who follow this recommendation may wish to check for any macros they have  
 106171 overlooked by using a *make* implementation that provides, as an extension, a command-line  
 106172 option that causes *make* to report attempts to expand (or append to) macros that do not exist.  
 106173 Users of makefiles written by others can also benefit from the use of such an option to detect the  
 106174 opposite problem (where the author had a macro being set from an environment variable but the  
 106175 user does not have the variable set). This can avoid misbehaviors that would otherwise be hard  
 106176 to debug, such as a file-processing utility reading from standard input because it was not given  
 106177 any pathnames to process.

106178 Makefile authors who choose not to follow the recommendation can minimize the risk of  
 106179 misbehavior by ensuring all non-standard macros have names that begin with a suitable project-  
 106180 specific prefix.

106181 Use of the `-e` option is strongly discouraged, as it makes the problem discussed above even  
 106182 more likely by introducing the possibility of unexpected values occurring even for macros set in  
 106183 the makefile. If a specific macro needs a value from the environment to override a value set in  
 106184 the makefile, it is safer to set just that macro on the command line, using for example  
 106185 `make MYPROJ_FOO="$MYPROJ_FOO"`. Alternatively, the makefile can be modified to use the  
 106186 `?=` assignment operator for that macro.

106187 Delayed-expansion macros are evaluated when they are used rather than when they are defined.  
 106188 Therefore:

```
106189 MACRO = value1
106190 Immed ::= $(MACRO)
106191 DELAY = $(MACRO)
106192 MACRO = value2

106193 target:
106194     echo $(Immed) $(DELAY)
```

106195 would produce "value1 value2", since **Immed** was expanded while **MACRO** was *value1*,

106196 but **DELAY** was not expanded until it was needed in the *echo* command line when **MACRO** was  
106197 *value2*.

106198 Because the behavior of the += assignment differs depending on whether the macro being  
106199 appended to is a delayed-expansion macro or an immediate-expansion macro, it is  
106200 recommended that when both macro types are used in a set of multiple makefiles and include  
106201 files, a naming convention is adopted to distinguish the two macro types. This avoids any  
106202 confusion about whether += will append the expanded or unexpanded value. A suitable  
106203 convention might be to name delayed-expansion macros using uppercase and underscore  
106204 characters and immediate-expansion macros using a name that contains at least one lowercase  
106205 character.

106206 Some historical applications have been known to intermix *target\_name* and *macro=name* operands  
106207 on the command line, expecting that all of the macros are processed before any of the targets are  
106208 dealt with. Conforming applications do not do this, although some backwards-compatibility  
106209 support may be included in some implementations.

106210 The following characters in filenames may give trouble: '=', ':', '`', single-quote, and '@'.  
106211 In include filenames, pattern matching characters and '"' should also be avoided, as they may  
106212 be treated as special by some implementations.

106213 For inference rules, the descriptions of \$< and \$? seem similar. However, an example shows the  
106214 minor difference. In a makefile containing:

```
106215 foo.o: foo.h
```

106216 if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from  
106217 **foo.c** is used, with \$< equal to **foo.c** and \$? equal to **foo.h**. If **foo.c** is also newer than **foo.o**, \$<  
106218 is equal to **foo.c** and \$? is equal to **foo.h foo.c**.

106219 As a consequence of the general rules for target updating, a useful special case is that if a target  
106220 has no prerequisites and no commands, and the target of the rule is a nonexistent file, then *make*  
106221 acts as if this target has been updated whenever its rule is run.

106222 **Note:** This implies that all targets depending on this one will always have their commands run.

106223 Shell command sequences like `make; cp original copy`; *make* may have problems on  
106224 filesystems where the timestamp resolution is the minimum (1 second) required by the standard  
106225 and where *make* considers identical timestamps to be up-to-date. Conversely, rules like  
106226 `copy: original; cp -p original copy` will result in redundant work on *make*  
106227 implementations that consider identical timestamps to be out-of-date.

106228 The requirements relating to dynamic include files (ones that have rules to create or update  
106229 them) allow two different methods of implementing them, as explained in RATIONALE below.  
106230 In order to write a set of makefiles and include files (both dynamic and static) that work with  
106231 both methods, applications need to ensure the following:

- 106232 • Rules containing commands for creating dynamic include files (and any prerequisites)  
106233 precede the include line that will cause the file to be read.
- 106234 • Include lines and rules for creating dynamic include files do not depend on the contents of  
106235 any earlier dynamic include file. For example, defining a macro in a dynamic include file  
106236 and using that macro in a later include line should be avoided (unless the later include line  
106237 is itself inside the dynamic include file).
- 106238 • One of the rules used in creating a dynamic include file, or a dynamic prerequisite of an  
106239 include file, contains commands to create its target and does not ignore creation failure.

106240 Note that although the first point above appears at first sight to preclude a dynamic include file

106241 that defines its own prerequisites, this can be achieved by having two include lines that name  
 106242 the same file (provided the rules in the file do not contain command lines), as shown in  
 106243 EXAMPLES.

106244 This standard does not specify precedence between macro definition and include directives.  
 106245 Thus, the behavior of:

```
106246 include =foo.mk
```

106247 is unspecified. To define a variable named include, either the white space before the <equal-  
 106248 sign> should be removed, or another macro should be used, as in:

```
106249 INCLUDE_NAME = include
106250 $(INCLUDE_NAME) =foo.mk
```

106251 On the other hand, if the intent is to include a file which starts with an <equal-sign>, either the  
 106252 filename should be changed to ./=foo.mk, or the makefile should be written as:

```
106253 INCLUDE_FILE = =foo.mk
106254 include $(INCLUDE_FILE)
```

106255 It is important to be careful when using parallel execution (the `-j` option) and archives. If  
 106256 multiple `$(AR)` commands run at the same time on the same archive file, they will not know  
 106257 about each other and can corrupt the file. If the `-j` option is used, it is necessary to use `.WAIT` in  
 106258 between archive member prerequisites to prevent this (see EXAMPLES).

#### 106259 EXAMPLES

106260 1. The following command:

```
106261 make
```

106262 makes the first target found in the makefile.

106263 2. The following command:

```
106264 make junk
```

106265 makes the target **junk**.

106266 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in  
 106267 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:

```
106268 .POSIX:
106269 pgm: a.o b.o
106270      c17 a.o b.o -o pgm
106271 a.o: incl.h a.c
106272      c17 -c a.c
106273 b.o: incl.h b.c
106274      c17 -c b.c
```

106275 4. The following is an extended version of the previous example that generates *make* include  
 106276 files containing the prerequisites for each object file. The include file also defines its own  
 106277 prerequisites, and the makefile arranges that these are used by including the file twice.  
 106278 With implementations of *make* that create include files during parsing, the first time *make*  
 106279 is run the file does not exist but the use of the <hyphen-minus> prefix on the first include  
 106280 line means it is ignored and is then generated by the rule preceding the second include  
 106281 line. On subsequent runs, if any of the source or header files previously identified has  
 106282 changed, the include file will be updated. There are other ways of updating the include  
 106283 file when headers change, but they involve recursive execution of *make*.

```

106284     .POSIX:
106285     .SUFFIXES: .c .d

106286     OFILES = a.o b.o

106287     pgm: $(OFILES)
106288         c17 $(OFILES) -o pgm
106289     a.o:
106290         c17 -c a.c
106291     b.o:
106292         c17 -c b.c

106293     -include $(OFILES:.o=.d)
106294     .c.d:
106295         +{ \
106296             set -o pipefail; cfile=$<; ofile=${cfile%.c}.o; \
106297             printf '%s %s: %s ' "$$ofile" $@ $<; \
106298             c17 -E $< | LC_ALL=C sed -n \
106299                 '/^#[[:blank:]]*[[[:digit:]]*/s/.*"\([^"]*\.\h\)"*/\1/p' | \
106300                 LC_ALL=C sort -u | tr '\n' ' '; \
106301             echo; \
106302         } > $@
106303     include $(OFILES:.o=.d)

```

This example does not cope with a header file being removed (*make* will complain that it does not know how to make it), necessitating that \*.d be removed and the *make* run again. Alternatively, this can be handled by updating the commands which generate the .d file so that they add an empty target rule for each of its prerequisites.

5. An example for making optimized .o files from .c files is:

```

106309     .c.o:
106310         c17 -c -O 1 $*.c

106311     or:

106312     .c.o:
106313         c17 -c -O 1 $<

```

6. The most common use of the archive interface follows. Here, it is assumed that the source files are all C-language source:

```

106316     lib.a: lib.a(file1.o) .WAIT lib.a(file2.o) .WAIT lib.a(file3.o)
106317         @echo lib.a is now up-to-date

```

The .c.a rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib.a**.

7. The treatment of escaped <newline> characters throughout the makefile is historical practice. For example, the inference rule:

```

106321     .c.o\
106322     :
106323     works, and the macro:

106324     f= bar baz\
106325         biz
106326     a:

```



```

106327         echo ==$f==
106328     echoes "=="bar baz biz=="
106329     If $? were:
106330     /usr/include/stdio.h /usr/include/unistd.h foo.h
106331     then $(?D) would be:
106332     /usr/include /usr/include .
106333     and $(?F) would be:
106334     stdio.h unistd.h foo.h

```

8. The contents of the built-in rules can be viewed by running:

```

106336     make -p -f /dev/null 2>/dev/null

```

9. With the following makefile, `make -j 10 all` may bring `one` and `two` up-to-date in parallel despite the `.WAIT` in the prerequisite list for `foo`. This is because the `.WAIT` does not stop `make` from recursively processing `bar` and its prerequisites in parallel. However, if only `foo` is specified (`make -j 10 foo`), `make` will wait for `one` to be brought up-to-date before bringing `two` up-to-date. Note also that the `.WAIT` does not create a prerequisite relationship between `one` and `two`, so `make -j 10 two` will not build `one`.

```

106343     all: foo bar
106344     foo: one .WAIT two
106345     bar: one two
106346     foo bar one two: ; @echo $@

```

#### 106347 RATIONALE

106348 The *make* utility described in this volume of POSIX.1-2024 is intended to provide the means for  
 106349 changing portable source code into executables that can be run on a POSIX.1-2024-conforming  
 106350 system. It reflects the most common features present in System V and BSD *makes*.

106351 Historically, the *make* utility has been an especially fertile ground for vendor and research  
 106352 organization-specific syntax modifications and extensions. Examples include:

- 106353 • Syntax supporting parallel execution (such as from various multi-processor vendors, GNU,  
 106354 and others)
- 106355 • Additional “operators” separating targets and their prerequisites (System V, BSD, and  
 106356 others)
- 106357 • Modifications of the meaning of internal macros when referencing libraries (BSD and  
 106358 others)
- 106359 • Using a single instance of the shell for all of the command lines of the target (BSD and  
 106360 others)
- 106361 • Allowing <space> characters as well as <tab> characters to delimit command lines (BSD)
- 106362 • Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and  
 106363 others)
- 106364 • Remote execution of command lines (Sprite and others)
- 106365 • Specifying additional special targets (BSD, System V, and most others)

- Specifying an alternate shell to use to process commands.

106366

106367

106368

106369

106370

106371

Additionally, many vendors and research organizations have rethought the basic concepts of *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of *make* fulfills the needs of a different community of users; it is unreasonable for this volume of POSIX.1-2024 to require behavior that would be incompatible (and probably inferior) to historical practice for such a community.

106372

106373

106374

106375

In similar circumstances, when the industry has enough sufficiently incompatible formats as to make them irreconcilable, this volume of POSIX.1-2024 has followed one or both of two courses of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line options have been provided to select the desired behavior (*grep*, *od*, and *pax*).

106376

106377

106378

106379

106380

106381

106382

106383

106384

106385

106386

106387

Because the syntax specified for the *make* utility was, by and large, a subset of the syntaxes accepted by almost all versions of *make* when the original IEEE Std 1003.2-1992 shell and utilities standard was being developed, it was decided that it would be counter-productive to change the name. And since the makefile itself is a basic unit of portability, it would not be completely effective to reserve a new option letter, such as *make -P*, to achieve the portable behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to specify “standard” behavior. This special target does not preclude extensions in the *make* utility, nor does it preclude such extensions being used by the makefile specifying the target; it does, however, preclude any extensions from being applied that could alter the behavior of previously valid syntax; such extensions must be controlled via command line options or new special targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

106388

106389

106390

106391

106392

The portable version of *make* described in this reference page is not intended to be the state-of-the-art software generation tool and, as such, some newer and more leading-edge features have not been included. An attempt has been made to describe the portable makefile in a manner that does not preclude such extensions as long as they do not disturb the portable behavior described here.

106393

106394

When the *-n* option is specified, it is always added to *MAKEFLAGS*. This allows a recursive *make -n target* to be used to see all of the actions that would be taken to update *target*.

106395

106396

106397

The definition of *MAKEFLAGS* allows both the System V letter string and the BSD command line formats. The two formats are sufficiently different to allow implementations to support both without ambiguity.

106398

106399

106400

Early proposals stated that an “unquoted” <number-sign> was treated as the start of a comment. The *make* utility does not pay any attention to quotes. A <number-sign> starts a comment regardless of its surroundings.

106401

106402

106403

106404

106405

106406

The text about “other implementation-defined pathnames may also be tried” in addition to **./makefile** and **./Makefile** is to allow such extensions as **SCCS/s.Makefile** and other variations. It was made an implementation-defined requirement (as opposed to unspecified behavior) to highlight surprising implementations that might select something unexpected like **/etc/Makefile**. XSI-conformant systems also try **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and **SCCS/s.Makefile**.

106407

106408

106409

106410

106411

106412

The default rules are based on System V. The default **CC=** value is *c17* instead of *cc* because this volume of POSIX.1-2024 does not standardize the utility named *cc*. Thus, every conforming application would be required to define **CC=c17** to expect to run. There is no advantage conferred by the hope that the makefile might hit the “preferred” compiler because this cannot be guaranteed to work. Also, since the portable makescript can only use the *c17* options, no advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue

106413 as to whether *c17* is as valuable as *cc*.

106414 Implementations are permitted to include any macro of their choosing in the default set.  
 106415 However, they are encouraged to keep such additions to a minimum in order to reduce the risk  
 106416 of name clashes with user macros.

106417 Implementations are encouraged to provide, as an extension, a command-line option that causes  
 106418 *make* to report attempts to expand (or append to) macros that do not exist. See APPLICATION  
 106419 USAGE for the intended use cases of such an option.

106420 The `-d` option to *make* is frequently used to produce debugging information, but is too  
 106421 implementation-defined to add to this volume of POSIX.1-2024.

106422 The `-p` option is not passed in *MAKEFLAGS* on most historical implementations and to change  
 106423 this would cause many implementations to break without sufficiently increased portability.

106424 Commands that begin with a <plus-sign> ('+') are executed even if the `-n` option is present.  
 106425 Based on the GNU version of *make*, the behavior of `-n` when the <plus-sign> prefix is  
 106426 encountered has been extended to apply to `-q` and `-t` as well. The System V convention of  
 106427 forcing command execution with `-n` when the command line of a target expands the *MAKE*  
 106428 macro was not adopted in earlier versions of this standard, but it is now required because it has  
 106429 become widespread existing practice.

106430 The double <colon> in the target rule format is supported in BSD systems to allow more than  
 106431 one target line containing the same target name to have commands associated with it. Since this  
 106432 is not functionality described in the SVID or XPG3 it has been allowed as an extension, but not  
 106433 mandated.

106434 The default rules are provided with text specifying that the built-in rules shall be the same as if  
 106435 the listed set were used. The intent is that implementations should be able to use the rules  
 106436 without change, but will be allowed to alter them in ways that do not affect the primary  
 106437 behavior.

106438 One point of discussion was whether to drop the default rules list from this volume of  
 106439 POSIX.1-2024. They provide convenience, but do not enhance portability of applications. The  
 106440 prime benefit is in portability of users who wish to type *make command* and have the command  
 106441 build from a **command.c** file.

106442 The historical *MAKESHELL* feature, and related features provided by other *make*  
 106443 implementations, were omitted. In some implementations it is used to let a user override the  
 106444 shell to be used to run *make* commands. This was confusing; for a portable *make*, the shell  
 106445 should be chosen by the makefile writer. Further, a makefile writer cannot require an alternate shell  
 106446 to be used and still consider the makefile portable. While it would be possible to standardize a  
 106447 mechanism for specifying an alternate shell, existing implementations do not agree on such a  
 106448 mechanism, and makefile writers can already invoke an alternate shell by specifying the shell  
 106449 name in the rule for a target; for example:

```
106450 python -c "foo"
```

106451 The *make* utilities in most historical implementations process the prerequisites of a target in left-  
 106452 to-right order, and the makefile format requires this. It supports the standard idiom used in  
 106453 many makefiles that produce *yacc* programs; for example:

```
106454 foo: y.tab.o lex.o main.o
106455      $(CC) $(CFLAGS) -o $@ y.tab.o lex.o main.o
```

106456 In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct  
 106457 **y.tab.h**. Although there may be better ways to express this relationship, it is widely used

106458 historically. Implementations that desire to update prerequisites in parallel should require an  
106459 explicit extension to *make* or the makefile format to accomplish it, as described previously.

106460 The algorithm for determining a new entry for target rules is partially unspecified. Some  
106461 historical *makes* allow comment lines (including blank and empty lines) within the collection of  
106462 commands marked by leading <tab> characters. A conforming makefile must ensure that each  
106463 command starts with a <tab>, but implementations are free to ignore comments without  
106464 triggering the start of a new entry.

106465 The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with  
106466 the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do  
106467 so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned  
106468 up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it  
106469 is required to resend itself the signal it received so that it exits with a status that reflects the  
106470 signal. The results from SIGQUIT are partially unspecified because, on systems that create a file  
106471 named **core** upon receipt of SIGQUIT, the **core** file from *make* would conflict with a **core** file from  
106472 the command that was running when the SIGQUIT arrived. The main concern was to prevent  
106473 damaged files from appearing up-to-date when *make* is rerun.

106474 The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no  
106475 prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites;  
106476 it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of  
106477 targets than for the entire makefile. These extensions to *make* in System V were made to match  
106478 historical practice from the BSD *make*.

106479 Macros are not exported to the environment of commands to be run. This was never the case in  
106480 any historical *make* and would have serious consequences. The environment is the same as the  
106481 environment to *make* except that **MAKEFLAGS** and macros defined on the *make* command line  
106482 are added, and except that macros defined by the **MAKEFLAGS** environment variable and  
106483 macros defined in the makefile(s) may update the value of an existing environment variable  
106484 (other than **SHELL**).

106485 Some implementations do not use *system()* for all command lines, as required by the portable  
106486 makefile format; as a performance enhancement, they select lines without shell metacharacters  
106487 for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but  
106488 merely that the same results be achieved. The metacharacters typically used to bypass the direct  
106489 *execve()* execution have been any of:

106490 = | ^ ( ) ; & < > \* ? [ ] : \$ ` ' " \ \n

106491 The default in some advanced versions of *make* is to group all the command lines for a target and  
106492 execute them using a single shell invocation; the System V method is to pass each line  
106493 individually to a separate shell. The single-shell method has the advantages in performance and  
106494 the lack of a requirement for many continued lines. However, converting to this newer method  
106495 has caused portability problems with many historical makefiles, so the behavior with the POSIX  
106496 makefile is specified to be the same as that of System V. It is suggested that the special target  
106497 **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a  
106498 target or group of targets.

106499 Novice users of *make* have had difficulty with the historical need to start commands with a  
106500 <tab>. Since it is often difficult to discern differences between <tab> and <space> characters on  
106501 terminals or printed listings, confusing bugs can arise. In early proposals, an attempt was made  
106502 to correct this problem by allowing leading <blank> characters instead of <tab> characters.  
106503 However, implementors reported many makefiles that failed in subtle ways following this  
106504 change, and it is difficult to implement a *make* that unambiguously can differentiate between  
106505 macro and command lines. There is extensive historical practice of allowing leading <space>

106506 characters before macro definitions. Forcing macro lines into column 1 would be a significant  
106507 backwards-compatibility problem for some makefiles. Therefore, historical practice was  
106508 restored.

106509 There is substantial variation in the handling of include lines by different implementations.  
106510 However, there is enough commonality for the standard to be able to specify a minimum set of  
106511 requirements that allow the feature to be used portably. Known variations have been explicitly  
106512 called out as unspecified behavior in the description.

106513 The System V dynamic dependency feature was not included. It would support:

106514 `cat: $$@.c`

106515 that would expand to;

106516 `cat: cat.c`

106517 This feature exists only in the new version of System V *make* and, while useful, is not in wide  
106518 usage. This means that macros are expanded twice for prerequisites: once at makefile parse time  
106519 and once at target update time.

106520 Consideration was given to adding metarules to the POSIX *make*. This would make `%.o: %.c` the  
106521 same as `.c.o:`. This is quite useful and available from some vendors, but it would cause too many  
106522 changes to this *make* to support. It would have introduced rule chaining and new substitution  
106523 rules. However, the rules for target names have been set to reserve the `'%'` and `'\"'` characters.  
106524 These are traditionally used to implement metarules and quoting of target names, respectively.  
106525 Implementors are strongly encouraged to use these characters only for these purposes.

106526 A request was made to extend the suffix delimiter character from a `<period>` to any character.  
106527 The metarules feature in newer *makes* solves this problem in a more general way. This volume of  
106528 POSIX.1-2024 is staying with the more conservative historical definition.

106529 The standard output format for the `-p` option is not described because it is primarily a  
106530 debugging option and because the format is not generally useful to programs. In historical  
106531 implementations the output is not suitable for use in generating makefiles. The `-p` format has  
106532 been variable across historical implementations. Therefore, the definition of `-p` was only to  
106533 provide a consistently named option for obtaining *make* script debugging information.

106534 Some historical implementations have not cleared the suffix list with `-r`.

106535 Implementations should be aware that some historical applications have intermixed *target\_name*  
106536 and *macro=value* operands on the command line, expecting that all of the macros are processed  
106537 before any of the targets are dealt with. Conforming applications do not do this, but some  
106538 backwards-compatibility support may be warranted.

106539 Empty inference rules are specified with a `<semicolon>` command rather than omitting all  
106540 commands, as described in an early proposal. The latter case has no traditional meaning and is  
106541 reserved for implementation extensions, such as in GNU *make*.

106542 Earlier versions of this standard defined comment lines only as lines with `'#'` as the first  
106543 character. Many places then talked about comments, blank lines, and empty lines; but some  
106544 places inadvertently only mentioned comments when blank lines and empty lines had also been  
106545 accepted in all known implementations. The standard now defines comment lines to be blank  
106546 lines, empty lines, and lines starting with a `'#'` character and explicitly lists cases where blank  
106547 lines and empty lines are not acceptable.

106548 On most historic systems, the *make* utility considered a target with a prerequisite that had an  
106549 identical timestamp as up-to-date. One implementation of *make* treated it as out-of-date. Note  
106550 that up-to-date and out-of-date are antonyms. The standard now allows either behavior, but

106551 implementations are encouraged to treat such targets as out-of-date. This is especially important  
 106552 on file systems where the timestamp resolution is the minimum (1 second) required by the  
 106553 standard. All implementations of *make* should make full use of the finest timestamp resolution  
 106554 available on the file systems holding targets and prerequisites to ensure that targets are up-to-  
 106555 date even for prerequisite files with timestamps that were updated within the same second.  
 106556 However, if the timestamp resolutions of the file systems containing a target and a prerequisite  
 106557 are different, the timestamp with the more precise resolution should be rounded down to the  
 106558 resolution of the less precise timestamp for the comparison.

106559 The traditional semantics of delayed-expansion macros have often been the source of subtle  
 106560 bugs for makefile writers not aware of those semantics. Furthermore, in implementations that  
 106561 support an extension of assigning the output of an arbitrary command to a macro definition, the  
 106562 use of delayed-expansion macros could result in an undesirable growth in execution time, as  
 106563 each use of the macro would re-run the arbitrary command. Historically, several  
 106564 implementations independently developed a form of immediate expansion, usually via the  
 106565 operator `:=`, so that execution of an arbitrary command happens once at the definition of the  
 106566 macro rather than each use of the macro; however, there are subtle differences in the expansion  
 106567 rules of those various implementations when the expanded value of *string2* contained a '\$'.  
 106568 Other implementations used the operator `:=` for conditional expansion, altogether unrelated  
 106569 to immediate-expansion macro definition.

106570 The standard developers felt that immediate-expansion semantics were useful enough to  
 106571 standardize, but requiring the semantics of any one implementation of `:=` would cause  
 106572 confusion in makefiles written for other implementation semantics, necessitating a reader to  
 106573 determine if `.POSIX:` had been specified at the beginning of the file (or worse, at the beginning  
 106574 of some other file that then includes the fragment in question) to know which semantics would  
 106575 be in use. Therefore, the standard developers opted to require two new operators, `:=` and  
 106576 `::=`, with specific semantics; the `::=` operator has semantics closest to the GNU *make*  
 106577 implementation of `:=`, where '\$' characters occurring in the immediate expansion of *string2*  
 106578 are not further expanded in subsequent use of the macro, and the `::=` operator has  
 106579 semantics closest to the BSD *make* and *smake* implementations of `:=`, where immediate  
 106580 expansion is performed when assigning to a delayed-expansion macro and '\$\$' is preserved. It  
 106581 was felt that other implementations could easily support the required semantics.

106582 Implementations that previously provided `:=` as an extension are encouraged to leave this  
 106583 extension intact, with no change in the implementation's particular semantics, to avoid breaking  
 106584 non-portable makefiles that had been targeting that particular implementation. A portable  
 106585 makefile, with `.POSIX:` specified at the beginning, should not use the `:=` operator.

106586 Traditionally, constructs such as

```
106587 DIR: FORCE
106588     (commands)
106589 FORCE:
```

106590 were used to allow `make DIR` to always run `(commands)`; however, this depended on the user  
 106591 never creating a file named **FORCE**. The addition of the **.PHONY** special target provides a more  
 106592 efficient manner of providing a target whose commands are always run, and where the user  
 106593 cannot create a file that influences the behavior in an unexpected manner.

106594 This standard allows two different methods of creating include files or bringing them up-to-  
 106595 date, reflecting established practice in SunPro *make* and GNU *make*. The former performs this  
 106596 action during parsing, before the include file is opened. The latter delays performing the action  
 106597 until after all makefiles have been read. Implementors who opt for the "delayed remaking"  
 106598 method should be aware of the following potential issues:

- 106599 • Diagnostic messages about missing include files must be deferred until the final exit status  
106600 is known. Note that this is a conformance issue, not just a quality of implementation issue.
- 106601 • If the way *make* handles using updated include file contents is to start over after include  
106602 files have been made up-to-date, it is possible for a poorly written makefile to cause *make*  
106603 to enter a sequence of restarts where nothing changes each time, resulting in the sequence  
106604 continuing indefinitely unless the situation is detected. Implementors are encouraged to  
106605 include a mechanism for detecting and reporting this, rather than allowing *make* to  
106606 consume an arbitrary amount of system resource until it is forcibly terminated.
- 106607 • If *make* uses this start-over method, makefile contents read from a pipe on standard input  
106608 or from a FIFO must be copied to a temporary file, and when *make* starts over it must use  
106609 this file instead.
- 106610 • If *make* starts over by executing itself using the *exec* family of functions, the need to replace  
106611 ' - ' or the pathnames of FIFOs with the pathnames of temporary files can lead to the *exec*  
106612 call failing with an [E2BIG] error if the original execution was close to the {ARG\_MAX}  
106613 limit. Although this is a quality of implementation issue, not a conformance issue (since  
106614 the general rules for utility errors allow utilities to fail when they encounter a variety of  
106615 internal errors - see Section 1.4, on page 2462), implementors are encouraged to explore  
106616 ways to prevent it, such as passing information via a temporary file instead of on the  
106617 command line when an [E2BIG] error has occurred. Another solution might be to jump  
106618 (e.g. using *siglongjmp()*) back to the start of *main()* as the way to start over. Making a  
106619 recursive call to *main()* is not recommended, as that would run into the stack limit if  
106620 sufficiently many restarts are needed.

106621 This standard specifies that a non-existent include file is first created if possible, and only if not  
106622 possible can other directories be searched. Historical versions of GNU *make* first searched the  
106623 include directories, then attempted to create the include file. This behavior was not considered  
106624 suitable for standardization as it means writers of portable applications have to use absolute  
106625 pathnames for all include files that need to be created via a rule (because they can never be sure  
106626 what relative pathnames are safe to use, since a file with the same relative pathname might  
106627 happen to exist in one of the searched directories when installing the application on a new  
106628 system). Note, however, that this only applies to directories searched by default. If an  
106629 application uses an extension to specify that one or more directories are searched, this standard  
106630 does not place any constraints on when the specified directories are searched.

106631 This standard specifies a way for portable applications to request parallel updating of targets  
106632 with commands by using the `-j maxjobs` option. This feature is described in terms of a token pool  
106633 initially containing up to `maxjobs - 1` tokens. Note that this is not intended to prescribe a  
106634 particular implementation design; the usual "as if" rule applies.

106635 Implementations are permitted to silently limit the pool size for a few reasons, including:

- 106636 • Implementations that do not support parallelism can support the `-j` option by simply  
106637 ignoring the option (other than passing it to *sub-make* invocations via the `MAKEFLAGS`  
106638 environment variable). In effect, such an implementation silently restricts the size of the  
106639 token pool to zero (and therefore need not create a token pool).
- 106640 • Some historical implementations dynamically limit the token pool size based on the  
106641 current system load to avoid overloading the system.
- 106642 • Implementations may want to limit the token pool size based on the number of processors  
106643 available.

106644 • Implementations may want to limit the token pool size based on resource limits.

106645 Limiting the pool size does not change the value of *maxjobs* that is passed to sub-*make*  
106646 invocations via the *MAKEFLAGS* environment variable.

106647 When a different *maxjobs* value is passed to a sub-*make*, some historical *make* implementations  
106648 created a separate pool of tokens while other historical *make* implementations continued to  
106649 obtain tokens from the invoking *make* but limited the number of tokens held at a time to the new  
106650 value of *maxjobs* – 1. Both behaviors are believed to have merit in different situations: the former  
106651 gives a sub-*make* complete control the amount of parallelism, while the latter allows the user to  
106652 control the overall system load. This standard permits either behavior.

106653 This standard calls for a token pool of size *maxjobs* – 1, and for removal from that pool only for  
106654 the second and subsequent tasks in a set of parallel tasks. This design was chosen because this is  
106655 effectively what existing implementations do, and also because the token consumed by a parallel  
106656 task that invokes a sub-*make* is effectively lent to the sub-*make*. Lending the token to the sub-  
106657 *make* has the following advantages:

106658 • It prevents the sub-*make* from being completely idle due to token starvation, allowing it to  
106659 always make some progress regardless of how many tokens other sub-*make* invocations  
106660 have consumed.

106661 • It prevents token pool exhaustion caused by a long chain of sub-*make* invocations. If the  
106662 token consumed by the invoking rule was not effectively lent to the sub-*make*, then the  
106663 pool would be exhausted by a chain of sub-*make* invocations that is *maxjobs* long. Such a  
106664 chain would never accomplish any work, and would thus never complete.

106665 When a rule invokes multiple sub-*make* processes asynchronously (for example by using an  
106666 asynchronous list in the shell), some implementations allow each sub-*make* to execute at least  
106667 one rule even though this would cause the total number of parallel rule executions across all  
106668 *make* instances to exceed *maxjobs* (after discounting the rules that execute sub-*make* processes).  
106669 This behavior may not be ideal, but it is easier to implement and is unlikely to cause problems in  
106670 practice because applications typically do not have any rules that invoke multiple sub-*make*  
106671 processes asynchronously. For this reason the behavior is unspecified if a rule executes multiple  
106672 sub-*make* processes asynchronously.

106673 When multiple sub-*make* processes are running in parallel there is no requirement placed on the  
106674 ordering of output from these processes. Some implementations of *make* attempt to serialize  
106675 output from each sub-*make*; others make no such attempt. If diagnostic messages from failed  
106676 commands are intermixed, the usual way to deal with this is to repeat the *make* without *-j* (or  
106677 with *-j 1*) so that intermixing will not occur.

#### 106678 FUTURE DIRECTIONS

106679 If this utility is directed to create a new directory entry that contains any bytes that have the  
106680 encoded value of a <newline> character, implementations are encouraged to treat this as an  
106681 error. A future version of this standard may require implementations to treat this as an error.

106682 Some implementations of *make* include an *export* directive to add specified *make* variables to the  
106683 environment. This may be considered for standardization in a future version.

106684 A future version of this standard may add a command-line option that causes *make* to report  
106685 attempts to expand (or append to) macros that do not exist.

106686 A future version of this standard may require that a target with a prerequisite with an identical  
106687 timestamp is considered out-of-date.



106688 **SEE ALSO**

- 106689 Chapter 2 (on page 2472), *ar*, *c17*, *get*, *lex*, *sccs*, *sh*, *yacc*
- 106690 XBD Section 6.1 (on page 117), Chapter 8 (on page 167), Section 12.2 (on page 215)
- 106691 XSH *exec*, *system()*

106692 **CHANGE HISTORY**

- 106693 First released in Issue 2.

106694 **Issue 5**

- 106695 The FUTURE DIRECTIONS section is added.

106696 **Issue 6**

- 106697 This utility is marked as part of the Software Development Utilities option.
- 106698 The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the  
106699 SPECIAL TARGETS section.
- 106700 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from  
106701 ``otherwise, the home directory of a user of that name is examined'' to ``otherwise, the value of  
106702 *PROJECTDIR* is treated as a user name and that user's initial working directory is examined''.
- 106703 It is specified whether the command line is related to the makefile or to the *make* command, and  
106704 the macro processing rules are updated to align with the IEEE P1003.2b draft standard.
- 106705 The normative text is reworded to avoid use of the term ``must'' for application requirements.
- 106706 PASC Interpretation 1003.2 #193 is applied.

106707 **Issue 7**

- 106708 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
106709 apply.
- 106710 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 106711 Include lines in makefiles are introduced.
- 106712 Austin Group Interpretation 1003.1-2001 #131 is applied, changing the **Makefile Execution**  
106713 section.
- 106714 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0121 [257] is applied.
- 106715 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0122 [509], XCU/TC2-2008/0123  
106716 [584], XCU/TC2-2008/0124 [857], XCU/TC2-2008/0125 [505], XCU/TC2-2008/0126 [584],  
106717 XCU/TC2-2008/0127 [505], XCU/TC2-2008/0128 [865], XCU/TC2-2008/0129 [693],  
106718 XCU/TC2-2008/0130 [602], XCU/TC2-2008/0131 [848], XCU/TC2-2008/0132 [763],  
106719 XCU/TC2-2008/0133 [857], XCU/TC2-2008/0134 [866], XCU/TC2-2008/0135 [525],  
106720 XCU/TC2-2008/0136 [848], XCU/TC2-2008/0137 [769], XCU/TC2-2008/0138 [525],  
106721 XCU/TC2-2008/0139 [769], XCU/TC2-2008/0140 [505], XCU/TC2-2008/0141 [693],  
106722 XCU/TC2-2008/0142 [505], XCU/TC2-2008/0143 [857], and XCU/TC2-2008/0144 [693,865] are  
106723 applied.

106724 **Issue 8**

- 106725 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
106726 filenames containing any bytes that have the encoded value of a <newline> character.
- 106727 Austin Group Defects 330, 1417, 1422, 1709, and 1710 are applied, adding new forms of macro  
106728 assignment using the " := ", " ?=", and "+=" operators.
- 106729 Austin Group Defect 333 is applied, adding support for ``silent includes'' using **-include**.

- 106730 Austin Group Defects 336 and 1711 are applied, specifying the behavior when *string1* in a macro expansion contains a macro expansion.  
106731
- 106732 Austin Group Defect 337 is applied, adding a new form of macro assignment using the "!=" operator.  
106733
- 106734 Austin Group Defects 373 and 1417 are applied, changing the set of characters that portable applications can use in macro names to the entire portable filename character set (thus adding <hyphen-minus> to the set that could previously be used).  
106735  
106736
- 106737 Austin Group Defects 514 and 1520 are applied, adding the \$+ and \$^ internal macros.
- 106738 Austin Group Defect 518 is applied, allowing multiple files to be specified on an **include** line.
- 106739 Austin Group Defects 519, 1712, and 1715 are applied, adding support for pattern macro expansions.  
106740
- 106741 Austin Group Defects 523, 1708, and 1749 are applied, adding the **.PHONY** special target.
- 106742 Austin Group Defect 875 is applied, clarifying the requirements for inference rules.
- 106743 Austin Group Defect 1104 is applied, changing ``s2.a'' to ``.s2.a''.
- 106744 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 106745 Austin Group Defect 1141 is applied, changing ``core files'' to ``a file named core''.
- 106746 Austin Group Defect 1155 is applied, clarifying the handling of the *MAKE* macro.
- 106747 Austin Group Defect 1325 is applied, adding requirements relating to the creation of include files.  
106748
- 106749 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 106750 Austin Group Defect 1419 is applied, updating the **.SCCS\_GET** default rule.
- 106751 Austin Group Defect 1420 is applied, clarifying where internal macros can be used.
- 106752 Austin Group Defect 1421 is applied, changing the APPLICATION USAGE section.
- 106753 Austin Group Defects 1424, 1658, 1690, 1701, 1702, 1703, 1704, 1707, 1719, 1720, 1721, 1722, and 1750 are applied, making various minor editorial wording changes.  
106754
- 106755 Austin Group Defects 1436, 1437, 1652, 1660, 1661, and 1733 are applied, adding the **-j maxjobs** option and the **.NOTPARALLEL** and **.WAIT** special targets, and changing the **-n** option.  
106756
- 106757 Austin Group Defects 1471 and 1513 are applied, adding a new form of macro assignment using the " : : =" operator.  
106758
- 106759 Austin Group Defect 1479 is applied, clarifying the requirements for default rules and macro values.  
106760
- 106761 Austin Group Defect 1492 is applied, changing the EXIT STATUS section.
- 106762 Austin Group Defect 1505 is applied, clarifying the requirements for expansion of macros that do not exist.  
106763
- 106764 Austin Group Defect 1510 is applied, correcting a typographic error in the RATIONALE section.
- 106765 Austin Group Defect 1549 is applied, clarifying the requirements for an escaped <newline> in a command line.  
106766
- 106767 Austin Group Defect 1615 is applied, allowing target names to contain slashes and hyphens.
- 106768 Austin Group Defect 1626 is applied, adding the *CURDIR* macro.

- 106769 Austin Group Defect 1631 is applied, adding information about use of the `-j` option with the `.ca`  
106770 default rule to the APPLICATION USAGE and EXAMPLES sections.
- 106771 Austin Group Defect 1650 is applied, changing the few occurrences of “dependencies” to use the  
106772 more common “prerequisites”.
- 106773 Austin Group Defect 1653 is applied, clarifying the difference between how `MAKEFLAGS` is  
106774 parsed compared to shell commands that use the `make` utility.
- 106775 Austin Group Defects 1654 and 1655 are applied, changing the APPLICATION USAGE section.
- 106776 Austin Group Defect 1656 is applied, changing the NAME section.
- 106777 Austin Group Defect 1657 is applied, moving some requirements unrelated to makefile syntax  
106778 from the Makefile Syntax subsection to the beginning of the EXTENDED DESCRIPTION section.
- 106779 Austin Group Defect 1689 is applied, removing some redundant wording from the  
106780 DESCRIPTION section.
- 106781 Austin Group Defect 1692 is applied, allowing `make`, when invoked with the `-q` or `-t` option, to  
106782 execute command lines (without a `<plus-sign>` prefix) that expand the `MAKE` macro.
- 106783 Austin Group Defect 1693 is applied, changing “command lines” to “execution lines” in the  
106784 description of the `-s` option.
- 106785 Austin Group Defect 1694 is applied, changing “in the order they appear” to “in the order  
106786 specified” in the OPERANDS section.
- 106787 Austin Group Defect 1696 is applied, changing the STDOUT section.
- 106788 Austin Group Defect 1697 is applied, changing the RATIONALE and FUTURE DIRECTIONS  
106789 sections.
- 106790 Austin Group Defect 1698 is applied, changing “of a target” to “of the target” in the EXTENDED  
106791 DESCRIPTION section.
- 106792 Austin Group Defect 1699 is applied, addressing some inconsistencies in the use of the term  
106793 “rules”.
- 106794 Austin Group Defect 1706 is applied, removing a line from the format specified for target rules.
- 106795 Austin Group Defect 1714 is applied, changing “beginning of the line” to “beginning of the  
106796 value”.
- 106797 Austin Group Defect 1716 is applied, changing the typographic convention used for variable  
106798 elements within target names, in particular the inference rule suffixes `s1` and `s2`.
- 106799 Austin Group Defect 1723 is applied, adding historical context to a paragraph in the  
106800 RATIONALE section.
- 106801 Austin Group Defect 1772 is applied, clarifying the ASYNCHRONOUS EVENTS section.

106802 **NAME**

106803 man — display system documentation

106804 **SYNOPSIS**106805 UP `man [-k] name...`106806 **DESCRIPTION**

106807 The *man* utility shall write information about each of the *name* operands. If *name* is the name of a  
 106808 standard utility, *man* at a minimum shall write a message describing the syntax used by the  
 106809 standard utility, its options, operands, environment variables affecting its execution, and its list  
 106810 of exit status codes. If more information is available, the *man* utility shall provide it in an  
 106811 implementation-defined manner.

106812 An implementation may provide information for values of *name* other than the standard utilities.  
 106813 Standard utilities that are listed as optional and that are not supported by the implementation  
 106814 either shall cause a brief message indicating that fact to be displayed or shall cause a full display  
 106815 of information as described previously.

106816 **OPTIONS**106817 The *man* utility shall conform to XBD [Section 12.2](#) (on page 215).

106818 The following option shall be supported:

106819 **-k** Interpret *name* operands as keywords to be used in searching a utilities summary  
 106820 database that contains a brief purpose entry for each standard utility and write lines  
 106821 from the summary database that match any of the keywords. The keyword search shall  
 106822 produce results that are the equivalent of the output of the following command:

```
106823 grep -Ei '
106824 name
106825 name
106826 ...
106827 ' summary-database
```

106828 This assumes that the *summary-database* is a text file with a single entry per line; this  
 106829 organization is not required and the example using *grep -Ei* is merely illustrative of the  
 106830 type of search intended. The purpose entry to be included in the database shall consist  
 106831 of a terse description of the purpose of the utility.

106832 **OPERANDS**

106833 The following operand shall be supported:

106834 *name* A keyword or the name of a standard utility. When **-k** is not specified and *name*  
 106835 does not represent one of the standard utilities, the results are unspecified.

106836 **STDIN**

106837 Not used.

106838 **INPUT FILES**

106839 None.

106840 **ENVIRONMENT VARIABLES**106841 The following environment variables shall affect the execution of *man*:

106842 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 106843 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 106844 variables used to determine the values of locale categories.)

- 106845 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
106846 internationalization variables.
- 106847 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
106848 characters (for example, single-byte as opposed to multi-byte characters in  
106849 arguments and in the summary database). The value of *LC\_CTYPE* need not affect  
106850 the format of the information written about the *name* operands.
- 106851 *LC\_MESSAGES*  
106852 Determine the locale that should be used to affect the format and contents of  
106853 diagnostic messages written to standard error and informative messages written to  
106854 standard output.
- 106855 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 106856 *PAGER* Determine an output filtering command for writing the output to a terminal. Any  
106857 string acceptable as a *command\_string* operand to the *sh -c* command shall be valid.  
106858 When standard output is a terminal device, the reference page output shall be  
106859 piped through the command. If the *PAGER* variable is null or not set, the command  
106860 shall be either *more* or another paginator utility documented in the system  
106861 documentation.
- 106862 **ASYNCHRONOUS EVENTS**  
106863 Default.
- 106864 **STDOUT**  
106865 The *man* utility shall write text describing the syntax of the utility *name*, its options and its  
106866 operands, or, when *-k* is specified, lines from the summary database. The format of this text is  
106867 implementation-defined.
- 106868 **STDERR**  
106869 The standard error shall be used for diagnostic messages, and may also be used for  
106870 informational messages of unspecified format.
- 106871 **OUTPUT FILES**  
106872 None.
- 106873 **EXTENDED DESCRIPTION**  
106874 None.
- 106875 **EXIT STATUS**  
106876 The following exit values shall be returned:  
106877 0 Successful completion.  
106878 >0 An error occurred.
- 106879 **CONSEQUENCES OF ERRORS**  
106880 Default.

106881 **APPLICATION USAGE**

106882 None.

106883 **EXAMPLES**

106884 None.

106885 **RATIONALE**

106886 It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the  
 106887 standard developers was strongly divided as to how much or how little information *man* should  
 106888 be required to provide. They considered, however, that the provision of some portable way of  
 106889 accessing documentation would aid user portability. The arguments against a fuller specification  
 106890 were:

- 106891 • Large quantities of documentation should not be required on a system that does not have  
 106892 excess disk space.
- 106893 • The current manual system does not present information in a manner that greatly aids user  
 106894 portability.
- 106895 • A “better help system” is currently an area in which vendors feel that they can add value  
 106896 to their POSIX implementations.

106897 The `-f` option was considered, but due to implementation differences, it was not included in this  
 106898 volume of POSIX.1-2024.

106899 The description was changed to be more specific about what has to be displayed for a utility. The  
 106900 standard developers considered it insufficient to allow a display of only the synopsis without  
 106901 giving a short description of what each option and operand does.

106902 The “purpose” entry to be included in the database can be similar to the section title (less the  
 106903 numeric prefix) from this volume of POSIX.1-2024 for each utility. These titles are similar to  
 106904 those used in historical systems for this purpose.

106905 See *mailx* for rationale concerning the default paginator.

106906 The caveat in the *LC\_CTYPE* description was added because it is not a requirement that an  
 106907 implementation provide reference pages for all of its supported locales on each system;  
 106908 changing *LC\_CTYPE* does not necessarily translate the reference page into another language.  
 106909 This is equivalent to the current state of *LC\_MESSAGES* in POSIX.1-2024—locale-specific  
 106910 messages are not yet a requirement.

106911 The historical *MANPATH* variable is not included in POSIX because no attempt is made to  
 106912 specify naming conventions for reference page files, nor even to mandate that they are files at  
 106913 all. On some implementations they could be a true database, a hypertext file, or even fixed  
 106914 strings within the *man* executable. The standard developers considered the portability of  
 106915 reference pages to be outside their scope of work. However, users should be aware that  
 106916 *MANPATH* is implemented on a number of historical systems and that it can be used to tailor  
 106917 the search pattern for reference pages from the various categories (utilities, functions, file  
 106918 formats, and so on) when the system administrator reveals the location and conventions for  
 106919 reference pages on the system.

106920 The keyword search can rely on at least the text of the section titles from these utility  
 106921 descriptions, and the implementation may add more keywords. The term “section titles” refers  
 106922 to the strings such as:

106923 `man` – Display system documentation  
 106924 `ps` – Report process status

**106925 FUTURE DIRECTIONS**

106926 If this utility is directed to display a pathname that contains any bytes that have the encoded  
106927 value of a <newline> character when <newline> is a terminator or separator in the output  
106928 format being used, implementations are encouraged to treat this as an error. A future version of  
106929 this standard may require implementations to treat this as an error.

**106930 SEE ALSO**

106931 *more*

106932 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**106933 CHANGE HISTORY**

106934 First released in Issue 4.

**106935 Issue 5**

106936 The FUTURE DIRECTIONS section is added.

**106937 Issue 7**

106938 Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages  
106939 may appear on standard error.

**106940 Issue 8**

106941 Austin Group Defect 190 is applied, marking the *man* utility as part of the User Portability  
106942 Utilities option, and adding a requirement for the message it writes for a standard utility to  
106943 include the environment variables affecting its execution and its list of exit status codes.

106944 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
106945 directed to display a pathname that contains any bytes that have the encoded value of a  
106946 <newline> character when <newline> is a terminator or separator in the output format being  
106947 used.

106948 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

106949 **NAME**

106950 mesg — permit or deny messages

106951 **SYNOPSIS**106952 mesg [*y* | *n*]106953 **DESCRIPTION**

106954 The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or  
 106955 other utilities to a terminal device. The terminal device affected shall be determined by searching  
 106956 for the first terminal in the sequence of devices associated with standard input, standard output,  
 106957 and standard error, respectively. With no arguments, *mesg* shall report the current state without  
 106958 changing it. Processes with appropriate privileges may be able to send messages to the terminal  
 106959 independent of the current state.

106960 **OPTIONS**

106961 None.

106962 **OPERANDS**

106963 The following operands shall be supported in the POSIX locale:

106964 *y* Grant permission to other users to send messages to the terminal device.106965 *n* Deny permission to other users to send messages to the terminal device.106966 **STDIN**

106967 Not used.

106968 **INPUT FILES**

106969 None.

106970 **ENVIRONMENT VARIABLES**106971 The following environment variables shall affect the execution of *mesg*:

106972 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 106973 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 106974 variables used to determine the values of locale categories.)

106975 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 106976 internationalization variables.

106977 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 106978 characters (for example, single-byte as opposed to multi-byte characters in  
 106979 arguments).

106980 *LC\_MESSAGES*

106981 Determine the locale that should be used to affect the format and contents of  
 106982 diagnostic messages written (by *mesg*) to standard error.

106983 *XSI* *NLSPATH* Determine the location of messages objects and message catalogs.106984 **ASYNCHRONOUS EVENTS**

106985 Default.

106986 **STDOUT**106987 If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.106988 **STDERR**

106989 The standard error shall be used only for diagnostic messages.



106990 **OUTPUT FILES**

106991 None.

106992 **EXTENDED DESCRIPTION**

106993 None.

106994 **EXIT STATUS**

106995 The following exit values shall be returned:

106996 0 Receiving messages is allowed.

106997 1 Receiving messages is not allowed.

106998 &gt;1 An error occurred.

106999 **CONSEQUENCES OF ERRORS**

107000 Default.

107001 **APPLICATION USAGE**

107002 The mechanism by which the message status of the terminal is changed is unspecified.  
107003 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has  
107004 successfully completed. These actions may include, but are not limited to: another invocation of  
107005 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or  
107006 *chmod()* function, and so on.

107007 **EXAMPLES**

107008 None.

107009 **RATIONALE**

107010 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather  
107011 than the controlling terminal for the session. This is because users logged in more than once  
107012 should be able to change any of their login terminals without having to stop the job running in  
107013 those sessions. This is not a security problem involving the terminals of other users because  
107014 appropriate privileges would be required to affect the terminal of another user.

107015 The method of checking each of the first three file descriptors in sequence until a terminal is  
107016 found was adopted from System V.

107017 The file */dev/tty* is not specified for the terminal device because it was thought to be too  
107018 restrictive. Typical environment changes for the *n* operand are that write permissions are  
107019 removed for *others* and *group* from the appropriate device. It was decided to leave the actual  
107020 description of what is done as unspecified because of potential differences between  
107021 implementations.

107022 The format for standard output is unspecified because of differences between historical  
107023 implementations. This output is generally not useful to shell scripts (they can use the exit status),  
107024 so exact parsing of the output is unnecessary.

107025 **FUTURE DIRECTIONS**

107026 None.

107027 **SEE ALSO**107028 *talk*, *write*107029 [XBD Chapter 8](#) (on page 167)

107030 **CHANGE HISTORY**

107031 First released in Issue 2.

107032 **Issue 6**

107033 This utility is marked as part of the User Portability Utilities option.

107034 **Issue 7**

107035 The *msg* utility is moved from the User Portability Utilities option to the Base. User Portability  
107036 Utilities is now an option for interactive utilities.

107037 **Issue 8**

107038 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

107039 **NAME**

107040 mkdir — make directories

107041 **SYNOPSIS**107042 mkdir [-p] [-m *mode*] *dir*...107043 **DESCRIPTION**107044 The *mkdir* utility shall create the directories specified by the operands, in the order specified.107045 For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function  
107046 defined in the System Interfaces volume of POSIX.1-2024, called with the following arguments:

- 107047 1. The *dir* operand is used as the *path* argument.
- 107048 2. The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO is used as  
107049 the *mode* argument. (If the **-m** option is specified, the value of the *mkdir()* *mode* argument  
107050 is unspecified, but the directory shall at no time have permissions less restrictive than the  
107051 **-m mode** option-argument.)

107052 **OPTIONS**107053 The *mkdir* utility shall conform to XBD [Section 12.2](#) (on page 215).

107054 The following options shall be supported:

107055 **-m mode** Set the file permission bits of the newly-created directory to the specified *mode*  
107056 value. The *mode* option-argument shall be the same as the *mode* operand defined  
107057 for the *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-'  
107058 shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add  
107059 permissions to the default mode, '-' shall delete permissions from the default  
107060 mode.

107061 **-p** Create any missing intermediate pathname components.

107062 For each *dir* operand that does not name an existing directory, before performing  
107063 the actions described in the DESCRIPTION above, the *mkdir* utility shall create any  
107064 pathname components of the path prefix of *dir* that do not name an existing  
107065 directory by performing actions equivalent to first calling the *mkdir()* function with  
107066 the following arguments:

- 107067 1. A pathname naming the missing pathname component, ending with a  
107068 trailing <slash> character, as the *path* argument
- 107069 2. The value zero as the *mode* argument

107070 and then calling the *chmod()* function with the following arguments:

- 107071 1. The same *path* argument as in the *mkdir()* call
- 107072 2. The value  $(S\_IWUSR | S\_IXUSR | \sim filemask) \& 0777$  as the *mode*  
107073 argument, where *filemask* is the file mode creation mask of the process (see  
107074 XSH *umask()*)

107075 Each *dir* operand that names an existing directory shall be ignored without error.107076 **OPERANDS**

107077 The following operand shall be supported:

107078 *dir* A pathname of a directory to be created.

107079 **STDIN**

107080 Not used.

107081 **INPUT FILES**

107082 None.

107083 **ENVIRONMENT VARIABLES**107084 The following environment variables shall affect the execution of *mkdir*:

107085 *LANG* Provide a default value for the internationalization variables that are unset or null.  
107086 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
107087 variables used to determine the values of locale categories.)

107088 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
107089 internationalization variables.

107090 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
107091 characters (for example, single-byte as opposed to multi-byte characters in  
107092 arguments).

107093 *LC\_MESSAGES*

107094 Determine the locale that should be used to affect the format and contents of  
107095 diagnostic messages written to standard error.

107096 *XSI NLSPATH* Determine the location of messages objects and message catalogs.

107097 **ASYNCHRONOUS EVENTS**

107098 Default.

107099 **STDOUT**

107100 Not used.

107101 **STDERR**

107102 The standard error shall be used only for diagnostic messages.

107103 **OUTPUT FILES**

107104 None.

107105 **EXTENDED DESCRIPTION**

107106 None.

107107 **EXIT STATUS**

107108 The following exit values shall be returned:

107109 0 All the specified directories were created successfully, or the **-p** option was specified and all  
107110 the specified directories either already existed or were created successfully.

107111 >0 An error occurred.

107112 **CONSEQUENCES OF ERRORS**

107113 Default.

**107114 APPLICATION USAGE**

107115 The default file mode for directories is *a=rwx* (777 on most systems) with selected permissions  
107116 removed in accordance with the file mode creation mask. For intermediate pathname  
107117 components created by *mkdir*, the mode is the default modified by *u+rwx* so that the  
107118 subdirectories can always be created regardless of the file mode creation mask; if different  
107119 ultimate permissions are desired for the intermediate directories, they can be changed  
107120 afterwards with *chmod*.

107121 Note that some of the requested directories may have been created even if an error occurs.

**107122 EXAMPLES**

107123 None.

**107124 RATIONALE**

107125 The System V *-m* option was included to control the file mode.

107126 The System V *-p* option was included to create any needed intermediate directories and to  
107127 complement the functionality provided by *rmdir* for removing directories in the path prefix as  
107128 they become empty. Because no error is produced if any path component already exists, the *-p*  
107129 option is also useful to ensure that a particular directory exists.

107130 The functionality of *mkdir* is described substantially through a reference to the *mkdir()*  
107131 function in the System Interfaces volume of POSIX.1-2024. For example, by default, the mode of the  
107132 directory is affected by the file mode creation mask in accordance with the specified behavior of  
107133 the *mkdir()* function. In this way, there is less duplication of effort required for describing details  
107134 of the directory creation.

**107135 FUTURE DIRECTIONS**

107136 If this utility is directed to create a new directory entry that contains any bytes that have the  
107137 encoded value of a <newline> character, implementations are encouraged to treat this as an  
107138 error. A future version of this standard may require implementations to treat this as an error.

**107139 SEE ALSO**

107140 *chmod*, *rm*, *rmdir*, *umask*

107141 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

107142 XSH *mkdir()*, *umask()*

**107143 CHANGE HISTORY**

107144 First released in Issue 2.

**107145 Issue 5**

107146 The FUTURE DIRECTIONS section is added.

**107147 Issue 7**

107148 SD5-XCU-ERN-56 is applied, aligning the *-m* option with the IEEE P1003.2b draft standard to  
107149 clarify an ambiguity.

107150 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

107151 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0122 [161] is applied.

107152 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0145 [843] is applied.

**107153 Issue 8**

107154 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
107155 filenames containing any bytes that have the encoded value of a <newline> character.

107156

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

107157 **NAME**

107158 mkfifo — make FIFO special files

107159 **SYNOPSIS**107160 `mkfifo [-m mode] file...`107161 **DESCRIPTION**107162 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order  
107163 specified.107164 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo()* function  
107165 defined in the System Interfaces volume of POSIX.1-2024, called with the following arguments:

- 107166 1. The *file* operand is used as the *path* argument.
- 107167 2. The value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP,  
107168 S\_IROTH, and S\_IWOTH is used as the *mode* argument. (If the `-m` option is specified, the  
107169 value of the *mkfifo()* *mode* argument is unspecified, but the FIFO shall at no time have  
107170 permissions less restrictive than the `-m mode` option-argument.)

107171 **OPTIONS**107172 The *mkfifo* utility shall conform to XBD [Section 12.2](#) (on page 215).

107173 The following option shall be supported:

107174 `-m mode` Set the file permission bits of the newly-created FIFO to the specified *mode* value.  
107175 The *mode* option-argument shall be the same as the *mode* operand defined for the  
107176 *chmod* utility. In the *symbolic\_mode* strings, the *op* characters '+' and '-' shall be  
107177 interpreted relative to an assumed initial mode of *a=rw*.

107178 **OPERANDS**

107179 The following operand shall be supported:

107180 *file* A pathname of the FIFO special file to be created.107181 **STDIN**

107182 Not used.

107183 **INPUT FILES**

107184 None.

107185 **ENVIRONMENT VARIABLES**107186 The following environment variables shall affect the execution of *mkfifo*:

107187 *LANG* Provide a default value for the internationalization variables that are unset or null.  
107188 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
107189 variables used to determine the values of locale categories.)

107190 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
107191 internationalization variables.

107192 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
107193 characters (for example, single-byte as opposed to multi-byte characters in  
107194 arguments).

107195 *LC\_MESSAGES*

107196 Determine the locale that should be used to affect the format and contents of  
107197 diagnostic messages written to standard error.

- 107198 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 107199 **ASYNCHRONOUS EVENTS**
- 107200 Default.
- 107201 **STDOUT**
- 107202 Not used.
- 107203 **STDERR**
- 107204 The standard error shall be used only for diagnostic messages.
- 107205 **OUTPUT FILES**
- 107206 None.
- 107207 **EXTENDED DESCRIPTION**
- 107208 None.
- 107209 **EXIT STATUS**
- 107210 The following exit values shall be returned:
- 107211 0 All the specified FIFO special files were created successfully.
- 107212 >0 An error occurred.
- 107213 **CONSEQUENCES OF ERRORS**
- 107214 Default.
- 107215 **APPLICATION USAGE**
- 107216 None.
- 107217 **EXAMPLES**
- 107218 None.
- 107219 **RATIONALE**
- 107220 This utility was added to permit shell applications to create FIFO special files.
- 107221 The **-m** option was added to control the file mode, for consistency with the similar functionality provided by the *mkdir* utility.
- 107222
- 107223 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate directories leading up to the FIFO specified by the final component. This was removed because it is not commonly needed and is not common practice with similar utilities.
- 107224
- 107225
- 107226 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function in the System Interfaces volume of POSIX.1-2024. For example, by default, the mode of the FIFO file is affected by the file mode creation mask in accordance with the specified behavior of the *mkfifo()* function. In this way, there is less duplication of effort required for describing details of the file creation.
- 107227
- 107228
- 107229
- 107230
- 107231 **FUTURE DIRECTIONS**
- 107232 If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.
- 107233
- 107234
- 107235 **SEE ALSO**
- 107236 *chmod*, *umask*
- 107237 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)
- 107238 XSH *mkfifo()*



107239 **CHANGE HISTORY**

107240 First released in Issue 3.

107241 **Issue 6**

107242 The **-m** option is aligned with the IEEE P1003.2b draft standard to clarify an ambiguity.

107243 **Issue 8**

107244 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

107246 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

107247 **NAME**107248 `more` — display files on a page-by-page basis107249 **SYNOPSIS**107250 UP `more [-ceisu] [-n number] [-p command] [-t tagstring] [file...]`107251 **DESCRIPTION**

107252 The *more* utility shall read files and either write them to the terminal on a page-by-page basis or  
 107253 filter them to standard output. If standard output is not a terminal device, all input files shall be  
 107254 copied to standard output in their entirety, without modification, except as specified for the `-s`  
 107255 option. If standard output is a terminal device, the files shall be written a number of lines (one  
 107256 screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION  
 107257 section.

107258 Certain block-mode terminals do not have all the capabilities necessary to support the complete  
 107259 *more* definition; they are incapable of accepting commands that are not terminated with a  
 107260 <newline>. Implementations that support such terminals shall provide an operating mode to  
 107261 *more* in which all commands can be terminated with a <newline> on those terminals. This mode:

- 107262 • Shall be documented in the system documentation
- 107263 • Shall, at invocation, inform the user of the terminal deficiency that requires the <newline>  
 107264 usage and provide instructions on how this warning can be suppressed in future  
 107265 invocations
- 107266 • Shall not be required for implementations supporting only fully capable terminals
- 107267 • Shall not affect commands already requiring <newline> characters
- 107268 • Shall not affect users on the capable terminals from using *more* as described in this volume  
 107269 of POSIX.1-2024

107270 **OPTIONS**

107271 The *more* utility shall conform to XBD [Section 12.2](#) (on page 215), except that '+' may be  
 107272 recognized as an option delimiter as well as '-'.

107273 The following options shall be supported:

- 107274 `-c` If a screen is to be written that has no lines in common with the current screen, or  
 107275 *more* is writing its first screen, *more* shall not scroll the screen, but instead shall  
 107276 redraw each line of the screen in turn, from the top of the screen to the bottom. In  
 107277 addition, if *more* is writing its first screen, the screen shall be cleared. This option  
 107278 may be silently ignored on devices with insufficient terminal capabilities.
- 107279 `-e` Exit immediately after writing the last line of the last file in the argument list; see  
 107280 the EXTENDED DESCRIPTION section.
- 107281 `-i` Perform pattern matching in a case-insensitive manner; see XBD [Section 9.2](#) (on  
 107282 page 180).
- 107283 `-n number` Specify the number of lines per screenful. The *number* argument is a positive  
 107284 decimal integer. The `-n` option shall override any values obtained from any other  
 107285 source.
- 107286 `-p command` Each time a screen from a new file is displayed or redisplayed (including as a  
 107287 result of *more* commands; for example, `:p`), execute the *more* command(s) in the  
 107288 command arguments in the order specified, as if entered by the user after the first  
 107289 screen has been displayed. No intermediate results shall be displayed (that is, if the  
 107290 command is a movement to a screen different from the normal first screen, only the

- 107291 screen resulting from the command shall be displayed.) If any of the commands  
 107292 fail for any reason, an informational message to this effect shall be written, and no  
 107293 further commands specified using the **-p** option shall be executed for this file.
- 107294 **-s** Behave as if consecutive empty lines were a single empty line.
- 107295 **-t tagstring** Write the screenful of the file containing the tag named by the *tagstring*  
 107296 argument. See the *ctags* utility. The tags feature represented by **-t tagstring** and the **:t**  
 107297 command is optional. It shall be provided on any system that also provides a  
 107298 conforming implementation of *ctags*; otherwise, the use of **-t** produces undefined  
 107299 results.
- 107300 The filename resulting from the **-t** option shall be logically added as a prefix to the  
 107301 list of command line files, as if specified by the user. If the tag named by the  
 107302 *tagstring* argument is not found, it shall be an error, and *more* shall take no further  
 107303 action.
- 107304 If the tag specifies a line number, the first line of the display shall contain the  
 107305 beginning of that line. If the tag specifies a pattern, the first line of the display shall  
 107306 contain the beginning of the matching text from the first line of the file that  
 107307 contains that pattern. If the line does not exist in the file or matching text is not  
 107308 found, an informational message to this effect shall be displayed, and *more* shall  
 107309 display the default screen as if **-t** had not been specified.
- 107310 If both the **-t tagstring** and **-p command** options are given, the **-t tagstring** shall be  
 107311 processed first; that is, the file and starting line for the display shall be as specified  
 107312 by **-t**, and then the **-p more** command shall be executed. If the line (matching text)  
 107313 specified by the **-t** command does not exist (is not found), no **-p more** command  
 107314 shall be executed for this file at any time.
- 107315 **-u** Treat a `<backspace>` as a printable control character, displayed as an  
 107316 implementation-defined character sequence (see the EXTENDED DESCRIPTION  
 107317 section), suppressing backspacing and the special handling that produces  
 107318 underlined or standout mode text on some terminal types. Also, do not ignore a  
 107319 `<carriage-return>` at the end of a line.

#### 107320 OPERANDS

107321 The following operand shall be supported:

- 107322 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
 107323 shall be used. If a *file* is '-', the standard input shall be read at that point in the  
 107324 sequence.

#### 107325 STDIN

107326 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.

#### 107327 INPUT FILES

107328 The input files being examined shall be text files. If standard output is a terminal, standard error  
 107329 shall be used to read commands from the user. If standard output is a terminal, standard error is  
 107330 not readable, and command input is needed, *more* may attempt to obtain user commands from  
 107331 the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error  
 107332 indicating that it was unable to read user commands. If standard output is not a terminal, no  
 107333 error shall result if standard error cannot be opened for reading.

107334 **ENVIRONMENT VARIABLES**

107335 The following environment variables shall affect the execution of *more*:

107336 *COLUMNS* Override the system-selected horizontal display line size. See XBD [Chapter 8](#) (on  
107337 page 167) for valid values and results when it is unset or null.

107338 *EDITOR* Used by the *v* command to select an editor. See the EXTENDED DESCRIPTION  
107339 section.

107340 *LANG* Provide a default value for the internationalization variables that are unset or null.  
107341 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
107342 variables used to determine the values of locale categories.)

107343 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
107344 internationalization variables.

107345 *LC\_COLLATE*

107346 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
107347 character collating elements within regular expressions.

107348 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
107349 characters (for example, single-byte as opposed to multi-byte characters in  
107350 arguments and input files) and the behavior of character classes within regular  
107351 expressions.

107352 *LC\_MESSAGES*

107353 Determine the locale that should be used to affect the format and contents of  
107354 diagnostic messages written to standard error and informative messages written to  
107355 standard output.

107356 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

107357 *LINES* Override the system-selected vertical screen size, used as the number of lines in a  
107358 screenful. See XBD [Chapter 8](#) (on page 167) for valid values and results when it is  
107359 unset or null. The *-n* option shall take precedence over the *LINES* variable for  
107360 determining the number of lines in a screenful.

107361 *MORE* Determine a string containing options described in the OPTIONS section preceded  
107362 with <hyphen-minus> characters and <blank>-separated as on the command line.  
107363 Any command line options shall be processed after those in the *MORE* variable, as  
107364 if the command line were:

107365 *more* \$MORE *options operands*

107366 The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for  
107367 determining the number of lines in a screenful.

107368 *TERM* Determine the name of the terminal type. If this variable is unset or null, an  
107369 unspecified default terminal type is used.

107370 **ASYNCHRONOUS EVENTS**

107371 The following actions shall be taken upon receipt of signals:

107372 *SIGCONT* The actions described below for *SIGWINCH* shall be taken, except that the screen  
107373 shall always be refreshed (regardless of whether the terminal window size  
107374 changed).

107375 *SIGWINCH* If standard output is a terminal, the current terminal window size associated with  
107376 the terminal on standard output shall be obtained, as if by a call to XSH  
107377 *tcgetwinsize()*. If the terminal window size is successfully obtained, it shall be used

107378 as follows:

- 107379 • If the *COLUMNS* environment variable is unset or does not contain a
- 107380 number, the horizontal display line size shall be set to the number of columns
- 107381 in the obtained terminal window size.
- 107382 • If the *-n* option was not specified (neither on the command line nor via the
- 107383 *MORE* environment variable) and the *LINES* environment variable is unset
- 107384 or does not contain a number, the vertical screen size shall be set to the
- 107385 number of rows in the obtained terminal window size.

107386 If the above resulted in either the vertical screen size or the horizontal display line  
 107387 size (or both) changing to a different value, the number of lines available per  
 107388 screen and the number of columns available per line shall be updated  
 107389 correspondingly (see XBD Chapter 8, on page 167) and the screen shall be  
 107390 refreshed; otherwise, the screen may be refreshed.

107391 The action taken for all other signals shall be the default.

#### 107392 **STDOUT**

107393 The standard output shall be used to write the contents of the input files.

#### 107394 **STDERR**

107395 The standard error shall be used for diagnostic messages and user commands (see the INPUT  
 107396 FILES section), and, if standard output is a terminal device, to write a prompting string. The  
 107397 prompting string shall appear on the screen line below the last line of the file displayed in the  
 107398 current screenful. The prompt shall contain the name of the file currently being examined and  
 107399 shall contain an end-of-file indication and the name of the next file, if any, when prompting at  
 107400 the end-of-file. If an error or informational message is displayed, it is unspecified whether it is  
 107401 contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the  
 107402 user shall be prompted for a continuation character, at which point another message or the user  
 107403 prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether  
 107404 informational messages are written for other user commands.

#### 107405 **OUTPUT FILES**

107406 None.

#### 107407 **EXTENDED DESCRIPTION**

107408 The following section describes the behavior of *more* when the standard output is a terminal  
 107409 device. If the standard output is not a terminal device, no options other than *-s* shall have any  
 107410 effect, and all input files shall be copied to standard output otherwise unmodified, at which time  
 107411 *more* shall exit without further action.

107412 The number of lines available per screen shall be determined by the *-n* option, if present, or by  
 107413 obtaining the vertical screen size from the *LINES* environment variable (see the  
 107414 ENVIRONMENT VARIABLES section) or from the terminal window size associated with the  
 107415 terminal on standard output (see XSH *tcgetwinsize()*), with a default value as described in XBD  
 107416 Chapter 8 (on page 167).

107417 The maximum number of lines written shall be one less than this number, because the screen  
 107418 line after the last line written shall be used to write a user prompt and user input. If the number  
 107419 of lines in the screen is less than two, the results are undefined. It is unspecified whether user  
 107420 input is permitted to be longer than the remainder of the single line where the prompt has been  
 107421 written.

107422 The number of columns available per line shall be determined by obtaining the horizontal  
 107423 display line size from the *COLUMNS* environment variable (see the ENVIRONMENT

107424 VARIABLES section) or from the terminal window size associated with the terminal on standard  
 107425 output (see XSH *tcgetwinsize()*), with a default value as described in XBD Chapter 8 (on page  
 107426 167).

107427 Lines that are longer than the display shall be folded; the length at which folding occurs is  
 107428 unspecified, but should be appropriate for the output device. Folding may occur between glyphs  
 107429 of single characters that take up multiple display columns.

107430 When standard output is a terminal and `-u` is not specified, *more* shall treat `<backspace>` and  
 107431 `<carriage-return>` characters specially:

- 107432 • A character, followed first by a sequence of *n* `<backspace>` characters (where *n* is the same  
 107433 as the number of column positions that the character occupies), then by *n* `<underscore>`  
 107434 characters (`'_'`), shall cause that character to be written as underlined text, if the terminal  
 107435 type supports that. The *n* `<underscore>` characters, followed first by *n* `<backspace>`  
 107436 characters, then any character with *n* column positions, shall also cause that character to be  
 107437 written as underlined text, if the terminal type supports that.
- 107438 • A sequence of *n* `<backspace>` characters (where *n* is the same as the number of column  
 107439 positions that the previous character occupies) that appears between two identical  
 107440 printable characters shall cause the first of those two characters to be written as  
 107441 emboldened text (that is, visually brighter, standout mode, or inverse-video mode), if the  
 107442 terminal type supports that, and the second to be discarded. Immediately subsequent  
 107443 occurrences of `<backspace>/character` pairs for that same character shall also be  
 107444 discarded. (For example, the sequence `"a\ba\ba\ba"` is interpreted as a single  
 107445 emboldened `'a'`.)
- 107446 • The *more* utility shall logically discard all other `<backspace>` characters from the line as  
 107447 well as the character which precedes them, if any.
- 107448 • A `<carriage-return>` at the end of a line shall be ignored, rather than being written as a  
 107449 non-printable character, as described in the next paragraph.

107450 It is implementation-defined how other non-printable characters are written. Implementations  
 107451 should use the same format that they use for the *ex print* command; see the OPTIONS section  
 107452 within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it  
 107453 crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the  
 107454 number of columns on the display is less than the number of columns any single character in the  
 107455 line being displayed would occupy.

107456 When each new file is displayed (or redisplayed), *more* shall write the first screen of the file.  
 107457 Once the initial screen has been written, *more* shall prompt for a user command. If the execution  
 107458 of the user command results in a screen that has lines in common with the current screen, and  
 107459 the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is  
 107460 unspecified whether the screen is scrolled or redrawn.

107461 For all files but the last (including standard input if no file was specified, and for the last file as  
 107462 well, if the `-e` option was not specified), when *more* has written the last line in the file, *more* shall  
 107463 prompt for a user command. This prompt shall contain the name of the next file as well as an  
 107464 indication that *more* has reached end-of-file. If the user command is `f`, `<control>-F`, `<space>`, `j`,  
 107465 `<newline>`, `d`, `<control>-D`, or `s`, *more* shall display the next file. Otherwise, if displaying the last  
 107466 file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

107467 Several of the commands described in this section display a previous screen from the input  
 107468 stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is  
 107469 implementation-defined how much backwards motion is supported. If a command cannot be  
 107470 executed because of a limitation on backwards motion, an error message to this effect shall be

107471 displayed, the current screen shall not change, and the user shall be prompted for another  
107472 command.

107473 If a command cannot be performed because there are insufficient lines to display, *more* shall alert  
107474 the terminal. If a command cannot be performed because there are insufficient lines to display or  
107475 a / command fails: if the input is the standard input, the last screen in the file may be displayed;  
107476 otherwise, the current file and screen shall not change, and the user shall be prompted for  
107477 another command.

107478 The interactive commands in the following sections shall be supported. Some commands can be  
107479 preceded by a decimal integer, called *count* in the following descriptions. If not specified with  
107480 the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular  
107481 expression, as described in XBD [Section 9.3](#) (on page 181). The term “examine” is historical  
107482 usage meaning “open the file for viewing”; for example, *more foo* would be expressed as  
107483 examining file **foo**.

107484 In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a  
107485 line from the file being examined.

107486 In the following descriptions, the *current position* refers to two things:

- 107487 1. The position of the current line on the screen
- 107488 2. The line number (in the file) of the current line on the screen

107489 Usually, the line on the screen corresponding to the current position is the third line on the  
107490 screen. If this is not possible (there are fewer than three lines to display or this is the first page of  
107491 the file, or it is the last page of the file), then the current position is either the first or last line on  
107492 the screen as described later.

## 107493 Help

107494 *Synopsis:*     h

107495 Write a summary of these commands and other implementation-defined commands. The  
107496 behavior shall be as if the *more* utility were executed with the `-e` option on a file that contained  
107497 the summary information. The user shall be prompted as described earlier in this section when  
107498 end-of-file is reached. If the user command is one of those specified to continue to the next file,  
107499 *more* shall return to the file and screen state from which the **h** command was executed.

## 107500 Scroll Forward One Screenful

107501 *Synopsis:*     [*count*]f  
107502                [*count*]<control>-F

107503 Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size,  
107504 only the final screenful shall be written.

## 107505 Scroll Backward One Screenful

107506 *Synopsis:*     [*count*]b  
107507                [*count*]<control>-B

107508 Scroll backward *count* lines, with a default of one screenful (see the `-n` option). If *count* is more  
107509 than the screen size, only the final screenful shall be written.

107510 **Scroll Forward One Line**

107511 *Synopsis:*    [*count*] <space>  
 107512            [*count*] j  
 107513            [*count*] <newline>

107514 Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for j and  
 107515 <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen  
 107516 size.

107517 **Scroll Backward One Line**

107518 *Synopsis:*    [*count*] k

107519 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the  
 107520 screen size.

107521 **Scroll Forward One Half Screenful**

107522 *Synopsis:*    [*count*] d  
 107523            [*count*] <control>-D

107524 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
 107525 shall become the new default for subsequent **d**, <control>-D, and **u** commands.

107526 **Skip Forward One Line**

107527 *Synopsis:*    [*count*] s

107528 Display the screenful beginning with the line *count* lines after the last line on the current screen.  
 107529 If *count* would cause the current position to be such that less than one screenful would be  
 107530 written, the last screenful in the file shall be written.

107531 **Scroll Backward One Half Screenful**

107532 *Synopsis:*    [*count*] u  
 107533            [*count*] <control>-U

107534 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it  
 107535 shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands.  
 107536 The entire *count* lines shall be written, even if *count* is more than the screen size.

107537 **Go to Beginning of File**

107538 *Synopsis:*    [*count*] g

107539 Display the screenful beginning with line *count*.

107540 **Go to End-of-File**

107541 *Synopsis:*    [*count*] G

107542 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the  
 107543 last screenful of the file.



**107544 Refresh the Screen**

107545 *Synopsis:*     r  
107546                   <control>-L

107547 Refresh the screen.

**107548 Discard and Refresh**

107549 *Synopsis:*     R

107550 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered  
107551 input shall not be discarded and the **R** command shall be equivalent to the **r** command.

**107552 Mark Position**

107553 *Synopsis:*     m*letter*

107554 Mark the current position with the letter named by *letter*, where *letter* represents the name of one  
107555 of the lowercase letters of the portable character set. When a new file is examined, all marks may  
107556 be lost.

**107557 Return to Mark**

107558 *Synopsis:*     '*letter*

107559 Return to the position that was previously marked with the letter named by *letter*, making that  
107560 line the current position.

**107561 Return to Previous Position**

107562 *Synopsis:*     ''

107563 Return to the position from which the last large movement command was executed (where a  
107564 "large movement" is defined as any movement of more than a screenful of lines). If no such  
107565 movements have been made, return to the beginning of the file.

**107566 Search Forward for Pattern**

107567 *Synopsis:*     [*count*]/[!]*pattern*<newline>

107568 Display the screenful beginning with the *count*th line containing the pattern. The search shall  
107569 start after the first line currently displayed. The null regular expression ('/' followed by a  
107570 <newline>) shall repeat the search using the previous regular expression, with a default *count*. If  
107571 the character '!' is included, the matching lines shall be those that do not contain the *pattern*. If  
107572 no match is found for the *pattern*, a message to that effect shall be displayed.

**107573 Search Backward for Pattern**

107574 *Synopsis:*     [*count*?][!]*pattern*<newline>

107575 Display the screenful beginning with the *count*th previous line containing the pattern. The search  
107576 shall start on the last line before the first line currently displayed. The null regular expression  
107577 ('?' followed by a <newline>) shall repeat the search using the previous regular expression,  
107578 with a default *count*. If the character '!' is included, matching lines shall be those that do not  
107579 contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be  
107580 displayed.

**107581 Repeat Search**

107582 *Synopsis:* [count]n

107583 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last  
107584 *pattern*, if the previous search was "/" or "?").

**107585 Repeat Search in Reverse**

107586 *Synopsis:* [count]N

107587 Repeat the search in the opposite direction of the previous search for the *count*th line containing  
107588 the last *pattern* (or not containing the last *pattern*, if the previous search was "/" or "?").

**107589 Examine New File**

107590 *Synopsis:* :e [filename]<newline>

107591 Examine a new file. If the *filename* argument is not specified, the current file (see the :n and :p  
107592 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell  
107593 word expansions (see Section 2.6, on page 2483); if more than a single pathname results, the  
107594 effects are unspecified. If *filename* is a <number-sign> ('#'), the previously examined file shall  
107595 be re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable  
107596 file), an error message to this effect shall be displayed and the current file and screen shall not  
107597 change.

**107598 Examine Next File**

107599 *Synopsis:* [count]:n

107600 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If  
107601 *filename* refers to a non-seekable file, the results are unspecified.

**107602 Examine Previous File**

107603 *Synopsis:* [count]:p

107604 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be  
107605 examined. If *filename* refers to a non-seekable file, the results are unspecified.

**107606 Go to Tag**

107607 *Synopsis:* :t tagstring<newline>

107608 If the file containing the tag named by the *tagstring* argument is not the current file, examine the  
107609 file, as if the :e command was executed with that file as the argument. Otherwise, or in addition,  
107610 display the screenful beginning with the tag, as described for the -t option (see the OPTIONS  
107611 section). If the *ctags* utility is not supported by the system, the use of :t produces undefined  
107612 results.

107613 **Invoke Editor**107614 *Synopsis:*    v

107615 Invoke an editor to edit the current file being examined. If standard input is being examined, the  
 107616 results are unspecified. The name of the editor shall be taken from the environment variable  
 107617 *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the  
 107618 editor shall be invoked with a *-c linenum* command line argument, where *linenum* is the  
 107619 line number of the file line containing the display line currently displayed as the first line of the  
 107620 screen. It is implementation-defined whether line-setting options are passed to editors other  
 107621 than *vi* and *ex*.

107622 When the editor exits, *more* shall resume with the same file and screen as when the editor was  
 107623 invoked.

107624 **Display Position**

107625 *Synopsis:*    =  
 107626               <control>-G

107627 Write a message for which the information references the first byte of the line after the last line of  
 107628 the file on the screen. This message shall include the name of the file currently being examined,  
 107629 its number relative to the total number of files there are to examine, the line number in the file,  
 107630 the byte number and the total bytes in the file, and what percentage of the file precedes the  
 107631 current position. If *more* is reading from standard input, or the file is shorter than a single screen,  
 107632 the line number, the byte number, the total bytes, and the percentage need not be written.

107633 **Quit**

107634 *Synopsis:*    q  
 107635               :q  
 107636               ZZ

107637 Exit *more*.107638 **EXIT STATUS**

107639 The following exit values shall be returned:

107640    0 Successful completion.

107641    &gt;0 An error occurred.

107642 **CONSEQUENCES OF ERRORS**

107643 If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to  
 107644 examine the next file in the argument list, but the final exit status shall be affected. If an error is  
 107645 encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file  
 107646 in the argument list, but the final exit status shall be affected. If an error is encountered accessing  
 107647 a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not  
 107648 be affected.

107649 **APPLICATION USAGE**

107650 When the standard output is not a terminal, only the `-s` filter-modification option is effective.  
 107651 This is based on historical practice. For example, a typical implementation of *man* pipes its  
 107652 output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to  
 107653 *lp*, however, it is undesirable for this squeezing to happen.

107654 **EXAMPLES**

107655 The `-p` allows arbitrary commands to be executed at the start of each file. Examples are:

107656 `more -p G file1 file2`

107657 Examine each file starting with its last screenful.

107658 `more -p 100 file1 file2`

107659 Examine each file starting with line 100 in the current position (usually the third line, so line  
 107660 98 would be the first line written).

107661 `more -p /100 file1 file2`

107662 Examine each file starting with the first line containing the string "100" in the current  
 107663 position

107664 **RATIONALE**

107665 The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the  
 107666 POSIX file display program since it is more widely available than either the public-domain  
 107667 program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the  
 107668 features selected; it is almost fully upwards-compatible from the 4.3 BSD version in wide use  
 107669 and has become more amenable for *vi* users. Several features originally derived from various file  
 107670 editors, found in both *less* and *pg*, have been added to this volume of POSIX.1-2024 as they have  
 107671 proved extremely popular with users.

107672 There are inconsistencies between *more* and *vi* that result from historical practice. For example,  
 107673 the single-character commands **h**, **f**, **b**, and `<space>` are screen movers in *more*, but cursor  
 107674 movers in *vi*. These inconsistencies were maintained because the cursor movements are not  
 107675 applicable to *more* and the powerful functionality achieved without the use of the control key  
 107676 justifies the differences.

107677 The tags interface has been included in a program that is not a text editor because it promotes  
 107678 another degree of consistent operation with *vi*. It is conceivable that the paging environment of  
 107679 *more* would be superior for browsing source code files in some circumstances.

107680 The operating mode referred to for block-mode terminals effectively adds a `<newline>` to each  
 107681 Synopsis line that currently has none. So, for example, `d<newline>` would page one screenful.  
 107682 The mode could be triggered by a command line option, environment variable, or some other  
 107683 method. The details are not imposed by this volume of POSIX.1-2024 because there are so few  
 107684 systems known to support such terminals. Nevertheless, it was considered that all systems  
 107685 should be able to support *more* given the exception cited for this small community of terminals  
 107686 because, in comparison to *vi*, the cursor movements are few and the command set relatively  
 107687 amenable to the optional `<newline>` characters.

107688 Historically some versions of *more* did not obtain the terminal window size on receipt of  
 107689 SIGCONT, resulting in incorrect screen contents when the screen was refreshed if the size had  
 107690 been changed while *more* was suspended. This is considered to be a bug in those  
 107691 implementations.

107692 Some versions of *more* provide a shell escaping mechanism similar to the `ex !` command. The  
 107693 standard developers did not consider that this was necessary in a paginator, particularly given  
 107694 the wide acceptance of multiple window terminals and job control features. (They chose to  
 107695 retain such features in the editors and *mailx* because the shell interaction also gives an

- 107696 opportunity to modify the editing buffer, which is not applicable to *more*.)
- 107697 The `-p` (position) option replaces the `+` command because of the Utility Syntax Guidelines. The  
 107698 `+command` option is no longer specified by POSIX.1-2024 but may be present in some  
 107699 implementations. In early proposals, it took a *pattern* argument, but historical *less* provided the  
 107700 *more* general facility of a command. It would have been desirable to use the same `-c` as *ex* and *vi*,  
 107701 but the letter was already in use.
- 107702 The text stating “from a non-rewindable stream ... implementations may limit the amount of  
 107703 backwards motion supported” would allow an implementation that permitted no backwards  
 107704 motion beyond text already on the screen. It was not possible to require a minimum amount of  
 107705 backwards motion that would be effective for all conceivable device types. The implementation  
 107706 should allow the user to back up as far as possible, within device and reasonable memory  
 107707 allocation constraints.
- 107708 Historically, non-printable characters were displayed using the ARPA standard mappings,  
 107709 which are as follows:
- 107710 1. Printable characters are left alone.
  - 107711 2. Control characters less than `\177` are represented as followed by the character offset from  
 107712 the '@' character in the ASCII map; for example, `\007` is represented as 'G'.
  - 107713 3. `\177` is represented as followed by '?'.
- 107714 The display of characters having their eighth bit set was less standard. Existing implementations  
 107715 use hex (`0x00`), octal (`\000`), and a meta-bit display. (The latter displayed characters with their  
 107716 eighth bit set as the two characters "M-", followed by the seven-bit display as described  
 107717 previously.) The latter probably has the best claim to historical practice because it was used with  
 107718 the `-v` option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.
- 107719 No specific display format is required by POSIX.1-2024. Implementations are encouraged to  
 107720 conform to historic practice in the absence of any strong reason to diverge.
- 107721 **FUTURE DIRECTIONS**
- 107722 None.
- 107723 **SEE ALSO**
- 107724 [Chapter 2](#) (on page 2472), *ctags*, *ed*, *ex*, *vi*
- 107725 [XBD Chapter 8](#) (on page 167), [Section 9.2](#) (on page 180), [Section 9.3](#) (on page 181), [Section 12.2](#)  
 107726 (on page 215)
- 107727 **CHANGE HISTORY**
- 107728 First released in Issue 4.
- 107729 **Issue 5**
- 107730 The FUTURE DIRECTIONS section is added.
- 107731 **Issue 6**
- 107732 This utility is marked as part of the User Portability Utilities option.
- 107733 The obsolescent SYNOPSIS is removed.
- 107734 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:
- 107735 • Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
  - 107736 • The *more* utility should be able to handle underlined and emboldened displays of  
 107737 characters that are wider than a single column position.

107738 **Issue 7**

107739 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized  
107740 as an option delimiter in the OPTIONS section.

107741 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

107742 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0123 [265] is applied.

107743 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0146 [584] is applied.

107744 **Issue 8**

107745 Austin Group Defect 1031 is applied, changing the description of the -i option.

107746 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

107747 Austin Group Defect 1185 is applied, changing the ASYNCHRONOUS EVENTS and  
107748 EXTENDED DESCRIPTION sections in relation to the terminal window size.

107749 **NAME**

107750 msgfmt — create messages objects from portable messages object source files

107751 **SYNOPSIS**107752 msgfmt [-cfSv] [-D *dir*] [-o *outputfile*] *pathname...*107753 **DESCRIPTION**107754 The *msgfmt* utility shall create messages object files from portable messages object source files  
107755 (dot-po files).107756 A dot-po file contains messages to be output by system commands or by applications. The  
107757 messages in these files should be able to be translated to any language supported by the system.107758 The *msgfmt* utility shall interpret message strings for output as characters according to the  
107759 codeset specified in the dot-po file or, if not present, the current setting of the *LC\_CTYPE* locale  
107760 category.107761 **OPTIONS**107762 The *msgfmt* utility shall conform to XBD [Section 12.2](#) (on page 215).

107763 The following options shall be supported:

107764 **-c** If this option and **-v** are both specified, *msgfmt* shall detect and diagnose input file  
107765 abnormalities which might represent translation errors. The **msgid** and **msgstr**  
107766 strings shall be compared. It shall be considered abnormal if one string starts or  
107767 ends with a <newline> while the other does not. Also, if the flag **c-format** appears  
107768 in a "#, " comment for a **msgid** directive (see EXTENDED DESCRIPTION), it shall  
107769 be considered abnormal if the strings do not have the same number of '%'   
107770 conversion specifiers, or if corresponding conversion specifiers take different  
107771 argument types (see XSH *fprintf()*, on page 995). If an abnormality is detected, the  
107772 exit status shall be non-zero and a diagnostic message shall be output. Additional  
107773 checks beyond those described here may also be performed. These checks may  
107774 produce diagnostics or informational messages and need not affect the exit status.  
107775 If **-c** is specified without **-v** or **-v** is specified without **-c**, the behavior is  
107776 unspecified.

107777 **-D *dir*** Add *dir* to the list of directories to search for input files.107778 **-f** Use fuzzy entries in output. If this option is not specified, fuzzy entries shall not be  
107779 included in the output.

107780 **-o *outputfile***  
107781 Specify the name of an output file to be used instead of the default filename(s)  
107782 specified in EXTENDED DESCRIPTION. All **domain *domainname*** directives in the  
107783 dot-po file(s) shall be ignored.

107784 **-S** Append the suffix **.mo** to each generated messages object filename if it does not  
107785 have this suffix.107786 **-v** See **-c**.107787 **OPERANDS**

107788 The following operand shall be supported:

107789 *pathname* A pathname of a dot-po file.

- 107790 **STDIN**  
 107791 Not used.
- 107792 **INPUT FILES**  
 107793 The input files shall be text files in the format described in EXTENDED DESCRIPTION.
- 107794 **ENVIRONMENT VARIABLES**  
 107795 The following environment variables shall affect the execution of *msgfmt*:
- 107796 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 107797 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 107798 variables used to determine the values of locale categories.)
- 107799 XSI *LANGUAGE* Determine the location of messages objects if *NLSPATH* is not set or the evaluation  
 107800 of *NLSPATH* did not lead to a suitable messages object being found.
- 107801 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 107802 internationalization variables.
- 107803 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 107804 characters (for example, single-byte as opposed to multi-byte characters in  
 107805 arguments and input files).
- 107806 *LC\_MESSAGES*  
 107807 Determine the locale name used to locate messages objects, and the locale that  
 107808 should be used to affect the format and contents of diagnostic messages written to  
 107809 standard error.
- 107810 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 107811 **ASYNCHRONOUS EVENTS**  
 107812 Default.
- 107813 **STDOUT**  
 107814 Not Used.
- 107815 **STDERR**  
 107816 The standard error shall be used for diagnostic messages and may also be used for warning  
 107817 messages. If the *-c* and *-v* options are specified, additional unspecified informational messages  
 107818 may be written to standard error.
- 107819 **OUTPUT FILES**  
 107820 The format of the created messages object files is unspecified.
- 107821 **EXTENDED DESCRIPTION**  
 107822 The *msgfmt* utility shall accept portable messages object source files (dot-po files) in the  
 107823 following format.
- 107824 A dot-po file contains zero or more lines, with each non-blank line containing a comment, a  
 107825 statement, or a statement continuation. A comment has an unquoted <number-sign> ('#') as  
 107826 the first non-<blank> character and ends with the next <newline> character. A statement  
 107827 continuation is a double-quoted string on a line by itself, optionally preceded and/or followed  
 107828 by <blank> characters, and the string shall be concatenated with the string on the previous  
 107829 statement line. If a comment occurs between a statement and a statement continuation, the  
 107830 behavior is unspecified. All other comments, except for comments beginning with <number-  
 107831 sign><comma> ("#, "), and blank lines shall be ignored.
- 107832 The format of a statement is:  
 107833 *directive value*



107834 The *directive* starts at the first non-`<blank>` character of the line and is separated from the *value*  
 107835 by one or more `<blank>` characters. The *value* consists of a double-quoted string optionally  
 107836 followed by `<blank>` characters. Zero or more statement continuation lines (see above) can  
 107837 follow the statement. The following directives shall be supported:

```
107838 domain domainname
107839 msgid message_identifier
107840 msgid_plural untranslated_string_plural
107841 msgstr message_string
107842 msgstr[index] message_string
```

107843 A dot-po file consists of zero or more sections. Each section specifies the messages to be  
 107844 processed in a domain. The first directive in each section shall be a **domain** directive (except for  
 107845 the first section which shall behave as if

```
107846 domain "messages"
```

107847 had been specified if the first directive is not a **domain** directive).

107848 The behavior of the **domain** directive is affected by the options used. See OPTIONS for the  
 107849 behavior when the `-o` option is specified. If the `-o` option is not specified, all data obtained from  
 107850 the non-**domain** directives in a dot-po section shall be output to the messages object file named  
 107851 *domainname.mo* when the `-S` option is specified. When the `-S` option is not specified, it is  
 107852 implementation-defined whether *domainname* or *domainname.mo* is used.

107853 If multiple **domain** directives specify the same *domainname*, the sections shall be processed as if  
 107854 there was only one section that starts with a **domain** *domainname* statement which contained the  
 107855 statements of the sections, in the same order, excluding all but the first **domain** *domainname*  
 107856 statement.

107857 Within each section, there can be a header. A header is identified by having a **msgid** directive  
 107858 with the empty string ("") as the *message\_identifier* immediately followed by a statement  
 107859 containing a **msgstr** directive. The *message\_string* in this **msgstr** statement in a header shall be  
 107860 treated specially. If *message\_string* contains a specification of the form:

```
107861 "nplurals=count; plural=expression"
```

107862 then *count* indicates the number of plural forms for messages in that domain, and *expression* is a  
 107863 C-language expression that evaluates to an unsigned integer value which determines the  
 107864 **msgstr**[*index*] directive to be used. The value of *expression* is used as the index value. The  
 107865 variable *n* in *expression* is assigned the value of the *n* argument to the *ngettext()*, *ngettext\_l()*,  
 107866 *dngettext()*, *dngettext\_l()*, *dcngettext()*, and *dcngettext\_l()* functions or of the *n* operand of the  
 107867 *ngettext* utility before *expression* is evaluated. The application shall ensure that *expression*  
 107868 evaluates to a non-negative value less than *count* for all *n* that can be supplied by the  
 107869 aforementioned functions and utility.

107870 If *message\_string* in the header contains a specification of the form:

```
107871 "charset=codeset"
```

107872 then *codeset* indicates the codeset to be used to encode the message strings in this section's  
 107873 domain (overriding *LC\_CTYPE*). If the output string's codeset is different from the message  
 107874 string's codeset, codeset conversion from the message string's codeset to the output string's  
 107875 codeset shall be performed by the *gettext* family of functions and by the *gettext* and *ngettext*  
 107876 utilities. See XSH *gettext* and *gettext*. The output string's codeset shall be determined by the  
 107877 current or specified locale's codeset.

107878 **Note:** It is the responsibility of translators to ensure that the characters they enter into message strings  
107879 in a dot-po file are encoded in the codeset specified in the header.

107880 If a header is present in a section, the application shall ensure that the header is provided by the  
107881 first **msgid** directive in that section.

107882 After the header, if present, zero or more messages are identified by a **msgid** directive with a  
107883 *message\_identifier* that is not an empty string. Each of these directives start a subsection that is  
107884 used to get a translated message from the *gettext* family of functions and from the *gettext* and  
107885 *ngettext* utilities. If the *message\_identifier* string is the string identified by the *gettext* family of  
107886 functions *msgid* argument or by the *gettext* and *ngettext* utility *msgid* operand, this subsection  
107887 specifies how that translation is to be processed.

107888 If there is only a singular form for the given *message\_identifier*, the application shall ensure that  
107889 the statement containing the **msgid** directive is immediately followed by a **msgstr** directive.

107890 If there are plural forms for the given *message\_identifier* and the header for this section exists and  
107891 contains an

107892 `"nplurals=count; plural=expression"`

107893 specification, the application shall ensure that the statement containing the **msgid** directive is  
107894 immediately followed by a **msgid\_plural** directive and that each statement containing a  
107895 **msgid\_plural** directive is followed by *count* statements containing **msgstr[index]** directives,  
107896 starting with **msgstr[0]** and ending with **msgstr[count-1]** in increasing order, with no duplicate  
107897 index values. If a header for this section does not exist or does not contain an

107898 `"nplurals=count; plural=expression"`

107899 specification, the application shall ensure that no **msgid\_plural** or **msgstr[index]** directives are  
107900 used in this section.

107901 For example, if the header's *message\_string* contains the specification:

107902 `"nplurals=2; plural= n == 1 ? 0 : 1"`

107903 there are two forms in the domain; **msgstr[0]** is used if *n* is equal to 1, otherwise **msgstr[1]** is  
107904 used. For another example, if the header's *message\_string* contains:

107905 `"nplurals=3; plural= n == 1 ? 0 : n == 2 ? 1 : 2"`

107906 there are three forms in the domain; **msgstr[0]** is used if *n* is equal to 1, **msgstr[1]** is used if *n* is  
107907 equal to 2, otherwise **msgstr[2]** is used.

107908 C-language escape sequences in strings shall be processed as specified for character string  
107909 literals in the ISO C standard, except that *universal-character-name* escape sequences need not be  
107910 supported.

107911 Comments in a dot-po file can be in one of the following formats:

107912 `#: reference`

107913 `#. utility-added-comments`

107914 `#, flag`

107915 `#translator-comments` (where *translator-comments* does not begin with '.', ': ' or ',')

107916 A `#: reference` comment indicates the location(s) of the **msgid** string in the source files, in

107917 `pathname1:linenumber1 [pathname2:linenumber2 ... ]`

107918 format. They can be added, as might "#." prefixed additional comments of unspecified format,  
107919 by the *xgettext* utility. All comments that do not begin with "#," are informative only and shall  
107920 be silently ignored by the *msgfmt* utility. In "#," comments the following values for *flag* can be

107921 specified:

107922 **fuzzy** This flag indicates that the **msgstr** string might not be a correct translation at this  
 107923 point in time. Only the translator can judge if the translation requires further  
 107924 modification or is acceptable as is. Once satisfied with the translation, the  
 107925 translator should remove this **fuzzy** flag. If this flag is specified, the *msgfmt* utility  
 107926 shall not generate the entry for the next following **msgid** in the output message  
 107927 catalog, unless the **-f** option is specified. If other flag comments are specified  
 107928 between **fuzzy** and the **msgid**, the behavior is unspecified.

107929 **c-format**

107930 **no-c-format** The **c-format** flag indicates that the next following **msgid** string contains a *printf()*  
 107931 format string. When the **c-format** flag is given and the **-c** and **-v** options are  
 107932 specified, the *msgfmt* utility shall perform additional tests to check the validity of  
 107933 the translation (see OPTIONS); these additional tests may also be performed if  
 107934 neither **c-format** nor **no-c-format** is given. When the **no-c-format** flag is given for a  
 107935 string, no additional checks shall be performed for the string. When both the **c-**  
 107936 **format** and the **no-c-format** flags are given, the last flag specified takes precedence.

### 107937 EXIT STATUS

107938 The following exit values shall be returned:

107939 0 Successful completion.

107940 >0 An error occurred.

### 107941 CONSEQUENCES OF ERRORS

107942 The *msgfmt* utility need not continue processing later *pathname* operands when an error  
 107943 condition that affects the exit status is detected. It is unspecified whether a messages object file is  
 107944 written when checks performed for the **-c** and **-v** options fail.

### 107945 APPLICATION USAGE

107946 The *xgettext* utility can be used to create template dot-po files from C-language source files.

107947 Installing messages object files for the POSIX or C locale is not recommended, since they may be  
 107948 ignored for the sake of efficiency.

107949 The first section for each domain in a dot-po file should include a header containing a

107950 "charset=codeset"

107951 specification. If this specification is omitted, message conversions in the *gettext* family of  
 107952 functions and in the *gettext* and *ngettext* utilities may fail.

107953 The **msgid\_plural** directive's *untranslated\_string\_plural* string comes from the *msgid\_plural*  
 107954 arguments in calls to the *ngettext()*, *ngettext\_l()*, *dngettext()*, *dngettext\_l()*, *dcngettext()*, and  
 107955 *dcngettext\_l()* functions when a prototype dot-po file is created by the *xgettext* utility. These  
 107956 strings (and the *msgid\_plural* operands in calls to the *ngettext* utility) can provide context when a  
 107957 translator is modifying a template dot-po file into a dot-po file for a specific language. These  
 107958 functions and the *ngettext* utility do not try to match the *msgid\_plural* arguments or operands  
 107959 with anything in a messages object file; they only match the *msgid* arguments and operands.

107960 Unlike shell command language strings, double-quoted strings in dot-po files cannot contain a  
 107961 literal <newline> character.

## 107962 EXAMPLES

```
107963     In this example, module1.po and module2.po are portable messages object source files.
107964     $ cat module1.po
107965     # default domain "messages"
107966     msgid ""
107967     msgstr "charset=utf-8"
107968     msgid "msg 1"
107969     msgstr "msg 1 translation"
107970     #
107971     domain "help_domain"
107972     msgid ""
107973     msgstr "charset=utf-8"
107974     msgid "help 2"
107975     msgstr "help 2 translation"
107976     #
107977     domain "error_domain"
107978     msgid ""
107979     msgstr "charset=utf-8"
107980     msgid "error 3"
107981     msgstr "error 3 translation"
107982
107983     $ cat module2.po
107984     # default domain "messages"
107985     msgid ""
107986     msgstr "charset=utf-8"
107987     msgid "mesg 4"
107988     msgstr "mesg 4 translation"
107989     #
107990     domain "error_domain"
107991     msgid ""
107992     msgstr "charset=utf-8"
107993     #, c-format
107994     msgid "error 5 %s"
107995     msgstr "error 5 translation %s"
107996     #
107997     domain "window_domain"
107998     msgid ""
107999     msgstr "charset=utf-8"
108000     msgid "window 6"
108001     msgstr "window 6 translation"
108002
108003     $ cat module3.po
108004     # default domain "messages"
108005     # header will be used for the whole output file in the third example
108006     msgid ""
108007     msgstr "charset=utf-8"
108008     msgid "info 0"
108009     msgstr "info 0 translation"
108010
108011     $ cat opt_debug.po
108012     #
108013     domain "debug_domain"
108014     msgid "debug 8"
```

108012 **msgstr "debug 8 translation"**

108013 The following command will produce the output files **messages.mo**, **help\_domain.mo**, and  
108014 **error\_domain.mo**:

108015 **\$ msgfmt -S module1.po**

108016 The following command will produce the output files **messages.mo**, **help\_domain.mo**,  
108017 **error\_domain.mo**, and **window\_domain.mo**:

108018 **\$ msgfmt -S module1.po module2.po**

108019 The following command will produce the output file **hello.mo**:

108020 **\$ msgfmt -o hello.mo module3.po opt\_debug.po**

#### 108021 **RATIONALE**

108022 Some implementations are less strict about the format of dot-po files and simply treat all  
108023 occurrences of one or more white space characters as a separator. The format described in this  
108024 standard is accepted by all known implementations.

108025 In some implementations, duplicate **msgid** directives within a domain are ignored, and only an  
108026 entry for the first **msgid** directive and the following **msgid**, **msgid\_plural**, **msgstr**, or  
108027 **msgstr[index]** directives is created. However, some implementations consider duplicate **msgid**  
108028 directives within a domain to be an error and do not produce output at all. Consequently this  
108029 standard does not specify the behavior of *msgfmt* if duplicate **msgid** directives are encountered  
108030 within one domain.

#### 108031 **FUTURE DIRECTIONS**

108032 If this utility is directed to create a new directory entry that contains any bytes that have the  
108033 encoded value of a <newline> character, implementations are encouraged to treat this as an  
108034 error. A future version of this standard may require implementations to treat this as an error.

#### 108035 **SEE ALSO**

108036 *gettext*, *xgettext*

108037 XSH *fprintf()*, *gettext*

#### 108038 **CHANGE HISTORY**

108039 First released in Issue 8.

108040 **NAME**

108041 mv — move files

108042 **SYNOPSIS**108043 mv [-if] *source\_file target\_file*108044 mv [-if] *source\_file... target\_dir*108045 **DESCRIPTION**

108046 In the first synopsis form, the *mv* utility shall move the file named by the *source\_file* operand to  
 108047 the destination specified by the *target\_file*. This first synopsis form is assumed when the final  
 108048 operand does not name an existing directory and is not a symbolic link referring to an existing  
 108049 directory. In this case, if *source\_file* names a non-directory file and *target\_file* ends with a trailing  
 108050 <slash> character, *mv* shall treat this as an error and no *source\_file* operands shall be processed.

108051 In the second synopsis form, *mv* shall move each file named by a *source\_file* operand to a  
 108052 destination file in the existing directory named by the *target\_dir* operand, or referenced if  
 108053 *target\_dir* is a symbolic link referring to an existing directory. The destination path for each  
 108054 *source\_file* shall be the concatenation of the target directory, a single <slash> character if the  
 108055 target did not end in a <slash>, and the last pathname component of the *source\_file*. This second  
 108056 form is assumed when the final operand names an existing directory.

108057 If any operand specifies an existing file of a type not specified by the System Interfaces volume  
 108058 of POSIX.1-2024, the behavior is implementation-defined.

108059 For each *source\_file* the following steps shall be taken:

108060 1. If the destination path exists, the *-f* option is not specified, and either of the following  
 108061 conditions is true:

108062 a. The permissions of the destination path do not permit writing and the standard  
 108063 input is a terminal.

108064 b. The *-i* option is specified.

108065 the *mv* utility shall write a prompt to standard error and read a line from standard input.  
 108066 If the response is not affirmative, *mv* shall do nothing more with the current *source\_file*  
 108067 and go on to any remaining *source\_files*.

108068 2. If the *source\_file* operand and destination path resolve to either the same existing directory  
 108069 entry or different directory entries for the same existing file, then the destination path  
 108070 shall not be removed, and one of the following shall occur:

108071 a. No change is made to *source\_file*, no error occurs, and no diagnostic is issued.

108072 b. No change is made to *source\_file*, a diagnostic is issued to standard error  
 108073 identifying the two names, and the exit status is affected.

108074 c. If the *source\_file* operand and destination path name distinct directory entries, then  
 108075 the *source\_file* operand is removed, no error occurs, and no diagnostic is issued.

108076 The *mv* utility shall do nothing more with the current *source\_file*, and go on to any  
 108077 remaining *source\_files*.

108078 3. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the  
 108079 System Interfaces volume of POSIX.1-2024, called with the following arguments:

108080 a. The *source\_file* operand is used as the *old* argument.

- 108081           b. The destination path is used as the *new* argument.
- 108082           If this succeeds, *mv* shall do nothing more with the current *source\_file* and go on to any  
108083           remaining *source\_files*. If this fails for any reasons other than those described for the *errno*  
108084           [EXDEV] in the System Interfaces volume of POSIX.1-2024, *mv* shall write a diagnostic  
108085           message to standard error, do nothing more with the current *source\_file*, and go on to any  
108086           remaining *source\_files*.
- 108087           4. If the destination path exists, and it is a file of type directory and *source\_file* is not a file of  
108088           type directory, or it is a file not of type directory and *source\_file* is a file of type directory,  
108089           *mv* shall write a diagnostic message to standard error, do nothing more with the current  
108090           *source\_file*, and go on to any remaining *source\_files*. If the destination path exists and was  
108091           created by a previous step, it is unspecified whether this will be treated as an error or the  
108092           destination path will be overwritten.
- 108093           5. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*  
108094           shall write a diagnostic message to standard error, do nothing more with the current  
108095           *source\_file*, and go on to any remaining *source\_files*.
- 108096           6. The file hierarchy rooted in *source\_file* shall be duplicated as a file hierarchy rooted in the  
108097           destination path. If *source\_file* or any of the files below it in the hierarchy are symbolic  
108098           links, the links themselves shall be duplicated, including their contents, rather than any  
108099           files to which they refer. The following characteristics of each file in the file hierarchy  
108100           shall be duplicated:
- 108101           • The time of last data modification and time of last access
  - 108102           • The user ID and group ID
  - 108103           • The file mode
- 108104           If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode  
108105           bits S\_ISUID and S\_ISGID shall not be duplicated.
- 108106           When files are duplicated to another file system, the implementation may require that the  
108107           process invoking *mv* has read access to each file being duplicated.
- 108108           If files being duplicated to another file system have hard links to other files, it is  
108109           unspecified whether the files copied to the new file system have the hard links preserved  
108110           or separate copies are created for the linked files.
- 108111           If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic  
108112           message to standard error, do nothing more with the current *source\_file*, and go on to any  
108113           remaining *source\_files*.
- 108114           If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic  
108115           message to standard error, but this failure shall not cause *mv* to modify its exit status.
- 108116           7. The file hierarchy rooted in *source\_file* shall be removed. If this fails for any reason, *mv*  
108117           shall write a diagnostic message to the standard error, do nothing more with the current  
108118           *source\_file*, and go on to any remaining *source\_files*.

## 108119 OPTIONS

- 108120           The *mv* utility shall conform to XBD [Section 12.2](#) (on page 215).
- 108121           The following options shall be supported:
- 108122           **-f**           Do not prompt for confirmation if the destination path exists. Any previous  
108123           occurrence of the **-i** option is ignored.

108124 **-i** Prompt for confirmation if the destination path exists. Any previous occurrence of  
 108125 the **-f** option is ignored.

108126 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option  
 108127 specified shall determine the behavior of *mv*.

#### 108128 OPERANDS

108129 The following operands shall be supported:

108130 *source\_file* A pathname of a file or directory to be moved.

108131 *target\_file* A new pathname for the file or directory being moved.

108132 *target\_dir* A pathname of an existing directory into which to move the input files.

#### 108133 STDIN

108134 The standard input shall be used to read an input line in response to each prompt specified in  
 108135 the STDERR section. Otherwise, the standard input shall not be used.

#### 108136 INPUT FILES

108137 The input files specified by each *source\_file* operand can be of any file type.

#### 108138 ENVIRONMENT VARIABLES

108139 The following environment variables shall affect the execution of *mv*:

108140 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 108141 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 108142 variables used to determine the values of locale categories.)

108143 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 108144 internationalization variables.

108145 *LC\_COLLATE*

108146 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 108147 character collating elements used in the extended regular expression defined for  
 108148 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

108149 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 108150 characters (for example, single-byte as opposed to multi-byte characters in  
 108151 arguments and input files), the behavior of character classes used in the extended  
 108152 regular expression defined for the **yesexpr** locale keyword in the *LC\_MESSAGES*  
 108153 category.

108154 *LC\_MESSAGES*

108155 Determine the locale used to process affirmative responses, and the locale used to  
 108156 affect the format and contents of diagnostic messages and prompts written to  
 108157 standard error.

108158 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

#### 108159 ASYNCHRONOUS EVENTS

108160 Default.

#### 108161 STDOUT

108162 Not used.

#### 108163 STDERR

108164 Prompts shall be written to the standard error under the conditions specified in the  
 108165 DESCRIPTION section. The prompts shall contain the destination pathname, but their format is  
 108166 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic



108167 messages.

#### 108168 OUTPUT FILES

108169 The output files may be of any file type.

#### 108170 EXTENDED DESCRIPTION

108171 None.

#### 108172 EXIT STATUS

108173 The following exit values shall be returned:

108174 0 All requested files (excluding files where a non-affirmative response was given to a request  
108175 for confirmation) were successfully moved.

108176 >0 An error occurred.

#### 108177 CONSEQUENCES OF ERRORS

108178 If the copying or removal of *source\_file* is prematurely terminated by a signal or error, *mv* may  
108179 leave a partial copy of *source\_file* at the source or destination. The *mv* utility shall not modify  
108180 both *source\_file* and the destination path simultaneously; termination at any point shall leave  
108181 either *source\_file* or the destination path complete.

#### 108182 APPLICATION USAGE

108183 Some implementations mark for update the last file status change timestamp of renamed files  
108184 and some do not. Applications which make use of the last file status change timestamp may  
108185 behave differently with respect to renamed files unless they are designed to allow for either  
108186 behavior.

108187 The specification ensures that *mv a a* will not alter the contents of file *a*, and allows the  
108188 implementation to issue an error that a file cannot be moved onto itself. Likewise, when *a* and *b*  
108189 are hard links to the same file, *mv a b* will not alter *b*, but if a diagnostic is not issued, then it is  
108190 unspecified whether *a* is left untouched (as it would be by the *rename()* function) or unlinked  
108191 (reducing the link count of *b*).

#### 108192 EXAMPLES

108193 If the current directory contains only files *a* (of any type defined by the System Interfaces  
108194 volume of POSIX.1-2024), *b* (also of any type), and a directory *c*:

108195 `mv a b c`

108196 `mv c d`

108197 results with the original files *a* and *b* residing in the directory *d* in the current directory.

#### 108198 RATIONALE

108199 Early proposals diverged from the SVID and BSD historical practice in that they required that  
108200 when the destination path exists, the `-f` option is not specified, and input is not a terminal, *mv*  
108201 fails. This was done for compatibility with *cp*. The current text returns to historical practice. It  
108202 should be noted that this is consistent with the *rename()* function defined in the System  
108203 Interfaces volume of POSIX.1-2024, which does not require write permission on the target.

108204 For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for  
108205 confirmation, should be interpreted in the following manner:

```
108206 if (exists AND (NOT f_option) AND
108207     ((not_writable AND input_is_terminal) OR i_option))
```

108208 The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally  
108209 unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv*  
108210 deletes all existing destination paths without prompting, even when `-i` is specified; this is

108211 inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when  
 108212 the file is unwritable and the standard input is not a terminal. The standard developers decided  
 108213 that use of `-i` is a request for interaction, so when the destination path exists, the utility takes  
 108214 instructions from whatever responds to standard input.

108215 The *rename()* function is able to move directories within the same file system. Some historical  
 108216 versions of *mv* have been able to move directories, but not to a different file system. The  
 108217 standard developers considered that this was an annoying inconsistency, so this volume of  
 108218 POSIX.1-2024 requires directories to be able to be moved even across file systems. There is no `-R`  
 108219 option to confirm that moving a directory is actually intended, since such an option was not  
 108220 required for moving directories in historical practice. Requiring the application to specify it  
 108221 sometimes, depending on the destination, seemed just as inconsistent. The semantics of the  
 108222 *rename()* function were preserved as much as possible. For example, *mv* is not permitted to  
 108223 “rename” files to or from directories, even though they might be empty and removable.

108224 Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to  
 108225 duplicate any file characteristics when moving a file across file systems, nor did they write a  
 108226 diagnostic message for the user. The former behavior has been preserved to prevent scripts from  
 108227 breaking; a diagnostic message is now required, however, so that users are alerted that the file  
 108228 characteristics have changed.

108229 The exact format of the interactive prompts is unspecified. Only the general nature of the  
 108230 contents of prompts are specified because implementations may desire more descriptive  
 108231 prompts than those used on historical implementations. Therefore, an application not using the  
 108232 `-f` option or using the `-i` option relies on the system to provide the most suitable dialog directly  
 108233 with the user, based on the behavior specified.

108234 When *mv* is dealing with a single file system and *source\_file* is a symbolic link, the link itself is  
 108235 moved as a consequence of the dependence on the *rename()* functionality, per the  
 108236 DESCRIPTION. Across file systems, this has to be made explicit.

#### 108237 **FUTURE DIRECTIONS**

108238 If this utility is directed to create a new directory entry that contains any bytes that have the  
 108239 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 108240 error. A future version of this standard may require implementations to treat this as an error.

#### 108241 **SEE ALSO**

108242 *cp*, *ln*  
 108243 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)  
 108244 XSH *rename()*

#### 108245 **CHANGE HISTORY**

108246 First released in Issue 2.

#### 108247 **Issue 6**

108248 The *mv* utility is changed to describe processing of symbolic links as specified in the  
 108249 IEEE P1003.2b draft standard.

108250 The APPLICATION USAGE section is added.

#### 108251 **Issue 7**

108252 Austin Group Interpretation 1003.1-2001 #016 is applied.

108253 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
 108254 *LC\_MESSAGES* environment variable.

108255 Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

- 108256 SD5-XCU-ERN-13 is applied, making an editorial correction to the SYNOPSIS.
- 108257 SD5-XCU-ERN-51 is applied to the DESCRIPTION, defining the behavior for when files are  
108258 being duplicated to another file system while having hard links.
- 108259 Changes are made related to support for finegrained timestamps.
- 108260 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0124 [48] is applied.
- 108261 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0147 [534] is applied.
- 108262 **Issue 8**
- 108263 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
108264 filenames containing any bytes that have the encoded value of a <newline> character.
- 108265 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 108266 Austin Group Defect 1732 is applied, changing the EXIT STATUS section.

108267 **NAME**

108268 newgrp — change to a new group

108269 **SYNOPSIS**108270 newgrp [-l] [*group*]108271 **DESCRIPTION**

108272 The *newgrp* utility shall create a new shell execution environment with a new real and effective  
 108273 group identification. Of the attributes listed in [Section 2.13](#) (on page 2522), the new shell  
 108274 execution environment shall retain the working directory, file creation mask, and exported  
 108275 variables from the previous environment (that is, open files, traps, unexported variables, alias  
 108276 definitions, shell functions, and *set* options may be lost). All other aspects of the process  
 108277 environment that are preserved by the *exec* family of functions defined in the System Interfaces  
 108278 volume of POSIX.1-2024 shall also be preserved by *newgrp*; whether other aspects are preserved  
 108279 is unspecified.

108280 A failure to assign the new group identifications (for example, for security or password-related  
 108281 reasons) shall not prevent the new shell execution environment from being created.

108282 The *newgrp* utility shall affect the supplemental groups for the process as follows:

- 108283 • On systems where the effective group ID is normally in the supplementary group list (or  
 108284 whenever the old effective group ID actually is in the supplementary group list):
  - 108285 — If the new effective group ID is also in the supplementary group list, *newgrp* shall  
 108286 change the effective group ID.
  - 108287 — If the new effective group ID is not in the supplementary group list, *newgrp* shall add  
 108288 the new effective group ID to the list, if there is room to add it.
- 108289 • On systems where the effective group ID is not normally in the supplementary group list  
 108290 (or whenever the old effective group ID is not in the supplementary group list):
  - 108291 — If the new effective group ID is in the supplementary group list, *newgrp* shall delete  
 108292 it.
  - 108293 — If the old effective group ID is not in the supplementary list, *newgrp* shall add it if  
 108294 there is room.

108295 **Note:** The System Interfaces volume of POSIX.1-2024 does not specify whether the effective group ID  
 108296 of a process is included in its supplementary group list.

108297 With no operands, *newgrp* shall change the effective group back to the groups identified in the  
 108298 user's user entry, and shall set the list of supplementary groups to that set in the user's group  
 108299 database entries.

108300 If the first argument is '-', the results are unspecified.

108301 If a password is required for the specified group, and the user is not listed as a member of that  
 108302 group in the group database, the user shall be prompted to enter the correct password for that  
 108303 group. If the user is listed as a member of that group, no password shall be requested. If no  
 108304 password is required for the specified group, it is implementation-defined whether users not  
 108305 listed as members of that group can change to that group. Whether or not a password is  
 108306 required, implementation-defined system accounting or security mechanisms may impose  
 108307 additional authorization restrictions that may cause *newgrp* to write a diagnostic message and  
 108308 suppress the changing of the group identification.

108309 **OPTIONS**

108310 The *newgrp* utility shall conform to XBD [Section 12.2](#) (on page 215), except for the unspecified  
108311 usage of `'-'`.

108312 The following option shall be supported:

108313 `-l` (The letter ell.) Change the environment to what would be expected if the user  
108314 actually logged in again.

108315 **OPERANDS**

108316 The following operand shall be supported:

108317 *group* A group name from the group database or a non-negative numeric group ID.  
108318 Specifies the group ID to which the real and effective group IDs shall be set. If  
108319 *group* is a non-negative numeric string and exists in the group database as a group  
108320 name (see *getgrnam()*), the numeric group ID associated with that group name  
108321 shall be used as the group ID.

108322 **STDIN**

108323 Not used.

108324 **INPUT FILES**

108325 The file `/dev/tty` shall be used to read a single line of text for password checking, when one is  
108326 required.

108327 **ENVIRONMENT VARIABLES**

108328 The following environment variables shall affect the execution of *newgrp*:

108329 *LANG* Provide a default value for the internationalization variables that are unset or null.  
108330 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
108331 variables used to determine the values of locale categories.)

108332 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
108333 internationalization variables.

108334 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
108335 characters (for example, single-byte as opposed to multi-byte characters in  
108336 arguments).

108337 *LC\_MESSAGES*

108338 Determine the locale that should be used to affect the format and contents of  
108339 diagnostic messages written to standard error.

108340 *XSI* *NLSPATH* Determine the location of messages objects and message catalogs.

108341 **ASYNCHRONOUS EVENTS**

108342 Default.

108343 **STDOUT**

108344 Not used.

108345 **STDERR**

108346 The standard error shall be used for diagnostic messages and a prompt string for a password, if  
108347 one is required. Diagnostic messages may be written in cases where the exit status is not  
108348 available. See the EXIT STATUS section.

108349 **OUTPUT FILES**

108350 None.

108351 **EXTENDED DESCRIPTION**

108352 None.

108353 **EXIT STATUS**

108354 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group identification was changed successfully, the exit status shall be the exit status of the shell.

108355 Otherwise, the following exit value shall be returned:

108356 &gt;0 An error occurred.

108358 **CONSEQUENCES OF ERRORS**

108359 The invoking shell may terminate.

108360 **APPLICATION USAGE**

108361 There is no convenient way to enter a password into the group database. Use of group passwords is not encouraged, because by their very nature they encourage poor security practices. Group passwords may disappear in the future.

108362 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with *newgrp*, which in turn overlays itself with a new shell after changing group. On some implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

108363 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a useful interface for the support of applications.

108364 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it successfully invokes a new shell and the rest of the original shell script is bypassed when the new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems. But usage such as:

108365 

```
newgrp foo
```

108366 

```
echo $?
```

108367 is not useful because the new shell might not have access to any status *newgrp* may have generated (and most historical systems do not provide this status). A zero status echoed here does not necessarily indicate that the user has changed to the new group successfully. Following *newgrp* with the *id* command provides a portable means of determining whether the group change was successful or not.

108380 **EXAMPLES**

108381 None.

108382 **RATIONALE**

108383 Most historical implementations use one of the *exec* functions to implement the behavior of *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp* issue a diagnostic message to tell the user that the environment changed, it would be inappropriate to require this change to some historical implementations.

108384 The password mechanism is allowed in the group database, but how this would be implemented is not specified.

108385 The *newgrp* utility was retained in this volume of POSIX.1-2024, even given the existence of the multiple group permissions feature in the System Interfaces volume of POSIX.1-2024, for several reasons. First, in some implementations, the group ownership of a newly created file is determined by the group of the directory in which the file is created, as allowed by the System

108394 Interfaces volume of POSIX.1-2024; on other implementations, the group ownership of a newly  
108395 created file is determined by the effective group ID. On implementations of the latter type,  
108396 *newgrp* allows files to be created with a specific group ownership. Finally, many  
108397 implementations use the real group ID in accounting, and on such systems, *newgrp* allows the  
108398 accounting identity of the user to be changed.

108399 **FUTURE DIRECTIONS**

108400 None.

108401 **SEE ALSO**

108402 [Chapter 2](#) (on page 2472), *sh*

108403 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

108404 XSH *exec*, *getgrnam()*

108405 **CHANGE HISTORY**

108406 First released in Issue 2.

108407 **Issue 6**

108408 This utility is marked as part of the User Portability Utilities option.

108409 The obsolescent SYNOPSIS is removed.

108410 The text describing supplemental groups is no longer conditional on {NGROUPS\_MAX} being  
108411 greater than 1. This is because {NGROUPS\_MAX} now has a minimum value of 8. This is a FIPS  
108412 requirement.

108413 **Issue 7**

108414 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first  
108415 argument is '- '.

108416 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

108417 The *newgrp* utility is moved from the User Portability Utilities option to the Base. User  
108418 Portability Utilities is now an option for interactive utilities.

108419 **Issue 8**

108420 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

108421 **NAME**

108422           ngettext — retrieve text string from messages object

108423 **SYNOPSIS**108424           ngettext [-e|-E] [-d *textdomain*] [*textdomain*] *msgid msgid\_plural n*108425 **DESCRIPTION**108426           Refer to *gettext*.



108427 **NAME**

108428 nice — invoke a utility with an altered nice value

108429 **SYNOPSIS**

108430 nice [-n *increment*] *utility* [*argument...*]

108431 **DESCRIPTION**

108432 The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see  
 108433 XBD [Section 3.225](#), on page 64). With no options, the executed utility shall be run with a nice  
 108434 value that is some implementation-defined quantity greater than or equal to the nice value of the  
 108435 current process. If the user lacks appropriate privileges to affect the nice value in the requested  
 108436 manner, the *nice* utility shall not affect the nice value; in this case, a warning message may be  
 108437 written to standard error, but this shall not prevent the invocation of *utility* or affect the exit  
 108438 status.

108439 **OPTIONS**

108440 The *nice* utility shall conform to XBD [Section 12.2](#) (on page 215).

108441 The following option is supported:

108442 **-n *increment*** A positive or negative decimal integer which shall have the same effect on the  
 108443 execution of the utility as if the utility had called the *nice()* function with the  
 108444 numeric value of the *increment* option-argument.

108445 **OPERANDS**

108446 The following operands shall be supported:

108447 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the  
 108448 special built-in utilities in [Section 2.15](#) (on page 2526), the results are undefined.

108449 *argument* Any string to be supplied as an argument when invoking the utility named by the  
 108450 *utility* operand.

108451 **STDIN**

108452 Not used.

108453 **INPUT FILES**

108454 None.

108455 **ENVIRONMENT VARIABLES**

108456 The following environment variables shall affect the execution of *nice*:

108457 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 108458 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 108459 variables used to determine the values of locale categories.)

108460 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 108461 internationalization variables.

108462 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 108463 characters (for example, single-byte as opposed to multi-byte characters in  
 108464 arguments).

108465 **LC\_MESSAGES**

108466 Determine the locale that should be used to affect the format and contents of  
 108467 diagnostic messages written to standard error.

108468 **XSIX** **NLSPATH** Determine the location of messages objects and message catalogs.

108469 *PATH* Determine the search path used to locate the utility to be invoked. See XBD  
108470 [Chapter 8](#) (on page 167).

108471 **ASYNCHRONOUS EVENTS**

108472 Default.

108473 **STDOUT**

108474 Not used.

108475 **STDERR**

108476 The standard error shall be used only for diagnostic messages.

108477 **OUTPUT FILES**

108478 None.

108479 **EXTENDED DESCRIPTION**

108480 None.

108481 **EXIT STATUS**

108482 If *utility* is invoked, the exit status of *nice* shall be the exit status of *utility*; otherwise, the *nice*  
108483 utility shall exit with one of the following values:

108484 1-125 An error occurred in the *nice* utility.

108485 126 The utility specified by *utility* was found but could not be invoked.

108486 127 The utility specified by *utility* could not be found.

108487 **CONSEQUENCES OF ERRORS**

108488 Default.

108489 **APPLICATION USAGE**

108490 The only guaranteed portable uses of this utility are:

108491 *nice utility*

108492 Run *utility* with the default higher or equal nice value.

108493 *nice -n <positive integer> utility*

108494 Run *utility* with a higher nice value.

108495 On some implementations they have no discernible effect on the invoked utility and on some  
108496 others they are exactly equivalent.

108497 Historical systems have frequently supported the *<positive integer>* up to 20. Since there is no  
108498 error penalty associated with guessing a number that is too high, users without access to the  
108499 system conformance document (to see what limits are actually in place) could use the historical 1  
108500 to 20 range or attempt to use very large numbers if the job should be truly low priority.

108501 The nice value of a process can be displayed using the command:

108502 `ps -o nice`

108503 The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit  
108504 code 127 if a utility to be invoked cannot be found, so that applications can distinguish “failure  
108505 to find a utility” from “invoked utility exited with an error indication”. The value 127 was  
108506 chosen because it is not commonly used for other meanings; most utilities use small values for  
108507 “normal error conditions” and the values above 128 can be confused with termination due to  
108508 receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could  
108509 be found, but not invoked. Some scripts produce meaningful error messages differentiating the  
108510 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice  
108511 that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any

108512 attempt to *exec* the utility fails for any other reason.

108513 **EXAMPLES**

108514 None.

108515 **RATIONALE**

108516 The 4.3 BSD version of *nice* does not check whether *increment* is a valid decimal integer. The  
108517 command *nice -x utility*, for example, would be treated the same as the command *nice --1*  
108518 *utility*. If the user does not have appropriate privileges, this results in a “permission denied”  
108519 error. This is considered a bug.

108520 When a user without appropriate privileges gives a negative *increment*, System V treats it like  
108521 the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not  
108522 run the utility. The standard specifies the System V behavior together with an optional BSD-style  
108523 “permission denied” message.

108524 The C shell has a built-in version of *nice* that has a different interface from the one described in  
108525 this volume of POSIX.1-2024.

108526 The term “utility” is used, rather than “command”, to highlight the fact that shell compound  
108527 commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used.  
108528 However, “utility” includes user application programs and shell scripts, not just utilities defined  
108529 in this volume of POSIX.1-2024.

108530 Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the  
108531 default nice value being the midpoint of that range. By default, they raise the nice value of the  
108532 executed utility by 10.

108533 Some historical documentation states that the *increment* value must be within a fixed range. This  
108534 is misleading; the valid *increment* values on any invocation are determined by the current  
108535 process nice value, which is not always the default.

108536 The definition of nice value is not intended to suggest that all processes in a system have  
108537 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the  
108538 System Interfaces volume of POSIX.1-2024 make the notion of a single underlying priority for all  
108539 scheduling policies problematic. Some implementations may implement the *nice*-related features  
108540 to affect all processes on the system, others to affect just the general time-sharing activities  
108541 implied by this volume of POSIX.1-2024, and others may have no effect at all. Because of the use  
108542 of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are  
108543 possible.

108544 Earlier versions of this standard allowed a *-increment* option. This form is no longer specified by  
108545 POSIX.1-2024 but may be present in some implementations.

108546 **FUTURE DIRECTIONS**

108547 None.

108548 **SEE ALSO**

108549 [Chapter 2](#) (on page 2472), [renice](#)

108550 [XBD Section 3.225](#) (on page 64), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

108551 XSH [nice\(\)](#)

108552 **CHANGE HISTORY**

108553 First released in Issue 4.

108554 **Issue 6**

108555 This utility is marked as part of the User Portability Utilities option.

108556 The obsolescent SYNOPSIS is removed.

108557 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/18 is applied, deleting a paragraph of  
108558 RATIONALE that referred to text no longer in the standard.

108559 **Issue 7**

108560 Austin Group Interpretation 1003.1-2001 #027 is applied.

108561 SD5-XCU-ERN-32 and SD5-XCU-ERN-33 are applied, updating the DESCRIPTION,  
108562 APPLICATION USAGE, and RATIONALE sections.

108563 The *nice* utility is moved from the User Portability Utilities option to the Base. User Portability  
108564 Utilities is now an option for interactive utilities.

108565 **Issue 8**

108566 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

108567 Austin Group Defect 1586 is applied, adding the *timeout* utility.

108568 Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

108569 **NAME**

108570 nl — line numbering filter

108571 **SYNOPSIS**

```
108572 XSI nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num]
108573 [-n format] [-s sep] [-v startnum] [-w width] [file]
```

108574 **DESCRIPTION**

108575 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and  
 108576 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional  
 108577 functionality may be provided in accordance with the command options in effect.

108578 The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at  
 108579 the start of each logical page. A logical page consists of a header, a body, and a footer section.  
 108580 Empty sections are valid. Different line numbering options are independently available for  
 108581 header, body, and footer (for example, no numbering of header and footer lines while  
 108582 numbering blank lines only in the body).

108583 The starts of logical page sections shall be signaled by input lines containing nothing but the  
 108584 following delimiter characters:

| Line     | Start of |
|----------|----------|
| \: \: \: | Header   |
| \: \:    | Body     |
| \:       | Footer   |

108585  
 108586  
 108587  
 108588  
 108589 Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

108590 **OPTIONS**

108591 The *nl* utility shall conform to XBD [Section 12.2](#) (on page 215). Only one file can be named.

108592 The following options shall be supported:

108593 **-b *type*** Specify which logical page body lines shall be numbered. Recognized *types* and  
 108594 their meaning are:

108595 **a** Number all lines.

108596 **t** Number only non-empty lines.

108597 **n** No line numbering.

108598 **pstring** Number only lines that contain the basic regular expression specified in  
 108599 *string*.

108600 The default *type* for logical page body shall be **t** (text lines numbered).

108601 **-d *delim*** Specify the delimiter characters that indicate the start of a logical page section.  
 108602 These can be changed from the default characters "\: \" to two user-specified  
 108603 characters. If only one character is entered, the second character shall remain the  
 108604 default character ' : '.

108605 **-f *type*** Specify the same as **b *type*** except for footer. The default for logical page footer shall  
 108606 be **n** (no lines numbered).

108607 **-h *type*** Specify the same as **b *type*** except for header. The default *type* for logical page  
 108608 header shall be **n** (no lines numbered).

- 108609        **-i** *incr*        Specify the increment value used to number logical page lines. The default shall be  
108610        1.
- 108611        **-l** *num*         Specify the number of blank lines to be considered as one. For example, **-l 2** results  
108612        in only the second adjacent blank line being numbered (if the appropriate **-h a**,  
108613        **-b a**, or **-f a** option is set). The default shall be 1.
- 108614        **-n** *format*       Specify the line numbering format. Recognized values are: **ln**, left justified, leading  
108615        zeros suppressed; **rn**, right justified, leading zeros suppressed; **rz**, right justified,  
108616        leading zeros kept. The default *format* shall be **rn** (right justified).
- 108617        **-p**             Specify that numbering should not be restarted at logical page delimiters.
- 108618        **-s** *sep*         Specify the characters used in separating the line number and the corresponding  
108619        text line. The default *sep* shall be a <tab>.
- 108620        **-v** *startnum*    Specify the initial value used to number logical page lines. The default shall be 1.
- 108621        **-w** *width*       Specify the number of characters to be used for the line number. The default *width*  
108622        shall be 6.

#### 108623 OPERANDS

108624        The following operand shall be supported:

- 108625        *file*            A pathname of a text file to be line-numbered.

#### 108626 STDIN

108627        The standard input shall be used if no *file* operand is specified, and shall be used if the *file*  
108628        operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
108629        the standard input shall not be used. See the INPUT FILES section.

#### 108630 INPUT FILES

108631        The input file shall be a text file.

#### 108632 ENVIRONMENT VARIABLES

108633        The following environment variables shall affect the execution of *nl*:

- 108634        *LANG*            Provide a default value for the internationalization variables that are unset or null.  
108635        (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
108636        variables used to determine the values of locale categories.)

- 108637        *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
108638        internationalization variables.

#### 108639 *LC\_COLLATE*

108640        Determine the locale for the behavior of ranges, equivalence classes, and multi-  
108641        character collating elements within regular expressions.

- 108642        *LC\_CTYPE*       Determine the locale for the interpretation of sequences of bytes of text data as  
108643        characters (for example, single-byte as opposed to multi-byte characters in  
108644        arguments and input files), the behavior of character classes within regular  
108645        expressions, and for deciding which characters are in character class **graph** (for the  
108646        **-b t**, **-f t**, and **-h t** options).

#### 108647 *LC\_MESSAGES*

108648        Determine the locale that should be used to affect the format and contents of  
108649        diagnostic messages written to standard error.

- 108650 *NLSPATH* Determine the location of messages objects and message catalogs.
- 108651 **ASYNCHRONOUS EVENTS**
- 108652 Default.
- 108653 **STDOUT**
- 108654 The standard output shall be a text file in the following format:
- 108655 "%s%s%s", <line number>, <separator>, <input line>
- 108656 where <line number> is one of the following numeric formats:
- 108657 %6d When the **rn** format is used (the default; see **-n**).
- 108658 %06d When the **rz** format is used.
- 108659 %-6d When the **ln** format is used.
- 108660 <empty> When line numbers are suppressed for a portion of the page; the <separator> is also  
108661 suppressed.
- 108662 In the preceding list, the number 6 is the default width; the **-w** option can change this value.
- 108663 **STDERR**
- 108664 The standard error shall be used only for diagnostic messages.
- 108665 **OUTPUT FILES**
- 108666 None.
- 108667 **EXTENDED DESCRIPTION**
- 108668 None.
- 108669 **EXIT STATUS**
- 108670 The following exit values shall be returned:
- 108671 0 Successful completion.
- 108672 >0 An error occurred.
- 108673 **CONSEQUENCES OF ERRORS**
- 108674 Default.
- 108675 **APPLICATION USAGE**
- 108676 In using the **-d** *delim* option, care should be taken to escape characters that have special meaning  
108677 to the command interpreter.
- 108678 **EXAMPLES**
- 108679 The command:
- 108680 `nl -v 10 -i 10 -d \!+ file1`
- 108681 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is  
108682 "!+". Note that the "!" has to be escaped when using *csh* as a command interpreter because of  
108683 its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any  
108684 harm.
- 108685 **RATIONALE**
- 108686 None.

108687 **FUTURE DIRECTIONS**

108688 None.

108689 **SEE ALSO**108690 *pr*108691 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)108692 **CHANGE HISTORY**

108693 First released in Issue 2.

108694 **Issue 5**108695 The option [-f *type*] is added to the SYNOPSIS. The option descriptions are presented in  
108696 alphabetic order. The description of -bt is changed to ``Number only non-empty lines''.108697 **Issue 6**108698 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand  
108699 is removed.108700 **Issue 7**

108701 Austin Group Interpretation 1003.1-2001 #092 is applied.

108702 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

108703 **Issue 8**108704 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.



108705 **NAME**108706 nm — write the name list of an object file (**DEVELOPMENT**)108707 **SYNOPSIS**108708 SD nm [-APv] [-g|-u] [-t *format*] *file...*108709 XSI nm [-APv] [-efox] [-g|-u] [-t *format*] *file...*108710 **DESCRIPTION**

108711 The *nm* utility shall display symbolic information appearing in the object file, executable file, or  
 108712 object-file library named by *file*. If no symbolic information is available for a valid input file, the  
 108713 *nm* utility shall report that fact, but not consider it an error condition.

108714 XSI The default base used when numeric values are written is unspecified. On XSI-conformant  
 108715 systems, it shall be decimal if the **-P** option is not specified.

108716 **OPTIONS**108717 The *nm* utility shall conform to XBD [Section 12.2](#) (on page 215).

108718 The following options shall be supported:

108719 **-A** Write the full pathname or library name of an object on each line.108720 XSI **-e** Write only external (global) and static symbol information.

108721 XSI **-f** Produce full output. Write redundant symbols (**.text**, **.data**, and **.bss**), normally  
 108722 suppressed.

108723 **-g** Write only external (global) symbol information.108724 XSI **-o** Write numeric values in octal (equivalent to **-t o**).108725 **-P** Write information in a portable output format, as specified in the STDOUT section.

108726 **-t *format*** Write each numeric value in the specified format. The format shall be dependent  
 108727 on the single character used as the *format* option-argument:

108728 XSI d decimal (default if **-P** is not specified).

108729 o octal.

108730 x hexadecimal (default if **-P** is specified).108731 **-u** Write only undefined symbols.108732 **-v** Sort output by value instead of by symbol name.108733 XSI **-x** Write numeric values in hexadecimal (equivalent to **-t x**).108734 **OPERANDS**

108735 The following operand shall be supported:

108736 *file* A pathname of an object file, executable file, or object-file library.108737 **STDIN**

108738 See the INPUT FILES section.

108739 **INPUT FILES**

108740 The input file shall be an object file, an object-file library whose format is the same as those  
 108741 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept  
 108742 additional implementation-defined object library formats for the input file.

108743 **ENVIRONMENT VARIABLES**

108744 The following environment variables shall affect the execution of *nm*:

108745 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 108746 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 108747 variables used to determine the values of locale categories.)

108748 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 108749 internationalization variables.

108750 *LC\_COLLATE*

108751 Determine the locale for character collation information for the symbol-name and  
 108752 symbol-value collation sequences.

108753 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 108754 characters (for example, single-byte as opposed to multi-byte characters in  
 108755 arguments).

108756 *LC\_MESSAGES*

108757 Determine the locale that should be used to affect the format and contents of  
 108758 diagnostic messages written to standard error.

108759 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

108760 **ASYNCHRONOUS EVENTS**

108761 Default.

108762 **STDOUT**

108763 If symbolic information is present in the input files, then for each file or for each member of an  
 108764 archive, the *nm* utility shall write the following information to standard output. By default, the  
 108765 format is unspecified, but the output shall be sorted by symbol name according to the collation  
 108766 sequence in the current locale.

- 108767 • Library or object name, if *-A* is specified
- 108768 • Symbol name
- 108769 • Symbol type, which shall either be one of the following single characters or an  
 108770 implementation-defined type represented by a single character:
  - 108771 A Global absolute symbol.
  - 108772 a Local absolute symbol.
  - 108773 B Global ``bss'' (that is, uninitialized data space) symbol.
  - 108774 b Local bss symbol.
  - 108775 D Global data symbol.
  - 108776 d Local data symbol.
  - 108777 T Global text symbol.
  - 108778 t Local text symbol.
  - 108779 U Undefined symbol.
- 108780 • Value of the symbol
- 108781 • The size associated with the symbol, if applicable

108782 This information may be supplemented by additional information specific to the

108783 implementation.

108784 If the **-P** option is specified, the previous information shall be displayed using the following  
108785 portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified,  
108786 respectively:

108787 "%s%s %s %d %d\n", <library/object name>, <name>, <type>,  
108788 <value>, <size>

108789 "%s%s %s %o %o\n", <library/object name>, <name>, <type>,  
108790 <value>, <size>

108791 "%s%s %s %x %x\n", <library/object name>, <name>, <type>,  
108792 <value>, <size>

108793 where <library/object name> shall be formatted as follows:

- 108794 • If **-A** is not specified, <library/object name> shall be an empty string.
- 108795 • If **-A** is specified and the corresponding *file* operand does not name a library:  
108796 "%s: ", <file>
- 108797 • If **-A** is specified and the corresponding *file* operand names a library. In this case,  
108798 <object file> shall name the object file in the library containing the symbol being described:  
108799 "%s[%s]: ", <file>, <object file>

108800 If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is  
108801 specified and it names a library, *nm* shall write a line identifying the object containing the  
108802 following symbols before the lines containing those symbols, in the form:

- 108803 • If the corresponding *file* operand does not name a library:  
108804 "%s:\n", <file>
- 108805 • If the corresponding *file* operand names a library; in this case, <object file> shall be the  
108806 name of the file in the library containing the following symbols:  
108807 "%s[%s]:\n", <file>, <object file>

108808 If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.

#### 108809 **STDERR**

108810 The standard error shall be used only for diagnostic messages.

#### 108811 **OUTPUT FILES**

108812 None.

#### 108813 **EXTENDED DESCRIPTION**

108814 None.

#### 108815 **EXIT STATUS**

108816 The following exit values shall be returned:

108817 0 Successful completion.

108818 >0 An error occurred.

#### 108819 **CONSEQUENCES OF ERRORS**

108820 Default.

**108821 APPLICATION USAGE**

108822 Mechanisms for dynamic linking make this utility less meaningful when applied to an  
108823 executable file because a dynamically linked executable may omit numerous library routines  
108824 that would be found in a statically linked executable.

**108825 EXAMPLES**

108826 None.

**108827 RATIONALE**

108828 Historical implementations of *nm* have used different bases for numeric output and supplied  
108829 different default types of symbols that were reported. The `-t format` option, similar to that used  
108830 in *od* and *strings*, can be used to specify the numeric base; `-g` and `-u` can be used to restrict the  
108831 amount of output or the types of symbols included in the output.

108832 The compromise of using `-t format` versus using `-d`, `-o`, and other similar options was necessary  
108833 because of differences in the meaning of `-o` between implementations. The `-o` option from BSD  
108834 has been provided here as `-A` to avoid confusion with the `-o` from System V (which has been  
108835 provided here as `-t` and as `-o` on XSI-conformant systems).

108836 The option list was significantly reduced from that provided by historical implementations.

108837 The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified  
108838 default output.

108839 It was recognized that mechanisms for dynamic linking make this utility less meaningful when  
108840 applied to an executable file (because a dynamically linked executable file may omit numerous  
108841 library routines that would be found in a statically linked executable file), but the value of *nm*  
108842 during software development was judged to outweigh other limitations.

108843 The default output format of *nm* is not specified because of differences in historical  
108844 implementations. The `-P` option was added to allow some type of portable output format. After  
108845 a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to  
108846 create one that did not match the current format of any of these four systems. The format  
108847 devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary  
108848 depending on locale (because no English descriptions are included). All of the systems currently  
108849 have the information available to use this format.

108850 The format given in *nm* STDOUT uses `<space>` characters between the fields, which may be any  
108851 number of `<blank>` characters required to align the columns. The single-character types were  
108852 selected to match historical practice, and the requirement that implementation additions also be  
108853 single characters made parsing the information easier for shell scripts.

**108854 FUTURE DIRECTIONS**

108855 If this utility is directed to display a pathname that contains any bytes that have the encoded  
108856 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
108857 format being used, implementations are encouraged to treat this as an error. A future version of  
108858 this standard may require implementations to treat this as an error.

**108859 SEE ALSO**

108860 *ar*, *c17*

108861 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**108862 CHANGE HISTORY**

108863 First released in Issue 2.

108864 **Issue 6**

108865 This utility is marked as supported when both the User Portability Utilities option and the  
108866 Software Development Utilities option are supported.

108867 **Issue 7**

108868 The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is  
108869 now an option for interactive utilities.

108870 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

108871 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0125 [263] and XCU/TC1-2008/0126  
108872 [263] are applied.

108873 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0148 [744] is applied.

108874 **Issue 8**

108875 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
108876 directed to display a pathname that contains any bytes that have the encoded value of a  
108877 <newline> character when <newline> is a terminator or separator in the output format being  
108878 used.

108879 Austin Group Defect 1062 is applied, inserting an empty line between the two SYNOPSIS forms.

108880 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

108881 **NAME**

108882       nohup — invoke a utility immune to hangups

108883 **SYNOPSIS**108884       nohup *utility* [*argument...*]108885 **DESCRIPTION**

108886       The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied  
 108887       as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be  
 108888       set to be ignored.

108889       If standard input is associated with a terminal, the *nohup* utility may redirect standard input  
 108890       from an unspecified file.

108891       If the standard output is a terminal, all output written by the named *utility* to its standard output  
 108892       shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot  
 108893       be created or opened for appending, the output shall be appended to the end of the file  
 108894       **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be  
 108895       created or opened for appending, *utility* shall not be invoked. If a file is created, the file's  
 108896       permission bits shall be set to S\_IRUSR | S\_IWUSR.

108897       If standard error is a terminal and standard output is open but is not a terminal, all output  
 108898       written by the named utility to its standard error shall be redirected to the same open file  
 108899       description as the standard output. If standard error is a terminal and standard output either is a  
 108900       terminal or is closed, the same output shall instead be appended to the end of the **nohup.out** file  
 108901       as described above.

108902 **OPTIONS**

108903       None.

108904 **OPERANDS**

108905       The following operands shall be supported:

108906       *utility*       The name of a utility that is to be invoked. If the *utility* operand names any of the  
 108907       special built-in utilities in [Section 2.15](#) (on page 2526), the results are undefined.

108908       *argument*     Any string to be supplied as an argument when invoking the utility named by the  
 108909       *utility* operand.

108910 **STDIN**

108911       Not used.

108912 **INPUT FILES**

108913       None.

108914 **ENVIRONMENT VARIABLES**108915       The following environment variables shall affect the execution of *nohup*:

108916       *HOME*       Determine the pathname of the user's home directory: if the output file **nohup.out**  
 108917       cannot be created in the current directory, the *nohup* utility shall use the directory  
 108918       named by *HOME* to create the file.

108919       *LANG*       Provide a default value for the internationalization variables that are unset or null.  
 108920       (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 108921       variables used to determine the values of locale categories.)

108922       *LC\_ALL*      If set to a non-empty string value, override the values of all the other  
 108923       internationalization variables.

- 108924            *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
108925            characters (for example, single-byte as opposed to multi-byte characters in  
108926            arguments).
- 108927            *LC\_MESSAGES*  
108928            Determine the locale that should be used to affect the format and contents of  
108929            diagnostic messages written to standard error.
- 108930 XSI        *NLSPATH*    Determine the location of messages objects and message catalogs.
- 108931            *PATH*        Determine the search path that is used to locate the utility to be invoked. See XBD  
108932            [Chapter 8](#) (on page 167).
- 108933 **ASYNCHRONOUS EVENTS**  
108934            The *nohup* utility shall take the standard action for all signals except that **SIGHUP** shall be  
108935            ignored.
- 108936 **STDOUT**  
108937            If the standard output is not a terminal, the standard output of *nohup* shall be the standard  
108938            output generated by the execution of the *utility* specified by the operands. Otherwise, nothing  
108939            shall be written to the standard output.
- 108940 **STDERR**  
108941            If the standard output is a terminal, a message shall be written to the standard error, indicating  
108942            the name of the file to which the output is being appended. The name of the file shall be either  
108943            **nohup.out** or **\$HOME/nohup.out**.
- 108944 **OUTPUT FILES**  
108945            Output written by the named utility is appended to the file **nohup.out** (or **\$HOME/nohup.out**),  
108946            if the conditions hold as described in the **DESCRIPTION**.
- 108947 **EXTENDED DESCRIPTION**  
108948            None.
- 108949 **EXIT STATUS**  
108950            The following exit values shall be returned:
- 108951            126        The utility specified by *utility* was found but could not be invoked.
- 108952            127        An error occurred in the *nohup* utility or the utility specified by *utility* could not be  
108953            found.
- 108954            Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.
- 108955 **CONSEQUENCES OF ERRORS**  
108956            Default.
- 108957 **APPLICATION USAGE**  
108958            The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit  
108959            code 127 if a utility to be invoked cannot be found, so that applications can distinguish “failure  
108960            to find a utility” from “invoked utility exited with an error indication”. However, the *command*  
108961            and *nohup* utilities also use exit code 127 when an error occurs in those utilities, which means  
108962            exit code 127 is not universally a “not found” indicator. The value 127 was chosen because it is  
108963            not commonly used for other meanings; most utilities use small values for “normal error  
108964            conditions” and the values above 128 can be confused with termination due to receipt of a  
108965            signal. The value 126 was chosen in a similar manner to indicate that the utility could be found,  
108966            but not invoked. Some scripts produce meaningful error messages differentiating the 126 and  
108967            127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that  
108968            uses 127 when all attempts to *exec* the utility fail with **[ENOENT]**, and uses 126 when any

108969 attempt to *exec* the utility fails for any other reason.

#### 108970 EXAMPLES

108971 It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by  
108972 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
108973 the *nohup* applies to everything in the file.

108974 Alternatively, the following command can be used to apply *nohup* to a complex command:

```
108975 nohup sh -c -- 'complex-command-line' </dev/null
```

#### 108976 RATIONALE

108977 The 4.3 BSD version ignores SIGTERM and SIGHUP, and if **./nohup.out** cannot be used, it fails  
108978 instead of trying to use **\$HOME/nohup.out**.

108979 The *cs* utility has a built-in version of *nohup* that acts differently from the *nohup* defined in this  
108980 volume of POSIX.1-2024.

108981 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
108982 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*  
108983 includes user application programs and shell scripts, not just the standard utilities.

108984 Historical versions of the *nohup* utility use default file creation semantics. Some more recent  
108985 versions use the permissions specified here as an added security precaution.

108986 Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore  
108987 SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several  
108988 reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this  
108989 volume of POSIX.1-2024.

108990 Historical versions of *nohup* did not affect standard input, but that causes problems in the  
108991 common scenario where the user logs into a system, types the command:

```
108992 nohup make &
```

108993 at the prompt, and then logs out. If standard input is not affected by *nohup*, the login session  
108994 may not terminate for quite some time, since standard input remains open until *make* exits. To  
108995 avoid this problem, POSIX.1-2024 allows implementations to redirect standard input if it is a  
108996 terminal. Since the behavior is implementation-defined, portable applications that may run into  
108997 the problem should redirect standard input themselves. For example, instead of:

```
108998 nohup make &
```

108999 an application can invoke:

```
109000 nohup make </dev/null &
```

#### 109001 FUTURE DIRECTIONS

109002 None.

#### 109003 SEE ALSO

109004 [Chapter 2](#) (on page 2472), *sh*

109005 [XBD Chapter 8](#) (on page 167)

109006 XSH *signal()*

#### 109007 CHANGE HISTORY

109008 First released in Issue 2.



109009 **Issue 7**

109010 Austin Group Interpretations 1003.1-2001 #104, #105, and #106 are applied.

109011 **Issue 8**

109012 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

109013 Austin Group Defect 1530 is applied, changing ``sh -c'' to ``sh -c --''.

109014 Austin Group Defect 1586 is applied, adding the *timeout* utility.

109015 Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

109016 **NAME**

109017 od — dump files in various formats

109018 **SYNOPSIS**109019 od [-v] [-A *address\_base*] [-j *skip*] [-N *count*] [-t *type\_string*]...  
109020 [*file*...]109021 XSI od [-bcdosx] [*file*] [[+]offset[.][b]]109022 **DESCRIPTION**109023 The *od* utility shall write the contents of its input files to standard output in a user-specified  
109024 format.109025 **OPTIONS**109026 The *od* utility shall conform to XBD Section 12.2 (on page 215), except that the order of  
109027 XSI presentation of the **-t** options and the **-bcdosx** options is significant.

109028 The following options shall be supported:

109029 **-A** *address\_base*109030 Specify the input offset base. See the EXTENDED DESCRIPTION section. The  
109031 application shall ensure that the *address\_base* option-argument is a character. The  
109032 characters 'd', 'o', and 'x' specify that the offset base shall be written in  
109033 decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the  
109034 offset shall not be written.109035 XSI **-b** Interpret bytes in octal. This shall be equivalent to **-t o1**.109036 XSI **-c** Interpret bytes as characters specified by the current setting of the *LC\_CTYPE*  
109037 category. Certain non-graphic characters appear as C escapes: "NUL=\0",  
109038 "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal  
109039 numbers.109040 XSI **-d** Interpret *words* (two-byte units) in unsigned decimal. This shall be equivalent to  
109041 **-t u2**.109042 **-j** *skip* Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or  
109043 seek past the first *skip* bytes in the concatenated input files. If the combined input is  
109044 not at least *skip* bytes long, the *od* utility shall write a diagnostic message to  
109045 standard error and exit with a non-zero exit status.109046 By default, the *skip* option-argument shall be interpreted as a decimal number.  
109047 With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;  
109048 otherwise, with a leading '0', the offset shall be interpreted as an octal number.  
109049 Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted  
109050 as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip* number is  
109051 hexadecimal, any appended 'b' shall be considered to be the final hexadecimal  
109052 digit.109053 **-N** *count* Format no more than *count* bytes of input. By default, *count* shall be interpreted as  
109054 a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a  
109055 hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an  
109056 octal number. If *count* bytes of input (after successfully skipping, if **-j** *skip* is  
109057 specified) are not available, it shall not be considered an error; the *od* utility shall  
109058 format the input that is available.

|        |     |                                      |                                                                                                                                            |
|--------|-----|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 109059 | XSI | <b>-o</b>                            | Interpret <i>words</i> (two-byte units) in octal. This shall be equivalent to <b>-t o2</b> .                                               |
| 109060 | XSI | <b>-s</b>                            | Interpret <i>words</i> (two-byte units) in signed decimal. This shall be equivalent to                                                     |
| 109061 |     | <b>-t d2</b> .                       |                                                                                                                                            |
| 109062 |     | <b>-t</b> <i>type_string</i>         |                                                                                                                                            |
| 109063 |     |                                      | Specify one or more output types. See the EXTENDED DESCRIPTION section. The                                                                |
| 109064 |     |                                      | application shall ensure that the <i>type_string</i> option-argument is a string specifying                                                |
| 109065 |     |                                      | the types to be used when writing the input data. The string shall consist of the                                                          |
| 109066 |     |                                      | type specification characters <i>a</i> , <i>c</i> , <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> , specifying named character, |
| 109067 |     |                                      | character, signed decimal, floating point, octal, unsigned decimal, and                                                                    |
| 109068 |     |                                      | hexadecimal, respectively. The type specification characters <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be               |
| 109069 |     |                                      | followed by an optional unsigned decimal integer that specifies the number of                                                              |
| 109070 |     |                                      | bytes to be transformed by each instance of the output type. The type specification                                                        |
| 109071 |     |                                      | character <i>f</i> can be followed by an optional <i>F</i> , <i>D</i> , or <i>L</i> indicating that the conversion                         |
| 109072 |     |                                      | should be applied to an item of type <b>float</b> , <b>double</b> , or <b>long double</b> , respectively.                                  |
| 109073 |     |                                      | The type specification characters <i>d</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be followed by an optional <i>C</i> , <i>S</i> ,       |
| 109074 |     |                                      | <i>I</i> , or <i>L</i> indicating that the conversion should be applied to an item of type <b>char</b> ,                                   |
| 109075 |     |                                      | <b>short</b> , <b>int</b> , or <b>long</b> , respectively. Multiple types can be concatenated within the                                   |
| 109076 |     |                                      | same <i>type_string</i> and multiple <b>-t</b> options can be specified. Output lines shall be                                             |
| 109077 |     |                                      | written for each type specified in the order in which the type specification                                                               |
| 109078 |     |                                      | characters are specified.                                                                                                                  |
| 109079 |     | <b>-v</b>                            | Write all input data. Without the <b>-v</b> option, any number of groups of output lines,                                                  |
| 109080 |     |                                      | which would be identical to the immediately preceding group of output lines                                                                |
| 109081 |     |                                      | (except for the byte offsets), shall be replaced with a line containing only an                                                            |
| 109082 |     |                                      | <asterisk> ('*').                                                                                                                          |
| 109083 | XSI | <b>-x</b>                            | Interpret <i>words</i> (two-byte units) in hexadecimal. This shall be equivalent to <b>-t x2</b> .                                         |
| 109084 | XSI |                                      | Multiple types can be specified by using multiple <b>-bcdostx</b> options. Output lines are written for                                    |
| 109085 |     |                                      | each type specified in the order in which the types are specified.                                                                         |
| 109086 |     | <b>OPERANDS</b>                      |                                                                                                                                            |
| 109087 |     |                                      | The following operands shall be supported:                                                                                                 |
| 109088 |     | <i>file</i>                          | A pathname of a file to be read. If no <i>file</i> operands are specified, the standard input                                              |
| 109089 |     |                                      | shall be used.                                                                                                                             |
| 109090 |     |                                      | If there are no more than two operands, none of the <b>-A</b> , <b>-j</b> , <b>-N</b> , <b>-t</b> , or <b>-v</b> options is                |
| 109091 |     |                                      | specified, and either of the following is true: the first character of the last operand                                                    |
| 109092 |     |                                      | is a <plus-sign> ('+'), or there are two operands and the first character of the last                                                      |
| 109093 | XSI |                                      | operand is numeric; the last operand shall be interpreted as an offset operand on                                                          |
| 109094 |     |                                      | XSI-conformant systems. Under these conditions, the results are unspecified on                                                             |
| 109095 |     |                                      | systems that are not XSI-conformant systems.                                                                                               |
| 109096 | XSI | <b>[+]<i>offset</i>[.][<i>b</i>]</b> | The <i>offset</i> operand specifies the offset in the file where dumping is to commence.                                                   |
| 109097 |     |                                      | This operand is normally interpreted as octal bytes. If '.' is appended, the offset                                                        |
| 109098 |     |                                      | shall be interpreted in decimal. If 'b' is appended, the offset shall be interpreted                                                       |
| 109099 |     |                                      | in units of 512 bytes.                                                                                                                     |
| 109100 |     | <b>STDIN</b>                         |                                                                                                                                            |
| 109101 |     |                                      | The standard input shall be used if no <i>file</i> operands are specified, and shall be used if a <i>file</i>                              |
| 109102 |     |                                      | operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,                                                 |
| 109103 |     |                                      | the standard input shall not be used. See the INPUT FILES section.                                                                         |

109104 **INPUT FILES**

109105 The input files can be any file type.

109106 **ENVIRONMENT VARIABLES**

109107 The following environment variables shall affect the execution of *od*:

109108 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 109109 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 109110 variables used to determine the values of locale categories.)

109111 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 109112 internationalization variables.

109113 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 109114 characters (for example, single-byte as opposed to multi-byte characters in  
 109115 arguments and input files).

109116 *LC\_MESSAGES*

109117 Determine the locale that should be used to affect the format and contents of  
 109118 diagnostic messages written to standard error.

109119 *LC\_NUMERIC*

109120 Determine the locale for selecting the radix character used when writing floating-  
 109121 point formatted output.

109122 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

109123 **ASYNCHRONOUS EVENTS**

109124 Default.

109125 **STDOUT**

109126 See the EXTENDED DESCRIPTION section.

109127 **STDERR**

109128 The standard error shall be used only for diagnostic messages.

109129 **OUTPUT FILES**

109130 None.

109131 **EXTENDED DESCRIPTION**

109132 The *od* utility shall copy sequentially each input file to standard output, transforming the input  
 109133 XSI data according to the output types specified by the *-t* option or the *-bcdosx* options. If no  
 109134 output type is specified, the default output shall be as if *-t oS* had been specified.

109135 The number of bytes transformed by the output type specifier *c* may be variable depending on  
 109136 the *LC\_CTYPE* category.

109137 The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds  
 109138 to the various C-language types as follows. If the *c17* compiler is present on the system, these  
 109139 specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes  
 109140 may vary among systems that conform to POSIX.1-2024.

- 109141 • For the type specifier characters *d*, *o*, *u*, and *x*, the default number of bytes shall
- 109142 correspond to the size of the underlying implementation's basic integer type. For these
- 109143 specifier characters, the implementation shall support values of the optional number of
- 109144 bytes to be converted corresponding to the number of bytes in the C-language types **char**,
- 109145 **short**, **int**, and **long**. These numbers can also be specified by an application as the
- 109146 characters 'C', 'S', 'I', and 'L', respectively. The implementation shall also support
- 109147 the values 1, 2, 4, and 8, even if it provides no C-Language types of those sizes. The

109148 implementation shall support the decimal value corresponding to the C-language type  
 109149 **long long**. The byte order used when interpreting numeric values is implementation-  
 109150 defined, but shall correspond to the order in which a constant of the corresponding type is  
 109151 stored in memory on the system.

109152 • For the type specifier character  $\epsilon$ , the default number of bytes shall correspond to the  
 109153 number of bytes in the underlying implementation's basic double precision floating-point  
 109154 data type. The implementation shall support values of the optional number of bytes to be  
 109155 converted corresponding to the number of bytes in the C-language types **float**, **double**,  
 109156 and **long double**. These numbers can also be specified by an application as the characters  
 109157 'F', 'D', and 'L', respectively.

109158 The type specifier character  $a$  specifies that bytes shall be interpreted as named characters from  
 109159 the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least  
 109160 significant seven bits of each byte shall be used for this type specification. Bytes with the values  
 109161 listed in the following table shall be written using the corresponding names for those characters.

109162 **Table 3-14** Named Characters in *od*

| Value | Name | Value | Name | Value | Name      | Value | Name |
|-------|------|-------|------|-------|-----------|-------|------|
| \000  | nul  | \001  | soh  | \002  | stx       | \003  | etx  |
| \004  | eot  | \005  | enq  | \006  | ack       | \007  | bel  |
| \010  | bs   | \011  | ht   | \012  | lf or nl* | \013  | vt   |
| \014  | ff   | \015  | cr   | \016  | so        | \017  | si   |
| \020  | dle  | \021  | dc1  | \022  | dc2       | \023  | dc3  |
| \024  | dc4  | \025  | nak  | \026  | syn       | \027  | etb  |
| \030  | can  | \031  | em   | \032  | sub       | \033  | esc  |
| \034  | fs   | \035  | gs   | \036  | rs        | \037  | us   |
| \040  | sp   | \177  | del  |       |           |       |      |

109173 **Note:** The "\012" value may be written either as **lf** or **nl**.

109174 The type specifier character  $c$  specifies that bytes shall be interpreted as characters specified by  
 109175 the current setting of the *LC\_CTYPE* locale category. Characters listed in the table in XBD  
 109176 [Chapter 5](#) (on page 113) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be written as  
 109177 the corresponding escape sequences, except that <backslash> shall be written as a single  
 109178 <backslash> and a NUL shall be written as '\0'. Other non-printable characters shall be  
 109179 written as one three-digit octal number for each byte in the character. Printable multi-byte  
 109180 characters shall be written in the area corresponding to the first byte of the character; the two-  
 109181 character sequence "\*\*\*" shall be written in the area corresponding to each remaining byte in the  
 109182 character, as an indication that the character is continued. When either the  $-j$  *skip* or  $-N$  *count*  
 109183 option is specified along with the  $c$  type specifier, and this results in an attempt to start or finish  
 109184 in the middle of a multi-byte character, the result is implementation-defined.

109185 The input data shall be manipulated in blocks, where a block is defined as a multiple of the least  
 109186 common multiple of the number of bytes transformed by the specified output types. If the least  
 109187 common multiple is greater than 16, the results are unspecified. Each input block shall be  
 109188 written as transformed by each output type, one per written line, in the order that the output  
 109189 types were specified. If the input block size is larger than the number of bytes transformed by  
 109190 the output type, the output type shall sequentially transform the parts of the input block, and  
 109191 the output from each of the transformations shall be separated by one or more <blank>  
 109192 characters.

109193 If, as a result of the specification of the  $-N$  option or end-of-file being reached on the last input  
 109194 file, input data only partially satisfies an output type, the input shall be extended sufficiently

109195 with null bytes to write the last byte of the input.

109196 Unless **-A n** is specified, the first output line produced for each input block shall be preceded by  
 109197 the input offset, cumulative across input files, of the next byte to be written. The format of the  
 109198 input offset is unspecified; however, it shall not contain any <blank> characters, shall start at the  
 109199 first character of the output line, and shall be followed by one or more <blank> characters. In  
 109200 addition, the offset of the byte following the last byte written shall be written after all the input  
 109201 data has been processed, but shall not be followed by any <blank> characters. If **-A n** is  
 109202 specified, it is unspecified whether the line that would contain this final offset is written as an  
 109203 empty line or is not written.

109204 If no **-A** option is specified, the input offset base is unspecified.

#### 109205 EXIT STATUS

109206 The following exit values shall be returned:

109207 0 All input files were processed successfully.

109208 >0 An error occurred.

#### 109209 CONSEQUENCES OF ERRORS

109210 Default.

#### 109211 APPLICATION USAGE

109212 XSI-conformant applications are warned not to use filenames starting with '+' or a first  
 109213 operand starting with a numeric character so that the old functionality can be maintained by  
 109214 implementations, unless they specify one of the **-A**, **-j**, or **-N** options. To guarantee that one of  
 109215 these filenames is always interpreted as a filename, an application could always specify the  
 109216 address base format with the **-A** option.

#### 109217 EXAMPLES

109218 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as  
 109219 standard input to the command:

```
109220 od -A d -t a
```

109221 on an implementation using an input block size of 16 bytes, the standard output, independent of  
 109222 the current locale setting, would be similar to:

```
109223 0000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
109224 0000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
109225 0000032 sp ! " # $ % & ' ( ) * + , - . /
109226 0000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
109227 0000064 @ A B C D E F G H I J K L M N O
109228 0000080 P Q R S T U V W X Y Z [ \ ] ^ _
109229 0000096 ` a b c d e f g h i j k l m n o
109230 0000112 p q r s t u v w x y z { | } ~ del
109231 0000128
```

109232 Note that this volume of POSIX.1-2024 allows **nl** or **lf** to be used as the name for the  
 109233 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character  
 109234 **lf** (line feed), but traditional implementations have referred to this character as newline (**nl**) and  
 109235 the POSIX locale character set symbolic name for the corresponding character is a <newline>.

109236 The command:

```
109237 od -A o -t o2x2x -N 18
```

109238 on a system with 32-bit words and an implementation using an input block size of 16 bytes

109239 could write 18 bytes in approximately the following format:

```

109240 0000000 032056 031440 041123 042040 052516 044530 020043 031464
109241          342e  3320  4253  4420  554e  4958  2023  3334
109242          342e3320      42534420      554e4958      20233334
109243 0000020 032472
109244          353a
109245          353a0000
109246 0000022
```

109247 The command:

```
109248 od -A d -t f -t o4 -t x4 -N 24 -j 0x15
```

109249 on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point  
 109250 format) would skip 21 bytes of input data and then write 24 bytes in approximately the  
 109251 following format:

```

109252 0000000 1.0000000000000000e+00 1.5735000000000000e+01
109253 07774000000 00000000000 10013674121 35341217270
109254 3ff00000 00000000 402f3851 eb851eb8
109255 0000016 1.4066823000000000e+02
109256 10030312542 04370303230
109257 40619562 23e18698
109258 0000024
```

#### 109259 RATIONALE

109260 The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently  
 109261 *hexdump*. There were several objections to all of these based on the following reasons:

- 109262 • The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- 109263 • The *hexdump* description was much more complex than needed for a simple dump utility.
- 109264 • The *od* utility has been available on all historical implementations and there was no need to  
 109265 create a new name for a utility so similar to the historical *od* utility.

109266 The original reasons for not standardizing historical *od* were also fairly widespread. Those  
 109267 reasons are given below along with rationale explaining why the standard developers believe  
 109268 that this version does not suffer from the indicated problem:

- 109269 • The BSD and System V versions of *od* have diverged, and the intersection of features  
 109270 provided by both does not meet the needs of the user community. In fact, the System V  
 109271 version only provides a mechanism for dumping octal bytes and **shorts**, signed and  
 109272 unsigned decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability  
 109273 to dump **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned  
 109274 decimal, and hexadecimal **longs**. The version presented here provides more normalized  
 109275 forms for dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned  
 109276 decimal, and hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as  
 109277 current locale characters.
- 109278 • It would not be possible to come up with a compatible superset of the BSD and System V  
 109279 flags that met the requirements of the standard developers. The historical default *od* output  
 109280 is the specified default output of this utility. None of the option letters chosen for this  
 109281 version of *od* conflict with any of the options to historical versions of *od*.

109282 • On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps  
 109283 of **ints**, even in the BSD version. Because of the way options are named, the name space  
 109284 could not be extended to solve these problems. This is why the **-t** option was added (with  
 109285 type specifiers more closely matched to the *printf()* formats used in the rest of this volume  
 109286 of POSIX.1-2024) and the optional field sizes were added to the **d**, **f**, **o**, **u**, and **x** type  
 109287 specifiers. It is also one of the reasons why the historical practice was not mandated as a  
 109288 required obsolescent form of *od*. (Although the old versions of *od* are not listed as an  
 109289 obsolescent form, implementations are urged to continue to recognize the older forms for  
 109290 several more years.) The **a**, **c**, **f**, **o**, and **x** types match the meaning of the corresponding  
 109291 format characters in the historical implementations of *od* except for the default sizes of the  
 109292 fields converted. The **d** format is signed in this volume of POSIX.1-2024 to match the  
 109293 *printf()* notation. (Historical versions of *od* used **d** as a synonym for **u** in this version. The  
 109294 System V implementation uses **s** for signed decimal; BSD uses **i** for signed decimal and **s**  
 109295 for null-terminated strings.) Other than **d** and **u**, all of the type specifiers match format  
 109296 characters in the historical BSD version of *od*.

109297 The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are  
 109298 used even though it is recognized that there may be zero or more than one compiler for the  
 109299 C language on an implementation and that they may use different sizes for some of these  
 109300 types. (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**,  
 109301 while another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes  
 109302 **ints**, and 4 bytes **longs**.) Nonetheless, there has to be a basic size known by the  
 109303 implementation for these types, corresponding to the values reported by invocations of the  
 109304 *getconf* utility when called with *system\_var* operands {UCHAR\_MAX}, {USHORT\_MAX},  
 109305 {UINT\_MAX}, and {ULONG\_MAX} for the types **char**, **short**, **int**, and **long**, respectively.  
 109306 There are similar constants required by the ISO C standard, but not required by the System  
 109307 Interfaces volume of POSIX.1-2024 or this volume of POSIX.1-2024. They are  
 109308 {FLT\_MANT\_DIG}, {DBL\_MANT\_DIG}, and {LDBL\_MANT\_DIG} for the types **float**,  
 109309 **double**, and **long double**, respectively. If the optional *c17* utility is provided by the  
 109310 implementation and used as specified by this volume of POSIX.1-2024, these are the sizes  
 109311 that would be provided. If an option is used that specifies different sizes for these types,  
 109312 there is no guarantee that the *od* utility is able to interpret binary data output by such a  
 109313 program correctly.

109314 This volume of POSIX.1-2024 requires that the numeric values of these lengths be  
 109315 recognized by the *od* utility and that symbolic forms also be recognized. Thus, a  
 109316 conforming application can always look at an array of **unsigned long** data elements using  
 109317 *od -t uL*.

109318 • The method of specifying the format for the address field based on specifying a starting  
 109319 offset in a file unnecessarily tied the two together. The **-A** option now specifies the address  
 109320 base and the **-S** option specifies a starting offset.

109321 • It would be difficult to break the dependence on US ASCII to achieve an internationalized  
 109322 utility. It does not seem to be any harder for *od* to dump characters in the current locale  
 109323 than it is for the *ed* or *sed* **I** commands. The **c** type specifier does this without difficulty and  
 109324 is completely compatible with the historical implementations of the **c** format character  
 109325 when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The  
 109326 **a** type specifier (from the BSD **a** format character) was left as a portable means to dump  
 109327 ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by  
 109328 *pax* could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard  
 109329 as a subset of their base codeset.

109330 The use of **"\*\*"** as an indication of continuation of a multi-byte character in **c** specifier output



109331 was chosen based on seeing an implementation that uses this method. The continuation bytes  
 109332 have to be marked in a way that is not ambiguous with another single-byte or multi-byte  
 109333 character.

109334 An early proposal used `-S` and `-n`, respectively, for the `-j` and `-N` options eventually selected.  
 109335 These were changed to avoid conflicts with historical implementations.

109336 The original standard specified `-t o2` as the default when no output type was given. This was  
 109337 changed to `-t oS` (the length of a **short**) to accommodate a supercomputer implementation that  
 109338 historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not  
 109339 affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at  
 109340 the same time to address an historical implementation that had no two-byte data types in its C  
 109341 compiler.

109342 The use of a basic integer data type is intended to allow the implementation to choose a word  
 109343 size commonly used by applications on that architecture.

109344 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
 109345 but this has been modified in this version.

#### 109346 **FUTURE DIRECTIONS**

109347 All option and operand interfaces marked XSI may be removed in a future version.

#### 109348 **SEE ALSO**

109349 *c17, sed*

109350 XBD [Chapter 5](#) (on page 113), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 109351 **CHANGE HISTORY**

109352 First released in Issue 2.

#### 109353 **Issue 5**

109354 In the description of the `-c` option, the phrase “This is equivalent to `-t c.`” is deleted.

109355 The FUTURE DIRECTIONS section is modified.

#### 109356 **Issue 6**

109357 The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the  
 109358 revisions in the IEEE P1003.2b draft standard.

109359 The normative text is reworded to avoid use of the term “must” for application requirements.

109360 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/33 is applied, correcting the examples  
 109361 which used an undefined `-n` option, which should have been `-N`.

109362 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/19 is applied, removing text describing  
 109363 behavior on systems with bytes consisting of more than eight bits.

#### 109364 **Issue 7**

109365 Austin Group Interpretation 1003.1-2001 #092 is applied.

109366 SD5-XCU-ERN-37 is applied, updating the OPERANDS section.

109367 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

#### 109368 **Issue 8**

109369 Austin Group Defect 1017 is applied, clarifying that when `-A n` is specified, the line that would  
 109370 contain the final offset can either be written as an empty line or not be written.

109371 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

109372 **NAME**

109373 paste — merge corresponding or subsequent lines of files

109374 **SYNOPSIS**

109375 paste [-s] [-d *list*] *file...*

109376 **DESCRIPTION**

109377 The *paste* utility shall concatenate the corresponding lines of the given input files, and write the  
109378 resulting lines to standard output.

109379 The default operation of *paste* shall concatenate the corresponding lines of the input files. The  
109380 <newline> of every line except the line from the last input file shall be replaced with a <tab>.

109381 If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall  
109382 behave as though empty lines were read from the files on which end-of-file was detected, unless  
109383 the **-s** option is specified.

109384 **OPTIONS**

109385 The *paste* utility shall conform to XBD [Section 12.2](#) (on page 215).

109386 The following options shall be supported:

109387 **-d list** Unless a <backslash> character appears in *list*, each character in *list* is an element  
109388 specifying a delimiter character. If a <backslash> character appears in *list*, the  
109389 <backslash> character and one or more characters following it are an element  
109390 specifying a delimiter character as described below. These elements specify one or  
109391 more delimiters to use, instead of the default <tab>, to replace the <newline> of  
109392 the input lines. The elements in *list* shall be used circularly; that is, when the list is  
109393 exhausted the first element from the list is reused. When the **-s** option is specified:

- 109394 • The last <newline> in a file shall not be modified.
- 109395 • The delimiter shall be reset to the first element of *list* after each *file* operand is  
109396 processed.

109397 When the **-s** option is not specified:

- 109398 • The <newline> characters in the file specified by the last *file* operand shall  
109399 not be modified.
- 109400 • The delimiter shall be reset to the first element of list each time a line is  
109401 processed from each file.

109402 If a <backslash> character appears in *list*, it and the character following it shall be  
109403 used to represent the following delimiter characters:

109404 \n <newline>.

109405 \t <tab>.

109406 \\ <backslash> character.

109407 \0 Empty string (not a null character). If '\0' is immediately followed by the  
109408 character 'x', the character 'X', or any character defined by the *LC\_CTYPE*  
109409 **digit** keyword (see XBD [Chapter 7](#), on page 127), the results are unspecified.

109410 If any other characters follow the <backslash>, the results are unspecified.

109411 **-s** Concatenate all of the lines from each input file into one line of output per file, in  
109412 command line order. The <newline> of every line except the last line in each input  
109413 file shall be replaced with a <tab>, unless otherwise specified by the **-d** option. If  
109414 an input file is empty, the output line corresponding to that file shall consist of

- 109415 only a <newline> character.
- 109416 **OPERANDS**
- 109417 The following operand shall be supported:
- 109418 *file* A pathname of an input file. If '-' is specified for one or more of the *files*, the  
109419 standard input shall be used; the standard input shall be read one line at a time,  
109420 circularly, for each instance of '-'. Implementations shall support pasting of at  
109421 least 12 *file* operands.
- 109422 **STDIN**
- 109423 The standard input shall be used only if one or more *file* operands is '-'. See the INPUT FILES  
109424 section.
- 109425 **INPUT FILES**
- 109426 The input files shall be text files, except that line lengths shall be unlimited.
- 109427 **ENVIRONMENT VARIABLES**
- 109428 The following environment variables shall affect the execution of *paste*:
- 109429 *LANG* Provide a default value for the internationalization variables that are unset or null.  
109430 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables  
109431 used to determine the values of locale categories.)
- 109432 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
109433 internationalization variables.
- 109434 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
109435 characters (for example, single-byte as opposed to multi-byte characters in  
109436 arguments and input files).
- 109437 *LC\_MESSAGES*
- 109438 Determine the locale that should be used to affect the format and contents of  
109439 diagnostic messages written to standard error.
- 109440 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 109441 **ASYNCHRONOUS EVENTS**
- 109442 Default.
- 109443 **STDOUT**
- 109444 Concatenated lines of input files shall be separated by the <tab> (or other characters under the  
109445 control of the -d option) and terminated by a <newline>.
- 109446 **STDERR**
- 109447 The standard error shall be used only for diagnostic messages.
- 109448 **OUTPUT FILES**
- 109449 None.
- 109450 **EXTENDED DESCRIPTION**
- 109451 None.
- 109452 **EXIT STATUS**
- 109453 The following exit values shall be returned:
- 109454 0 Successful completion.
- 109455 >0 An error occurred.

109456 **CONSEQUENCES OF ERRORS**

109457 If one or more input files cannot be opened when the `-s` option is not specified, a diagnostic  
 109458 message shall be written to standard error, but no output is written to standard output. If the `-s`  
 109459 option is specified, the *paste* utility shall provide the default behavior described in [Section 1.4](#) (on  
 109460 page 2462).

109461 **APPLICATION USAGE**

109462 When the escape sequences of the *list* option-argument are used in a shell script, they must be  
 109463 quoted; otherwise, the shell treats the `<backslash>` as a special character.

109464 Conforming applications should only use the specific `<backslash>`-escaped delimiters presented  
 109465 in this volume of POSIX.1-2024. Historical implementations treat `'\x'`, where `'x'` is not in this  
 109466 list, as `'x'`, but future implementations are free to expand this list to recognize other common  
 109467 escapes similar to those accepted by *printf* and other standard utilities.

109468 Most of the standard utilities work on text files. The *cut* utility can be used to turn files with  
 109469 arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used  
 109470 to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
109471 cut -b 1-500 -n file > file1
109472 cut -b 501- -n file > file2
```

109473 creates **file1** (a text file) with lines no longer than 500 bytes (plus the `<newline>`) and **file2** that  
 109474 contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in  
 109475 *file* that are longer than `500 + {LINE_MAX}` bytes. The original file can be recreated from **file1**  
 109476 and **file2** using the command:

```
109477 paste -d "\0" file1 file2 > file
```

109478 The commands:

```
109479 paste -d "\0" ...
109480 paste -d "" ...
```

109481 are not necessarily equivalent; the latter is not specified by this volume of POSIX.1-2024 and  
 109482 may result in an error. The construct `'\0'` is used to mean “no separator” because historical  
 109483 versions of *paste* did not follow the syntax guidelines, and the command:

```
109484 paste -d"" ...
```

109485 could not be handled properly by *getopt()*.

109486 **EXAMPLES**

109487 1. Write out a directory in four columns:

```
109488 ls | paste - - - -
```

109489 2. Combine pairs of lines from a file into single lines:

```
109490 paste -s -d "\t\n" file
```

109491 **RATIONALE**

109492 None.

109493 **FUTURE DIRECTIONS**

109494 None.

109495 **SEE ALSO**109496 [Section 1.4](#) (on page 2462), *cut*, *grep*, *pr*109497 XBD [Chapter 7](#) (on page 127), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)109498 **CHANGE HISTORY**

109499 First released in Issue 2.

109500 **Issue 6**

109501 The normative text is reworded to avoid use of the term “must” for application requirements.

109502 **Issue 7**

109503 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

109504 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0149 [973] is applied.

109505 **Issue 8**109506 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

109507 **NAME**

109508 patch — apply changes to files

109509 **SYNOPSIS**

```
109510 patch [-blNR] [-c|-e|-n|-u] [-d dir] [-D define] [-i patchfile]
109511 [-o outfile] [-p num] [-r rejectfile] [file]
```

109512 **DESCRIPTION**

109513 The *patch* utility shall read a source (patch) file containing any of four forms of difference (diff)  
 109514 listings produced by the *diff* utility (normal, copied context, unified context, or in the style of *ed*)  
 109515 and apply those differences to a file. By default, *patch* shall read from the standard input.

109516 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a *-c*,  
 109517 *-e*, *-n*, or *-u* option.

109518 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they  
 109519 came from separate patch files. (In this case, the application shall ensure that the name of the  
 109520 patch file is determinable for each *diff* listing.)

109521 **OPTIONS**

109522 The *patch* utility shall conform to XBD [Section 12.2](#) (on page 215).

109523 The following options shall be supported:

109524 **-b** Save a copy of the original contents of each modified file, before the differences are  
 109525 applied, in a file of the same name with the suffix **.orig** appended to it. If the file  
 109526 already exists, it shall be overwritten; if multiple patches are applied to the same  
 109527 file, the **.orig** file shall be written only for the first patch. When the *-o outfile* option  
 109528 is also specified, *file.orig* shall not be created but, if *outfile* already exists, *outfile.orig*  
 109529 shall be created.

109530 **-c** Interpret the patch file as a copied context difference (the output of the utility *diff*  
 109531 when the *-c* or *-C* options are specified).

109532 **-d dir** Change the current directory to *dir* before processing as described in the  
 109533 EXTENDED DESCRIPTION section.

109534 **-D define** Mark changes with one of the following C preprocessor constructs:

```
109535 #ifdef define
109536 ...
109537 #endif
109538 #ifndef define
109539 ...
109540 #endif
```

109541 optionally combined with the C preprocessor construct **#else**. If the patched file is  
 109542 processed with the C preprocessor, where the macro *define* is defined, the output  
 109543 shall contain the changes from the patch file; otherwise, the output shall not  
 109544 contain the patches specified in the patch file.

109545 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.

109546 **-i patchfile** Read the patch information from the file named by the pathname *patchfile*, rather  
 109547 than the standard input.

109548 **-l** (The letter ell.) Cause any sequence of <blank> characters in the difference script to  
 109549 match any sequence of <blank> characters in the input file. Other characters shall  
 109550 be matched exactly.

- 109551        **-n**            Interpret the script as a normal difference.
- 109552        **-N**            Ignore patches where the differences have already been applied to the file; by  
109553            default, already-applied patches shall be rejected.
- 109554        **-o** *outfile*    Instead of modifying the files (specified by the *file* operand or the difference  
109555            listings) directly, write a copy of the file referenced by each patch, with the  
109556            appropriate differences applied, to *outfile*. Multiple patches for a single file shall be  
109557            applied to the intermediate versions of the file created by any previous patches,  
109558            and shall result in multiple, concatenated versions of the file being written to  
109559            *outfile*.
- 109560        **-p** *num*        For all pathnames in the patch file that indicate the names of files to be patched,  
109561            delete *num* pathname components from the beginning of each pathname. If the  
109562            pathname in the patch file is absolute, any leading <slash> characters shall be  
109563            considered the first component (that is, **-p 1** shall remove the leading <slash>  
109564            characters). Specifying **-p 0** shall cause the full pathname to be used. If **-p** is not  
109565            specified, only the basename (the final pathname component) shall be used.
- 109566        **-R**            Reverse the sense of the patch script; that is, assume that the difference script was  
109567            created from the new version to the old version. The **-R** option cannot be used  
109568            with *ed* scripts. The *patch* utility shall attempt to reverse each portion of the script  
109569            before applying it. Rejected differences shall be saved in swapped format. If this  
109570            option is not specified, and until a portion of the patch file is successfully applied,  
109571            *patch* attempts to apply each portion in its reversed sense as well as in its normal  
109572            sense. If the attempt is successful, the user shall be prompted to determine whether  
109573            the **-R** option should be set.
- 109574        **-r** *rejectfile*    Override the default reject filename. In the default case, the reject file shall have the  
109575            same name as the output file, with the suffix **.rej** appended to it; see [Patch](#)  
109576            [Application](#) (on page 3241).
- 109577        **-u**            Interpret the patch file as a unified context difference (the output of the *diff* utility  
109578            when the **-u** or **-U** options are specified).

**109579 OPERANDS**

109580        The following operand shall be supported:

109581        *file*            A pathname of a file to patch.

**109582 STDIN**

109583        See the INPUT FILES section.

**109584 INPUT FILES**

109585        Input files shall be text files.

**109586 ENVIRONMENT VARIABLES**

109587        The following environment variables shall affect the execution of *patch*:

109588        *LANG*            Provide a default value for the internationalization variables that are unset or null.  
109589            (See [XBD Section 8.2](#) (on page 169) the precedence of internationalization variables  
109590            used to determine the values of locale categories.)

109591        *LC\_ALL*          If set to a non-empty string value, override the values of all the other  
109592            internationalization variables.

109593        *LC\_COLLATE*      Determine the locale for the behavior of ranges, equivalence classes, and multi-  
109594            character collating elements used in the extended regular expression defined for  
109595

109596 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

109597 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 109598 characters (for example, single-byte as opposed to multi-byte characters in  
 109599 arguments and input files), and the behavior of character classes used in the  
 109600 extended regular expression defined for the **yesexpr** locale keyword in the  
 109601 *LC\_MESSAGES* category.

109602 *LC\_MESSAGES*  
 109603 Determine the locale used to process affirmative responses, and the locale used to  
 109604 affect the format and contents of diagnostic messages and prompts written to  
 109605 standard error.

109606 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

109607 *LC\_TIME* Determine the locale for recognizing the format of file timestamps written by the  
 109608 *diff* utility in a context-difference input file.

109609 **ASYNCHRONOUS EVENTS**

109610 Default.

109611 **STDOUT**

109612 Not used.

109613 **STDERR**

109614 The standard error shall be used for diagnostic and informational messages.

109615 **OUTPUT FILES**

109616 The output of the *patch* utility, the save files (**.orig** suffixes), and the reject files (**.rej** suffixes) shall  
 109617 be text files.

109618 **EXTENDED DESCRIPTION**

109619 A patch file may contain patching instructions for more than one file; filenames shall be  
 109620 determined as specified in [Filename Determination](#) (on page 3241). When the **-b** option is  
 109621 specified, for each patched file, the original shall be saved in a file of the same name with the  
 109622 suffix **.orig** appended to it.

109623 For each patched file, a reject file may also be created as noted in [Patch Application](#) (on page  
 109624 3241). In the absence of a **-r** option, the name of this file shall be formed by appending the suffix  
 109625 **.rej** to the original filename.

109626 **Patch File Format**

109627 The patch file shall contain zero or more lines of header information followed by one or more  
 109628 patches. Each patch shall contain zero or more lines of filename identification in the format  
 109629 produced by the **-c**, **-C**, **-u**, or **-U** options of the *diff* utility, and one or more sets of *diff* output,  
 109630 which are customarily called *hunks*.

109631 The *patch* utility shall recognize the following expression in the header information:

109632 **Index:** *pathname*  
 109633 The file to be patched is named *pathname*.

109634 If all lines (including headers) within a patch begin with the same leading sequence of <blank>  
 109635 characters, the *patch* utility shall remove this sequence before proceeding. Within each patch, if  
 109636 the type of difference is common context, the *patch* utility shall recognize the following  
 109637 expressions:



109638 \*\*\* *filename timestamp*  
 109639 The patches arose from *filename*.

109640 --- *filename timestamp*  
 109641 The patches should be applied to *filename*.

109642 If the type of difference is unified context, the *patch* utility shall recognize the following  
 109643 expressions:

109644 --- *filename timestamp*  
 109645 The patches arose from *filename*.

109646 + + + *filename timestamp*  
 109647 The patches should be applied to *filename*.

109648 Each hunk within a patch shall be the *diff* output to change a line range within the original file.  
 109649 The line numbers for successive hunks within a patch shall occur in ascending order.

### 109650 **Filename Determination**

109651 If no *file* operand is specified, *patch* shall perform the following steps to determine the filename  
 109652 to use:

109653 1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as  
 109654 specified by the **-p** option) from the filename on the line beginning with "\*\*\*\*" (if copied  
 109655 context) or "----" (if unified context), then test for the existence of this file relative to the  
 109656 current directory (or the directory specified with the **-d** option). If the file exists, the *patch*  
 109657 utility shall use this filename.

109658 2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as  
 109659 specified by the **-p** option) from the filename on the line beginning with "----" (if copied  
 109660 context) or "+ + +" (if unified context), then test for the existence of this file relative to the  
 109661 current directory (or the directory specified with the **-d** option). If the file exists, the *patch*  
 109662 utility shall use this filename.

109663 3. If the header information contains a line beginning with the string **Index:**, the *patch* utility  
 109664 shall delete pathname components (as specified by the **-p** option) from this line, then test  
 109665 for the existence of this file relative to the current directory (or the directory specified  
 109666 with the **-d** option). If the file exists, the *patch* utility shall use this filename.

109667 XSI 4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a `get -e`  
 109668 `SCCS/s.filename` command to retrieve an editable version of the file. If the file exists, the  
 109669 *patch* utility shall use this filename.

109670 5. The *patch* utility shall write a prompt to standard output and request a filename  
 109671 interactively from the controlling terminal (for example, **/dev/tty**).

### 109672 **Patch Application**

109673 If the **-c**, **-e**, **-n**, or **-u** option is present, the *patch* utility shall interpret information within each  
 109674 hunk as a copied context difference, an *ed* difference, a normal difference, or a unified context  
 109675 difference, respectively. In the absence of any of these options, the *patch* utility shall determine  
 109676 the type of difference based on the format of information within the hunk.

109677 For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line  
 109678 number at the beginning of the hunk, plus or minus any offset used in applying the previous  
 109679 hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and  
 109680 backwards at least 1 000 bytes for a set of lines that match the hunk context.

109681 If no such place is found and it is a context difference, then another scan shall take place,  
109682 ignoring the first and last line of context. If that fails, the first two and last two lines of context  
109683 shall be ignored and another scan shall be made. Implementations may search more extensively  
109684 for installation locations.

109685 If no location can be found, the *patch* utility shall append the hunk to the reject file. A rejected  
109686 hunk that is a copied context difference, an *ed* difference, or a normal difference shall be written  
109687 in copied-context-difference format regardless of the format of the patch file. It is  
109688 implementation-defined whether a rejected hunk that is a unified context difference is written in  
109689 copied-context-difference format or in unified-context-difference format. If the input was a  
109690 normal or *ed*-style difference, the reject file may contain differences with zero lines of context.  
109691 The line numbers on the hunks in the reject file may be different from the line numbers in the  
109692 patch file since they shall reflect the approximate locations for the failed hunks in the new file  
109693 rather than the old one.

109694 If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking  
109695 the *ed* utility.

#### 109696 EXIT STATUS

109697 The following exit values shall be returned:

109698 0 Successful completion.

109699 1 One or more lines were written to a reject file.

109700 >1 An error occurred.

#### 109701 CONSEQUENCES OF ERRORS

109702 Patches that cannot be correctly placed in the file shall be written to a reject file.

#### 109703 APPLICATION USAGE

109704 The **-R** option does not work with *ed* scripts because there is too little information to reconstruct  
109705 the reverse operation.

109706 The **-p** option makes it possible to customize a patch file to local user directory structures  
109707 without manually editing the patch file. For example, if the filename in the patch file was:

109708 `/curds/whey/src/blurfl/blurfl.c`

109709 Setting **-p 0** gives the entire pathname unmodified; **-p 1** gives:

109710 `curds/whey/src/blurfl/blurfl.c`

109711 without the leading `<slash>`, **-p 4** gives:

109712 `blurfl/blurfl.c`

109713 and not specifying **-p** at all gives:

109714 `blurfl.c .`

#### 109715 EXAMPLES

109716 None.

#### 109717 RATIONALE

109718 Some of the functionality in historical *patch* implementations was not specified. The following  
109719 documents those features present in historical implementations that have not been specified.

109720 A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options  
109721 and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

109722 In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility

109723 would search for the corresponding version information (the string specified in the header,  
109724 delimited by <blank> characters or the beginning or end of a line or the file) anywhere in the  
109725 original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to  
109726 standardize. For example, if:

109727 Prereq: 1.2

109728 were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the  
109729 prerequisite.

109730 The following options were dropped from historical implementations of *patch* as insufficiently  
109731 useful to standardize:

109732 **-b** The **-b** option historically provided a method for changing the name extension of  
109733 the backup file from the default **.orig**. This option has been modified and retained  
109734 in this volume of POSIX.1-2024.

109735 **-F** The **-F** option specified the number of lines of a context diff to ignore when  
109736 searching for a place to install a patch.

109737 **-f** The **-f** option historically caused *patch* not to request additional information from  
109738 the user.

109739 **-r** The **-r** option historically provided a method of overriding the extension of the  
109740 reject file from the default **.rej**.

109741 **-s** The **-s** option historically caused *patch* to work silently unless an error occurred.

109742 **-x** The **-x** option historically set internal debugging flags.

109743 In some file system implementations, the saving of a **.orig** file may produce unwanted results. In  
109744 the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum  
109745 filenames), the **.orig** file overwrites the new file. The reject file may also exceed this filename  
109746 limit. It was suggested, due to some historical practice, that a <tilde> ('~') suffix be used  
109747 instead of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is  
109748 not obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more  
109749 understandable.

109750 The **-b** option has the opposite sense in some historical implementations—do not save the **.orig**  
109751 file. The default case here is not to save the files, making *patch* behave more consistently with the  
109752 other standard utilities.

109753 The **-w** option in early proposals was changed to **-I** to match historical practice.

109754 The **-N** option was included because without it, a non-interactive application cannot reject  
109755 previously applied patches. For example, if a user is piping the output of *diff* into the *patch*  
109756 utility, and the user only wants to patch a file to a newer version non-interactively, the **-N** option  
109757 is required.

109758 Changes to the **-I** option description were proposed to allow matching across <newline>  
109759 characters in addition to just <blank> characters. Since this is not historical practice, and since  
109760 some ambiguities could result, it is suggested that future developments in this area utilize  
109761 another option letter, such as **-L**.

109762 The **-u** option of GNU *patch* has been added, along with support for unified context formats.

**109763 FUTURE DIRECTIONS**

109764 If this utility is directed to create a new directory entry that contains any bytes that have the  
109765 encoded value of a <newline> character, implementations are encouraged to treat this as an  
109766 error. A future version of this standard may require implementations to treat this as an error.

**109767 SEE ALSO**

109768 *diff, ed*

109769 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**109770 CHANGE HISTORY**

109771 First released in Issue 4.

**109772 Issue 5**

109773 The FUTURE DIRECTIONS section is added.

**109774 Issue 6**

109775 This utility is marked as part of the User Portability Utilities option.

109776 The description of the `-D` option and the steps in [Filename Determination](#) (on page 3241) are  
109777 changed to match historical practice as defined in the IEEE P1003.2b draft standard.

109778 The normative text is reworded to avoid use of the term “must” for application requirements.

109779 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/34 is applied, clarifying the way that the  
109780 *patch* utility performs `ifdef` selection for the `-D` option.

**109781 Issue 7**

109782 The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability  
109783 Utilities is now an option for interactive utilities.

109784 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

109785 SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` option.

109786 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
109787 `LC_MESSAGES` and `LC_CTYPE` environment variables and adding the `LC_COLLATE`  
109788 environment variable.

**109789 Issue 8**

109790 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
109791 filenames containing any bytes that have the encoded value of a <newline> character.

109792 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.

109793 **NAME**

109794 pathchk — check pathnames

109795 **SYNOPSIS**109796 pathchk [-p] [-P] *pathname*...109797 **DESCRIPTION**

109798 The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used  
 109799 to access or create a file without causing syntax errors) and portable (that is, no filename  
 109800 truncation results). More extensive portability checks are provided by the **-p** and **-P** options.

109801 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the  
 109802 underlying file system. A diagnostic shall be written for each *pathname* operand that:

- 109803 • Is longer than {PATH\_MAX} bytes (see **Pathname Variable Values** in XBD [<limits.h>](#))
- 109804 • Contains any component longer than {NAME\_MAX} bytes in its containing directory
- 109805 • Contains any component in a directory that is not searchable
- 109806 • Contains any byte sequence that is not valid in its containing directory

109807 The format of the diagnostic message is not specified, but shall indicate the error detected and  
 109808 the corresponding *pathname* operand.

109809 It shall not be considered an error if one or more components of a *pathname* operand do not exist  
 109810 as long as a file matching the pathname specified by the missing components could be created  
 109811 that does not violate any of the checks specified above.

109812 **OPTIONS**109813 The *pathchk* utility shall conform to XBD [Section 12.2](#) (on page 215).

109814 The following option shall be supported:

109815 **-p** Instead of performing checks based on the underlying file system, write a  
 109816 diagnostic for each *pathname* operand that:

- 109817 • Is longer than {\_POSIX\_PATH\_MAX} bytes (see **Minimum Values** in XBD  
 109818 [<limits.h>](#))
- 109819 • Contains any component longer than {\_POSIX\_NAME\_MAX} bytes
- 109820 • Contains any character in any component that is not in the portable filename  
 109821 character set

109822 **-P** Write a diagnostic for each *pathname* operand that:

- 109823 • Contains a component whose first character is the <hyphen-minus>  
 109824 character
- 109825 • Is empty

109826 **OPERANDS**

109827 The following operand shall be supported:

109828 *pathname* A pathname to be checked.109829 **STDIN**

109830 Not used.

109831 **INPUT FILES**

109832 None.

109833 **ENVIRONMENT VARIABLES**109834 The following environment variables shall affect the execution of *pathchk*:

109835 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 109836 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables  
 109837 used to determine the values of locale categories.)

109838 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 109839 internationalization variables.

109840 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 109841 characters (for example, single-byte as opposed to multi-byte characters in  
 109842 arguments).

109843 *LC\_MESSAGES*

109844 Determine the locale that should be used to affect the format and contents of  
 109845 diagnostic messages written to standard error.

109846 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

109847 **ASYNCHRONOUS EVENTS**

109848 Default.

109849 **STDOUT**

109850 Not used.

109851 **STDERR**

109852 The standard error shall be used only for diagnostic messages.

109853 **OUTPUT FILES**

109854 None.

109855 **EXTENDED DESCRIPTION**

109856 None.

109857 **EXIT STATUS**

109858 The following exit values shall be returned:

109859 0 All *pathname* operands passed all of the checks.

109860 &gt;0 An error occurred.

109861 **CONSEQUENCES OF ERRORS**

109862 Default.

109863 **APPLICATION USAGE**

109864 The *test* utility can be used to determine whether a given pathname names an existing file; it  
 109865 does not, however, give any indication of whether or not any component of the pathname was  
 109866 truncated in a directory where the `_POSIX_NO_TRUNC` feature is not in effect. The *pathchk*  
 109867 utility does not check for file existence; it performs checks to determine whether a pathname  
 109868 does exist or could be created with no pathname component truncation.

109869 The *noclobber* option in the shell (see the *set* special built-in) can be used to atomically create a  
 109870 file. As with all file creation semantics in the System Interfaces volume of POSIX.1-2024, it  
 109871 guarantees atomic creation, but still depends on applications to agree on conventions and  
 109872 cooperate on the use of files after they have been created.

109873 To verify that a pathname meets the requirements of filename portability, applications should

109874 use both the `-p` and `-P` options together.

#### 109875 EXAMPLES

109876 To verify that all pathnames in an imported data interchange archive are legitimate and  
109877 unambiguous on the current system:

```
109878 # This example assumes that no pathnames in the archive
109879 # contain <newline> characters.
109880 pax -f archive | sed -e 's/^[^:alnum:]]/\\&/g' | xargs pathchk --
109881 if [ $? -eq 0 ]
109882 then
109883     pax -r -f archive
109884 else
109885     echo Investigate problems before importing files.
109886     exit 1
109887 fi
```

109888 To verify that all files in the current directory hierarchy could be moved to any system  
109889 conforming to the System Interfaces volume of POSIX.1-2024 that also supports the *pax* utility:

```
109890 find . -exec pathchk -p -P {} +
109891 if [ $? -eq 0 ]
109892 then
109893     pax -w -f ../archive .
109894 else
109895     echo Portable archive cannot be created.
109896     exit 1
109897 fi
```

109898 To verify that a user-supplied pathname names a readable file and that the application can create  
109899 a file extending the given path without truncation and without overwriting any existing file:

```
109900 case $- in
109901     *C*)   reset="";;
109902     *)    reset="set +C"
109903           set -C;;
109904 esac
109905 test -r "$path" && pathchk "$path.out" &&
109906     rm "$path.out" > "$path.out"
109907 if [ $? -ne 0 ]; then
109908     printf "%s: %s not found or %s.out fails \
109909 creation checks.\n" $0 "$path" "$path"
109910     $reset      # Reset the noclobber option in case a trap
109911                # on EXIT depends on it.
109912     exit 1
109913 fi
109914 $reset
109915 PROCESSING < "$path" > "$path.out"
```

109916 The following assumptions are made in this example:

- 109917 1. **PROCESSING** represents the code that is used by the application to use `$path` once it is  
109918 verified that `$path.out` works as intended.

- 109919 2. The state of the *noclobber* option is unknown when this code is invoked and should be set  
 109920 on exit to the state it was in when this code was invoked. (The **reset** variable is used in  
 109921 this example to restore the initial state.)
- 109922 3. Note the usage of:
- 109923 `rm "$path.out" > "$path.out"`
- 109924 a. The *pathchk* command has already verified, at this point, that **\$path.out** is not  
 109925 truncated.
- 109926 b. With the *noclobber* option set, the shell verifies that **\$path.out** does not already exist  
 109927 before invoking *rm*.
- 109928 c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application  
 109929 can create the file again in the **PROCESSING** step.
- 109930 d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:  
 109931 `rm "$path.out" > "$path.out"`  
 109932 should be replaced with:  
 109933 `> "$path.out"`  
 109934 which verifies that the file did not already exist, but leaves **\$path.out** in place for  
 109935 use by **PROCESSING**.

#### 109936 RATIONALE

109937 The *pathchk* utility was new for the ISO POSIX-2:1993 standard. It, along with the *set*  
 109938 `-C(noclobber)` option added to the shell, replaces the *mktemp*, *validfnam*, and *create* utilities that  
 109939 appeared in early proposals. All of these utilities were attempts to solve several common  
 109940 problems:

- 109941 • Verify the validity (for several different definitions of “valid”) of a pathname supplied by a  
 109942 user, generated by an application, or imported from an external source.
- 109943 • Atomically create a file.
- 109944 • Perform various string handling functions to generate a temporary filename.

109945 The *create* utility, included in an early proposal, provided checking and atomic creation in a  
 109946 single invocation of the utility; these are orthogonal issues and need not be grouped into a single  
 109947 utility. Note that the *noclobber* option also provides a way of creating a lock for process  
 109948 synchronization; since it provides an atomic *create*, there is no race between a test for existence  
 109949 and the following creation if it did not exist.

109950 Having a function like *tmpnam()* in the ISO C standard is important in many high-level  
 109951 languages. The shell programming language, however, has built-in string manipulation  
 109952 facilities, making it very easy to construct temporary filenames. The names needed obviously  
 109953 depend on the application, but are frequently of a form similar to:

109954 `$TMPDIR/application_abbreviation$$ .suffix`

109955 In cases where there is likely to be contention for a given suffix, a simple shell **for** or **while** loop  
 109956 can be used with the shell *noclobber* option to create a file without risk of collisions, as long as  
 109957 applications trying to use the same filename name space are cooperating on the use of files after  
 109958 they have been created.

109959 For historical purposes, `-p` does not check for the use of the <hyphen-minus> character as the  
 109960 first character in a component of the pathname, or for an empty *pathname* operand.



109961 **FUTURE DIRECTIONS**

109962 None.

109963 **SEE ALSO**109964 [Section 2.7](#) (on page 2493), [set](#) (on page 2553), [test](#)109965 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215), [<limits.h>](#)109966 **CHANGE HISTORY**

109967 First released in Issue 4.

109968 **Issue 7**

109969 Austin Group Interpretations 1003.1-2001 #039, #040, and #094 are applied.

109970 SD5-XCU-ERN-121 is applied, updating the EXAMPLES section.

109971 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0127 [291] is applied.

109972 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0150 [584] and XCU/TC2-2008/0151

109973 [584] are applied.

109974 **Issue 8**109975 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

109976 **NAME**

109977 pax — portable archive interchange

109978 **SYNOPSIS**109979 pax [-dv] [-c|-n] [-H|-L] [-o options] [-f archive] [-s replstr]...  
109980 [pattern...]109981 pax -r[-c|-n] [-dikuv] [-H|-L] [-f archive] [-o options]... [-p string]...  
109982 [-s replstr]... [pattern...]109983 pax -w [-dituvX] [-H|-L] [-b blocksize] [[-a] [-f archive]] [-o options]...  
109984 [-s replstr]... [-x format] [file...]109985 pax -r -w [-dikltuvX] [-H|-L] [-o options]... [-p string]...  
109986 [-s replstr]... [file...] directory109987 **DESCRIPTION**109988 The *pax* utility shall read, write, and write lists of the members of archive files and copy  
109989 directory hierarchies. A variety of archive formats shall be supported; see the *-x format* option.109990 The action to be taken depends on the presence of the *-r* and *-w* options. The four combinations  
109991 of *-r* and *-w* are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,  
109992 corresponding respectively to the four forms shown in the SYNOPSIS section.109993 **list** In **list** mode (when neither *-r* nor *-w* are specified), *pax* shall write the names of  
109994 the members of the archive file read from the standard input, with pathnames  
109995 matching the specified patterns, to standard output. If a named file is of type  
109996 directory, the file hierarchy rooted at that file shall be listed as well.109997 **read** In **read** mode (when *-r* is specified, but *-w* is not), *pax* shall extract the members of  
109998 the archive file read from the standard input, with pathnames matching the  
109999 specified patterns. If an extracted file is of type directory, the file hierarchy rooted  
110000 at that file shall be extracted as well. The extracted files shall be created performing  
110001 pathname resolution with the directory in which *pax* was invoked as the current  
110002 working directory.110003 If an attempt is made to extract a directory when the directory already exists, this  
110004 shall not be considered an error. If an attempt is made to extract a FIFO when the  
110005 FIFO already exists, this shall not be considered an error.110006 The ownership, access, and modification times, and file mode of the restored files  
110007 are discussed under the *-p* option.110008 **write** In **write** mode (when *-w* is specified, but *-r* is not), *pax* shall write the contents of  
110009 the *file* operands to the standard output in an archive format. If no *file* operands are  
110010 specified, a list of files to copy, one per line, shall be read from the standard input  
110011 and each entry in this list shall be processed as if it had been a *file* operand on the  
110012 command line. A file of type directory shall include all of the files in the file  
110013 hierarchy rooted at the file.110014 **copy** In **copy** mode (when both *-r* and *-w* are specified), *pax* shall copy the *file* operands  
110015 to the destination directory.110016 If no *file* operands are specified, a list of files to copy, one per line, shall be read  
110017 from the standard input. A file of type directory shall include all of the files in the  
110018 file hierarchy rooted at the file.110019 The effect of the **copy** shall be as if the copied files were written to a *pax* format  
110020 archive file and then subsequently extracted, except that copying of sockets may be

110021 supported even if archiving them in write mode is not supported, and that there  
 110022 may be hard links between the original and the copied files. If the destination  
 110023 directory is a subdirectory of one of the files to be copied, the results are  
 110024 unspecified. If the destination directory is a file of a type not defined by the System  
 110025 Interfaces volume of POSIX.1-2024, the results are implementation-defined;  
 110026 otherwise, it shall be an error for the file named by the *directory* operand not to  
 110027 exist, not be writable by the user, or not be a file of type directory.

110028 In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member,  
 110029 *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces  
 110030 volume of POSIX.1-2024, called with the following arguments:

- 110031 • The intermediate directory used as the *path* argument
- 110032 • The value of the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO as the *mode*  
 110033 argument

110034 If any specified *pattern* or *file* operands are not matched by at least one file or archive member,  
 110035 *pax* shall write a diagnostic message to standard error for each one that did not match and exit  
 110036 with a non-zero exit status.

110037 The archive formats described in the EXTENDED DESCRIPTION section shall be automatically  
 110038 detected on input. The default output archive format shall be implementation-defined.

110039 A single archive can span multiple files. The *pax* utility shall determine, in an implementation-  
 110040 defined manner, what file to read or write as the next file.

110041 If the selected archive format supports the specification of linked files, it shall be an error if these  
 110042 files cannot be linked when the archive is extracted. For archive formats that do not store file  
 110043 contents with each name that causes a hard link, if the file that contains the data is not extracted  
 110044 during this *pax* session, either the data shall be restored from the original file, or a diagnostic  
 110045 message shall be displayed with the name of a file that can be used to extract the data. In  
 110046 traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited  
 110047 directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall  
 110048 write a diagnostic message to standard error and shall terminate.

## 110049 OPTIONS

110050 The *pax* utility shall conform to XBD Section 12.2 (on page 215), except that the order of  
 110051 presentation of the **-o**, **-p**, and **-s** options is significant.

110052 The following options shall be supported:

- 110053 **-r** Read an archive file from standard input.
- 110054 **-w** Write files to the standard output in the specified archive format.
- 110055 **-a** Append files to the end of the archive. It is implementation-defined which devices  
 110056 on the system support appending. Additional file formats unspecified by this  
 110057 volume of POSIX.1-2024 may impose restrictions on appending.
- 110058 **-b** *blocksize* Block the output at a positive decimal integer number of bytes per write to the  
 110059 archive file. Devices and archive formats may impose restrictions on blocking.  
 110060 Blocking shall be automatically determined on input. Conforming applications  
 110061 shall not specify a *blocksize* value larger than 32256. Default blocking when  
 110062 creating archives depends on the archive format. (See the **-x** option below.)
- 110063 **-c** Match all file or archive members except those specified by the *pattern* or *file*  
 110064 operands.

|        |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 110065 | -d                | Cause files of type directory being copied or archived or archive members of type directory being extracted or listed to match only the file or archive member itself and not the file hierarchy rooted at the file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 110066 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110067 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110068 | -f <i>archive</i> | Specify the pathname of the input or output archive, overriding the default standard input (in <b>list</b> or <b>read</b> modes) or standard output ( <b>write</b> mode).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 110069 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110070 | -H                | If a symbolic link referencing a file of type directory is specified on the command line, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line, then <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.                                                                                                                                                                                                                                     |
| 110071 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110072 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110073 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110074 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110075 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110076 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110077 | -i                | Interactively rename files or archive members. For each archive member matching a <i>pattern</i> operand or file matching a <i>file</i> operand, a prompt shall be written to the file <b>/dev/tty</b> . The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from <b>/dev/tty</b> . If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The <i>pax</i> utility shall immediately exit with a non-zero exit status if end-of-file is encountered when reading a response or if <b>/dev/tty</b> cannot be opened for reading and writing. |
| 110078 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110079 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110080 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110081 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110082 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110083 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110084 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110085 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110086 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110087 |                   | The results of extracting a hard link to a file that has been renamed during extraction are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 110088 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110089 | -k                | Prevent the overwriting of existing files.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 110090 | -l                | (The letter ell.) In <b>copy</b> mode, hard links shall be made between the source and destination file hierarchies whenever possible. If specified in conjunction with -H or -L, when a symbolic link is encountered, the hard link created in the destination file hierarchy shall be to the file referenced by the symbolic link. If specified when neither -H nor -L is specified, when a symbolic link is encountered, the implementation shall create a hard link to the symbolic link in the source file hierarchy or copy the symbolic link to the destination.                                                                                                                                                                                                                                                                |
| 110091 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110092 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110093 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110094 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110095 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110096 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110097 | -L                | If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.                                                                                                                          |
| 110098 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110099 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110100 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110101 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110102 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110103 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110104 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110105 | -n                | Select the first archive member that matches each <i>pattern</i> operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 110106 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110107 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110108 | -o <i>options</i> | Provide information to the implementation to modify the algorithm for extracting or writing files. The value of <i>options</i> shall consist of one or more <comma>-separated keywords of the form:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 110109 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110110 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 110111 |                   | <i>keyword</i> [[:]= <i>value</i> ] [, <i>keyword</i> [[:]= <i>value</i> ], ...]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

110112 Some keywords apply only to certain file formats, as indicated with each  
 110113 description. Use of keywords that are inapplicable to the file format being  
 110114 processed produces undefined results.

110115 Keywords in the *options* argument shall be a string that would be a valid portable  
 110116 filename as described in XBD [Section 3.265](#) (on page 70).

110117 **Note:** Keywords are not expected to be filenames, merely to follow the same character  
 110118 composition rules as portable filenames.

110119 Keywords can be preceded with white space. The *value* field shall consist of zero or  
 110120 more characters; within *value*, the application shall precede any literal <comma>  
 110121 with a <backslash>, which shall be ignored, but preserves the <comma> as part of  
 110122 *value*. A <comma> as the final character, or a <comma> followed solely by white  
 110123 space as the final characters, in *options* shall be ignored. Multiple **-o** options can be  
 110124 specified; if keywords given to these multiple **-o** options conflict, the keywords  
 110125 and values appearing later in command line sequence shall take precedence and  
 110126 the earlier shall be silently ignored. The following keyword values of *options* shall  
 110127 be supported for the file formats as indicated:

110128 **delete=pattern**

110129 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*  
 110130 shall omit from extended header records that it produces any keywords  
 110131 matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore  
 110132 any keywords matching the string pattern in the extended header records. In  
 110133 both cases, matching shall be performed using the pattern matching notation  
 110134 described in [Section 2.14.1](#) (on page 2523) and [Section 2.14.2](#) (on page 2524).  
 110135 For example:

110136 **-o delete=security.\***

110137 would suppress security-related information. See [pax Extended Header](#) (on  
 110138 page 3264) for extended header record keyword usage.

110139 When multiple **-odelete=pattern** options are specified, the patterns shall be  
 110140 additive; all keywords matching the specified string patterns shall be omitted  
 110141 from extended header records that *pax* produces.

110142 **exthdr.name=string**

110143 (Applicable only to the **-x pax** format.) This keyword allows user control over  
 110144 the name that is written into the **ustar** header blocks for the extended header  
 110145 produced under the circumstances described in [pax Header Block](#) (on page  
 110146 3263). The name shall be the contents of *string*, after the following character  
 110147 substitutions have been made:

| <i>string</i><br>Includes: | Replaced by:                                                                                                       |
|----------------------------|--------------------------------------------------------------------------------------------------------------------|
| %d                         | The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated pathname. |
| %f                         | The filename of the file, equivalent to the result of the <i>basename</i> utility on the translated pathname.      |
| %p                         | The process ID of the <i>pax</i> process.                                                                          |
| %%                         | A '%' character.                                                                                                   |

110156 Any other '%' characters in *string* produce undefined results.

110157 If no **-o exthdr.name=string** is specified, *pax* shall use the following default

110158 value:  
 110159 %d/PaxHeaders.%p/%f

110160 **globexthdr.name=string**

(Applicable only to the **-x pax** format.) When used in **write** or **copy** mode with the appropriate options, *pax* shall create global extended header records with **ustar** header blocks that are treated as regular files by previous versions of *pax*. This keyword allows user control over the name that is written into the **ustar** header blocks for global extended header records. The name shall be the contents of *string*, after the following character substitutions have been made:

| <i>string</i><br>Includes: | Replaced by:                                                                                                       |
|----------------------------|--------------------------------------------------------------------------------------------------------------------|
| %n                         | An integer that represents the sequence number of the global extended header record in the archive, starting at 1. |
| %p                         | The process ID of the <i>pax</i> process.                                                                          |
| %%                         | A '%' character.                                                                                                   |

110167  
 110168  
 110169  
 110170  
 110171  
 110172

110173 Any other '%' characters in *string* produce undefined results.

110174 If no **-o globexthdr.name=string** is specified, *pax* shall use the following  
 110175 default value:

110176 \$TMPDIR/GlobalHead.%p.%n

110177 where *\$TMPDIR* represents the value of the *TMPDIR* environment variable. If  
 110178 *TMPDIR* is not set, *pax* shall use **/tmp**.

110179 **invalid=action**

(Applicable only to the **-x pax** format.) This keyword allows user control over the action *pax* takes upon encountering values in an extended header record that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list** mode, cannot be written in the codeset and current locale of the implementation. The following are invalid values that shall be recognized by *pax*:

110186  
 110187  
 110188  
 110189  
 110190  
 110191  
 110192  
 110193  
 110194

- In **read** or **copy** mode, a filename or link name that contains character encodings invalid in the destination hierarchy. (For example, the name may contain embedded NULs.)
- In **read** or **copy** mode, a filename or link name that is longer than the maximum allowed in the destination hierarchy (for either a pathname component or the entire pathname).
- In **list** mode, any character string value (filename, link name, user name, and so on) that cannot be written in the codeset and current locale of the implementation.

110195 The following mutually-exclusive values of the *action* argument are supported:

110196 **binary** In **write** mode, *pax* shall generate a **hdrcharset=BINARY**  
 110197 extended header record for each file with a filename, link name,  
 110198 group name, owner name, or any other field in an extended  
 110199 header record that cannot be translated to the UTF-8 codeset,  
 110200 allowing the archive to contain the files with unencoded  
 110201 extended header record values. In **read** or **copy** mode, *pax* shall  
 110202 use the values specified in the header without translation,

110203 regardless of whether this may overwrite an existing file with a  
 110204 valid name. In **list** mode, *pax* shall behave identically to the  
 110205 **bypass** action.

110206 **bypass** In **read** or **copy** mode, *pax* shall bypass the file, causing no  
 110207 change to the destination hierarchy. In **list** mode, *pax* shall write  
 110208 all requested valid values for the file, but its method for writing  
 110209 invalid values is unspecified.

110210 **rename** In **read** or **copy** mode, *pax* shall act as if the **-i** option were in  
 110211 effect for each file with invalid filename or link name values,  
 110212 allowing the user to provide a replacement name interactively.  
 110213 In **list** mode, *pax* shall behave identically to the **bypass** action.

110214 **UTF-8** When used in **read**, **copy**, or **list** mode and a filename, link  
 110215 name, owner name, or any other field in an extended header  
 110216 record cannot be translated from the **pax** UTF-8 codeset format  
 110217 to the codeset and current locale of the implementation, *pax* shall  
 110218 use the actual UTF-8 encoding for the name. If a **hdrcharset**  
 110219 extended header record is in effect for this file, the character set  
 110220 specified by that record shall be used instead of UTF-8. If a  
 110221 **hdrcharset=BINARY** extended header record is in effect for this  
 110222 file, no translation shall be performed.

110223 **write** In **read** or **copy** mode, *pax* shall write the file, translating the  
 110224 name, regardless of whether this may overwrite an existing file  
 110225 with a valid name. In **list** mode, *pax* shall behave identically to  
 110226 the **bypass** action.

110227 If no **-o invalid=option** is specified, *pax* shall act as if **-o invalid=bypass** were  
 110228 specified. Any overwriting of existing files that may be allowed by the  
 110229 **-o invalid=** actions shall be subject to permission (**-p**) and modification time  
 110230 (**-u**) restrictions, and shall be suppressed if the **-k** option is also specified.

110231 **linkdata**  
 110232 (Applicable only to the **-x pax** format.) In **write** mode, *pax* shall write the  
 110233 contents of a file to the archive even when that file is merely a hard link to a  
 110234 file whose contents have already been written to the archive.

110235 **listopt=format**  
 110236 This keyword specifies the output format of the table of contents produced  
 110237 when the **-v** option is specified in **list** mode. See [List Mode Format Specifications](#)  
 110238 (on page 3258). To avoid ambiguity, the **listopt=format** shall be  
 110239 the only or final **keyword=value** pair in a **-o** option-argument; all characters  
 110240 in the remainder of the option-argument shall be considered part of the format  
 110241 string. When multiple **-olistopt=format** options are specified, the format  
 110242 strings shall be considered a single, concatenated string, evaluated in  
 110243 command line order.

110244 **times**  
 110245 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*  
 110246 shall include **atime** and **mtime** extended header records for each file. See [pax Extended Header File Times](#)  
 110247 (on page 3267).

110248 In addition to these keywords, if the **-x pax** format is specified, any of the  
 110249 keywords and values defined in [pax Extended Header](#) (on page 3264), including

110250 implementation extensions, can be used in **-o** option-arguments, in either of two  
110251 modes:

110252 **keyword=***value*

110253 When used in **write** or **copy** mode, these keyword/value pairs shall be  
110254 included at the beginning of the archive as **typeflag g** global extended header  
110255 records. When used in **read** or **list** mode, these keyword/value pairs shall act  
110256 as if they had been at the beginning of the archive as **typeflag g** global  
110257 extended header records.

110258 **keyword:=***value*

110259 When used in **write** or **copy** mode, these keyword/value pairs shall be  
110260 included as records at the beginning of a **typeflag x** extended header for each  
110261 file. (This shall be equivalent to the `<equals-sign>` form except that it creates  
110262 no **typeflag g** global extended header records.) When used in **read** or **list**  
110263 mode, these keyword/value pairs shall act as if they were included as records  
110264 at the end of each extended header; thus, they shall override any global or file-  
110265 specific extended header record keywords of the same names. For example, in  
110266 the command:

```
110267 pax -r -o "  
110268 gname:=mygroup,  
110269 " <archive
```

110270 the group name is forced to a new value for all files read from the archive.

110271 The precedence of **-o** keywords over various fields in the archive is described in  
110272 [pax Extended Header Keyword Precedence](#) (on page 3267). If the **-o**  
110273 **delete=***pattern*, **-o keyword=***value*, or **-o keyword:=***value* options are used to  
110274 override or remove any extended header data needed to find files in an archive  
110275 (e.g., **-o delete=size** for a file whose size cannot be represented in a **ustar**  
110276 header or **-o size=100** for a file whose size is not 100 bytes), the behavior is  
110277 undefined.

110278 **-p** *string*

110279 Specify one or more file characteristic options (privileges). The *string* option-  
110280 argument shall be a string specifying file characteristics to be retained or discarded  
110281 on extraction. The string shall consist of the specification characters *a*, *e*, *m*, *o*, and  
110282 *p*. Other implementation-defined characters can be included. Multiple  
110283 characteristics can be concatenated within the same string and multiple **-p** options  
can be specified. The meaning of the specification characters are as follows:

110284 *a* Do not preserve file access times.

110285 *e* Preserve the user ID, group ID, file mode bits (see XBD [Section 3.145](#), on page  
110286 52), access time, modification time, and any other implementation-defined file  
110287 characteristics.

110288 *m* Do not preserve file modification times.

110289 *o* Preserve the user ID and group ID.

110290 *p* Preserve the file mode bits. Other implementation-defined file mode attributes  
110291 may be preserved.

110292 In the preceding list, “preserve” indicates that an attribute stored in the archive  
110293 shall be given to the extracted file, subject to the permissions of the invoking  
110294 process. The access and modification times of the file shall be preserved unless  
110295 otherwise specified with the **-p** option or not stored in the archive. All attributes



110296 that are not preserved shall be determined as part of the normal file creation action  
 110297 (see [Section 1.1.1.4](#), on page 2454).

110298 If neither the `e` nor the `o` specification character is specified, or the user ID and  
 110299 group ID are not preserved for any reason, *pax* shall not set the `S_ISUID` and  
 110300 `S_ISGID` bits of the file mode.

110301 If the preservation of any of these items fails for any reason, *pax* shall write a  
 110302 diagnostic message to standard error. Failure to preserve these items shall affect  
 110303 the final exit status, but shall not cause the extracted file to be deleted.

110304 If file characteristic letters in any of the *string* option-arguments are duplicated or  
 110305 conflict with each other, the ones given last shall take precedence. For example, if  
 110306 `-p eme` is specified, file modification times are preserved.

110307 `-s replstr` Modify file or archive member names named by *pattern* or *file* operands according  
 110308 to the substitution expression *replstr*, using the syntax of the *ed* utility. The concepts  
 110309 of “address” and “line” are meaningless in the context of the *pax* utility, and shall  
 110310 not be supplied. The format shall be:

110311 `-s /old/new/[gpsS]`

110312 where as in *ed*, *old* is a basic regular expression and *new* can contain an  
 110313 `<ampersand>`, `'\n'` (where *n* is a digit) back-references, or subexpression  
 110314 matching. The *old* string shall also be permitted to contain `<newline>` characters.

110315 Any non-null character can be used as a delimiter (`'/'` shown here). Multiple `-s`  
 110316 expressions can be specified; the expressions shall be applied in the order  
 110317 specified, terminating with the first successful substitution. The optional trailing  
 110318 `'g'` is as defined in the *ed* utility. The optional trailing `'p'` shall cause successful  
 110319 substitutions to be written to standard error. The optional trailing `'s'` and `'S'`  
 110320 control whether the substitutions are applied to symbolic link contents: `'s'` shall  
 110321 cause them not to be applied; `'S'` shall cause them to be applied. If neither is  
 110322 present, it is unspecified which is the default. If both are present, the behavior is  
 110323 unspecified. File or archive member names that substitute to the empty string shall  
 110324 be ignored when reading and writing archives. Symbolic link contents that  
 110325 substitute to the empty string shall not be treated specially.

110326 `-t` When reading files from the file system, and if the user has the permissions  
 110327 required by *futimens()* to do so, set the access time of each file read to the access  
 110328 time that it had before being read by *pax*.

110329 `-u` Ignore files that are older (having a less recent file modification time) than a pre-  
 110330 existing file or archive member with the same name. In **read** mode, an archive  
 110331 member with the same name as a file in the file system shall be extracted if the  
 110332 archive member is newer than the file. In **write** mode, an archive file member with  
 110333 the same name as a file in the file system shall be superseded if the file is newer  
 110334 than the archive member. If `-a` is also specified, this is accomplished by appending  
 110335 to the archive; otherwise, it is unspecified whether this is accomplished by actual  
 110336 replacement in the archive or by appending to the archive. In **copy** mode, the file  
 110337 in the destination hierarchy shall be replaced if the file in the source hierarchy is  
 110338 newer.

110339 `-v` In **list** mode, produce a verbose table of contents (see the **STDOUT** section).  
 110340 Otherwise, write archive member pathnames to standard error (see the **STDERR**  
 110341 section).

- 110342        **-x format**     Specify the output archive format. The *pax* utility shall support the following  
110343                   formats:
- 110344                **cpio**           The **cpio** interchange format; see the EXTENDED DESCRIPTION  
110345                   section. The default *blocksize* for this format for character special  
110346                   archive files shall be 5120. Implementations shall support all  
110347                   *blocksize* values less than or equal to 32256 that are multiples of 512.
- 110348                **pax**            The **pax** interchange format; see the EXTENDED DESCRIPTION  
110349                   section. The default *blocksize* for this format for character special  
110350                   archive files shall be 5120. Implementations shall support all  
110351                   *blocksize* values less than or equal to 32256 that are multiples of 512.
- 110352                **ustar**        The **tar** interchange format; see the EXTENDED DESCRIPTION  
110353                   section. The default *blocksize* for this format for character special  
110354                   archive files shall be 10240. Implementations shall support all  
110355                   *blocksize* values less than or equal to 32256 that are multiples of 512.
- 110356                   Implementation-defined formats shall specify a default block size as well as any  
110357                   other block sizes supported for character special archive files.
- 110358                   Any attempt to append to an archive file in a format different from the existing  
110359                   archive format shall cause *pax* to exit immediately with a non-zero exit status.
- 110360        **-X**            When traversing the file hierarchy specified by a pathname, *pax* shall not descend  
110361                   below directories that have a different device ID (*st\_dev*; see XSH *fstatat()*) than the  
110362                   specified pathname; that is, when a directory with a different device ID is  
110363                   encountered, *pax* shall process (archive or copy) the directory itself but shall not  
110364                   process any files below the directory.
- 110365                   Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered  
110366                   an error and the last option specified shall determine the behavior of the utility.
- 110367                   The options that operate on the names of files or archive members (**-c**, **-i**, **-n**, **-s**, **-u**, and **-v**)  
110368                   shall interact as follows. In **read** mode, the archive members shall be selected based on the user-  
110369                   specified *pattern* operands as modified by the **-c**, **-n**, and **-u** options. Then, any **-s** and **-i**  
110370                   options shall modify, in that order, the names of the selected files. The **-v** option shall write  
110371                   names resulting from these modifications.
- 110372                   In **write** mode, the files shall be selected based on the user-specified pathnames as modified by  
110373                   the **-u** option. Then, any **-s** and **-i** options shall modify, in that order, the names of these  
110374                   selected files. The **-v** option shall write names resulting from these modifications.
- 110375                   If both the **-u** and **-n** options are specified, *pax* shall not consider a file selected unless it is  
110376                   newer than the file to which it is compared.
- 110377        **List Mode Format Specifications**
- 110378                   In **list** mode with the **-o listopt=format** option, the *format* argument shall be applied for each  
110379                   selected file. The *pax* utility shall append a <newline> to the **listopt** output for each selected file.  
110380                   The *format* argument shall be used as the *format* string described in XBD Chapter 5 (on page 113),  
110381                   with the exceptions 1. through 6. defined in the EXTENDED DESCRIPTION section of *printf*,  
110382                   plus the following exceptions:
- 110383                   7. The sequence (*keyword*) can occur before a format conversion specifier. The conversion  
110384                   argument is defined by the value of *keyword*. The implementation shall support the  
110385                   following keywords:

- 110386 — Any of the Field Name entries in [Table 3-15](#) (on page 3268) and [Table 3-17](#) (on page 3272). The implementation may support the *cpio* keywords without the leading *c\_* in addition to the form required by [Table 3-17](#) (on page 3272).
- 110387
- 110388
- 110389 — Any keyword defined for the extended header in [pax Extended Header](#) (on page 3264).
- 110390
- 110391 — Any keyword provided as an implementation-defined extension within the extended header defined in [pax Extended Header](#) (on page 3264).
- 110392

110393 For example, the sequence "%(charset) s" is the string value of the name of the character set in the extended header.

110394

110395 The result of the keyword conversion argument shall be the value from the applicable header field or extended header, without any trailing NULs.

110396

110397 All keyword values used as conversion arguments shall be translated from the UTF-8 encoding (or alternative encoding specified by any **hdrcharset** extended header record) to the character set appropriate for the local file system, user database, and so on, as applicable.

110398

110399

110400

- 110401 8. An additional conversion specifier character, **T**, shall be used to specify time formats. The **T** conversion specifier character can be preceded by the sequence (*keyword=subformat*), where *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime** and the default subformat shall be:
- 110402
- 110403
- 110404

110405 %b %e %H:%M %Y

- 110406 9. An additional conversion specifier character, **M**, shall be used to specify the file mode string as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used. For example, %.1M writes the single character corresponding to the *<entry type>* field of the *ls -l* command.
- 110407
- 110408
- 110409

- 110410 10. An additional conversion specifier character, **D**, shall be used to specify the device for block or special files, if applicable, in an implementation-defined format. If not applicable, and (*keyword*) is specified, then this conversion shall be equivalent to %(*keyword*)u. If not applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to *<space>*.
- 110411
- 110412
- 110413

- 110414 11. An additional conversion specifier character, **F**, shall be used to specify a pathname. The **F** conversion character can be preceded by a sequence of *<comma>*-separated keywords:
- 110415

110416 (*keyword*[,*keyword*] ... )

110417 The values for all the keywords that are non-null shall be concatenated together, each separated by a '/'. The default shall be (**path**) if the keyword **path** is defined; otherwise, the default shall be (**prefix,name**).

110418

110419

- 110420 12. An additional conversion specifier character, **L**, shall be used to specify a symbolic link expansion. If the current file is a symbolic link, then %L shall expand to:
- 110421

110422 "%s -> %s", *<value of keyword>*, *<contents of link>*

110423 Otherwise, the %L conversion specification shall be the equivalent of %F.

## 110424 OPERANDS

110425 The following operands shall be supported:

110426 *directory* The destination directory pathname for **copy** mode.

- 110427 *file* A pathname of a file to be copied or archived.
- 110428 *pattern* A pattern matching one or more pathnames of archive members. A pattern needs  
110429 to be given in the name-generating notation of the pattern matching notation in  
110430 [Section 2.14](#) (on page 2523), including the filename expansion rules in [Section](#)  
110431 [2.14.3](#) (on page 2525). The default, if no *pattern* is specified, is to select all members  
110432 in the archive.
- 110433 **STDIN**
- 110434 In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a  
110435 file containing a list of pathnames, each terminated by a <newline> character.
- 110436 In **list** and **read** modes, if **-f** is not specified, the standard input shall be an archive file.
- 110437 Otherwise, the standard input shall not be used.
- 110438 **INPUT FILES**
- 110439 The input file named by the *archive* option-argument, or standard input when the archive is read  
110440 from there, shall be a file formatted according to one of the specifications in the EXTENDED  
110441 DESCRIPTION section or some other implementation-defined format.
- 110442 The file `/dev/tty` shall be used to write prompts and read responses.
- 110443 **ENVIRONMENT VARIABLES**
- 110444 The following environment variables shall affect the execution of *pax*:
- 110445 *LANG* Provide a default value for the internationalization variables that are unset or null.  
110446 (See [XBD Section 8.2](#) (on page 169) the precedence of internationalization variables  
110447 used to determine the values of locale categories.)
- 110448 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
110449 internationalization variables.
- 110450 *LC\_COLLATE*
- 110451 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
110452 character collating elements used in the pattern matching expressions for the  
110453 *pattern* operand and the basic regular expression for the **-s** option.
- 110454 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
110455 characters (for example, single-byte as opposed to multi-byte characters in  
110456 arguments and input files), and the behavior of character classes used in the  
110457 pattern matching expressions for the *pattern* operand and the basic regular  
110458 expression for the **-s** option.
- 110459 *LC\_MESSAGES*
- 110460 Determine the locale used to affect the format and contents of diagnostic messages  
110461 and prompts written to standard error.
- 110462 *LC\_TIME* Determine the format and contents of date and time strings when the **-v** option is  
110463 specified.
- 110464 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 110465 *TMPDIR* Determine the pathname that provides part of the default global extended header  
110466 record file, as described for the **-o globexthdr=** keyword in the OPTIONS section.
- 110467 *TZ* Determine the timezone used to calculate date and time strings when the **-v** option  
110468 is specified. If *TZ* is unset or null, an unspecified default timezone shall be used.

110469 **ASYNCHRONOUS EVENTS**

110470 Default.

110471 **STDOUT**

110472 In **write** mode, if **-f** is not specified, the standard output shall be the archive formatted  
 110473 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other  
 110474 implementation-defined format (see **-x format**).

110475 In **list** mode, when the **-olistopt=format** has been specified, the selected archive members shall  
 110476 be written to standard output using the format described under [List Mode Format Specifications](#)  
 110477 (on page 3258). In **list** mode without the **-olistopt=format** option, the table of contents of the  
 110478 selected archive members shall be written to standard output using the following format:

110479 "%s\n", &lt;pathname&gt;

110480 If the **-v** option is specified in **list** mode, the table of contents of the selected archive members  
 110481 shall be written to standard output using the following formats.

110482 For pathnames representing hard links to previous members of the archive:

110483 "%sΔ==Δ%s\n", &lt;ls -l listing&gt;, &lt;linkname&gt;

110484 For all other pathnames:

110485 "%s\n", &lt;ls -l listing&gt;

110486 where <ls -l listing> shall be the format specified by the *ls* utility with the **-l** option. When  
 110487 writing pathnames in this format, it is unspecified what is written for fields for which the  
 110488 underlying archive format does not have the correct information, although the correct number of  
 110489 <blank>-separated fields shall be written.

110490 In **list** mode, standard output shall not be buffered more than a pathname (plus any associated  
 110491 information and a <newline> terminator) at a time.

110492 **STDERR**

110493 If **-v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the  
 110494 standard error output using the following format:

110495 "%s\n", &lt;pathname&gt;

110496 These pathnames shall be written as soon as processing is begun on the file or archive member,  
 110497 and shall be flushed to standard error. The trailing <newline>, which shall not be buffered, is  
 110498 written when the file has been read or written.

110499 If the **-s** option is specified, and the replacement string has a trailing 'p', substitutions shall be  
 110500 written to standard error in the following format:

110501 "%sΔ&gt;&gt;Δ%s\n", &lt;original pathname&gt;, &lt;new pathname&gt;

110502 In all operating modes of *pax*, optional messages of unspecified format concerning the input  
 110503 archive format and volume number, the number of files, blocks, volumes, and media parts as  
 110504 well as other diagnostic messages may be written to standard error.

110505 In all formats, for both standard output and standard error, it is unspecified how non-printable  
 110506 characters in pathnames or link names are written.

110507 When using the **-xpax** archive format, if a filename, link name, group name, owner name, or any  
 110508 other field in an extended header record cannot be translated between the codeset in use for that  
 110509 extended header record and the character set of the current locale, *pax* shall write a diagnostic  
 110510 message to standard error, shall process the file as described for the **-o invalid=** option, and then  
 110511 shall continue processing with the next file.

110512 **OUTPUT FILES**

110513 In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the  
110514 copied output files shall be the type of the file being copied. In either mode, existing files in the  
110515 destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**),  
110516 and invalid-value (**-oinvalid=**) tests allow it.

110517 In **write** mode, the output file named by the **-f** option-argument shall be a file formatted  
110518 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other  
110519 implementation-defined format.

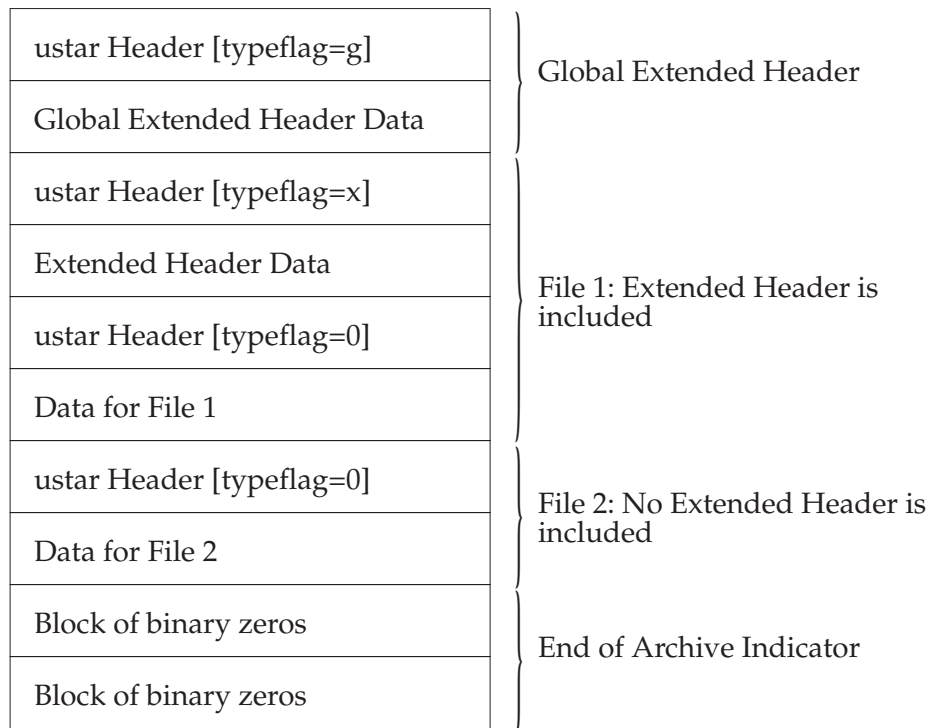
110520 **EXTENDED DESCRIPTION**110521 **pax Interchange Format**

110522 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The  
110523 physical layout of the archive shall be identical to the **ustar** format described in [ustar](#)  
110524 [Interchange Format](#) (on page 3268). Each file archived shall be represented by the following  
110525 sequence:

- 110526 • An optional header block with extended header records. This header block is of the form  
110527 described in [pax Header Block](#) (on page 3263), with a *typeflag* value of **x** or **g**. The  
110528 extended header records, described in [pax Extended Header](#) (on page 3264), shall be  
110529 included as the data for this header block.
- 110530 • A header block that describes the file. Any fields in the preceding optional extended  
110531 header shall override the associated fields in this header block for this file.
- 110532 • Zero or more blocks that contain the contents of the file.

110533 At the end of the archive file there shall be two 512-byte blocks filled with binary zeros,  
110534 interpreted as an end-of-archive indicator.

110535 A schematic of an example archive with global extended header records and two actual files is  
110536 shown in [Figure 3-1](#) (on page 3263). In the example, the second file in the archive has no  
110537 extended header preceding it, presumably because it has no need for extended attributes.



110538

Figure 3-1 pax Format Archive Example

110539

**pax Header Block**

110540

The **pax** header block shall be identical to the **ustar** header block described in [ustar Interchange Format](#) (on page 3268), except that two additional *typeflag* values are defined:

110541

110542

x Represents extended header records for the following file in the archive (which shall have its own **ustar** header block). The format of these extended header records shall be as described in [pax Extended Header](#) (on page 3264).

110543

110544

110545

g Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in [pax Extended Header](#) (on page 3264). Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag* g global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

110546

110547

110548

110549

110550

110551

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the *pax* utility. However, if this archive is read by a *pax* utility conforming to the ISO POSIX-2:1993 standard, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

110552

110553

110554

110555

110556

110557

A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit one) (hard link) may be included, which means that the size field may be greater than zero. Archives created by *pax -o linkdata* shall include these data blocks with the hard links.

110558

110559

110560 **pax Extended Header**

110561 A **pax** extended header contains values that are inappropriate for the **ustar** header block because  
 110562 of limitations in that format: fields requiring a character encoding other than that described in  
 110563 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**  
 110564 header, and fields whose format or length do not fit the requirements of the **ustar** header. The  
 110565 values in an extended header add attributes to the following file (or files; see the description of  
 110566 the *typeflag* **g** header block) or override values in the following header block(s), as indicated in  
 110567 the following list of keywords.

110568 An extended header shall consist of one or more records, each constructed as follows:

110569 `"%d %s=%s\n", <length>, <keyword>, <value>`

110570 The extended header records shall be encoded according to the ISO/IEC 10646:2020 standard  
 110571 UTF-8 encoding. The *<length>* field, *<blank>*, *<equals-sign>*, and *<newline>* shown shall be  
 110572 limited to the portable character set, as encoded in UTF-8. The *<keyword>* fields can be any  
 110573 UTF-8 characters. The *<length>* field shall be the decimal length of the extended header record  
 110574 in octets, including the trailing *<newline>*. If there is a **hdrcharset** extended header in effect for  
 110575 a file, the *value* field for any **gname**, **linkpath**, **path**, and **uname** extended header records shall be  
 110576 encoded using the character set specified by the **hdrcharset** extended header record; otherwise,  
 110577 the *value* field shall be encoded using UTF-8. The *value* field for all other keywords specified by  
 110578 POSIX.1-2024 shall be encoded using UTF-8.

110579 The *<keyword>* field shall be one of the entries from the following list or a keyword provided as  
 110580 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and  
 110581 periods are reserved for future standardization. A keyword shall not include an *<equals-sign>*.  
 110582 (In the following list, the notations `file(s)` or `block(s)` is used to acknowledge that a keyword  
 110583 affects the following single file after a *typeflag* **x** extended header, but possibly multiple files after  
 110584 *typeflag* **g**. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode  
 110585 shall apply only when such a record has not already been provided through the use of the **-o**  
 110586 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with  
 110587 applicable extended header records and then extracted.)

110588 **atime** The file access time for the following file(s), equivalent to the value of the *st\_atim*  
 110589 member of the **stat** structure for a file, as described by the *stat()* function. The  
 110590 access time shall be restored if the process has appropriate privileges required to  
 110591 do so. The format of the *<value>* shall be as described in [pax Extended Header File](#)  
 110592 [Times](#) (on page 3267).

110593 **charset** The name of the character set used to encode the data in the following file(s). The  
 110594 entries in the following table are defined to refer to known standards; additional  
 110595 names may be agreed on between the originator and recipient.



|        | <value>                 | Formal Standard               |
|--------|-------------------------|-------------------------------|
| 110596 | ISO-IRΔ646Δ1990         | ISO/IEC 646:1990              |
| 110597 | ISO-IRΔ8859Δ1Δ1998      | ISO/IEC 8859-1:1998           |
| 110598 | ISO-IRΔ8859Δ2Δ1999      | ISO/IEC 8859-2:1999           |
| 110599 | ISO-IRΔ8859Δ3Δ1999      | ISO/IEC 8859-3:1999           |
| 110600 | ISO-IRΔ8859Δ4Δ1998      | ISO/IEC 8859-4:1998           |
| 110601 | ISO-IRΔ8859Δ5Δ1999      | ISO/IEC 8859-5:1999           |
| 110602 | ISO-IRΔ8859Δ6Δ1999      | ISO/IEC 8859-6:1999           |
| 110603 | ISO-IRΔ8859Δ7Δ1987      | ISO/IEC 8859-7:1987           |
| 110604 | ISO-IRΔ8859Δ8Δ1999      | ISO/IEC 8859-8:1999           |
| 110605 | ISO-IRΔ8859Δ9Δ1999      | ISO/IEC 8859-9:1999           |
| 110606 | ISO-IRΔ8859Δ10Δ1998     | ISO/IEC 8859-10:1998          |
| 110607 | ISO-IRΔ8859Δ13Δ1998     | ISO/IEC 8859-13:1998          |
| 110608 | ISO-IRΔ8859Δ14Δ1998     | ISO/IEC 8859-14:1998          |
| 110609 | ISO-IRΔ8859Δ15Δ1999     | ISO/IEC 8859-15:1999          |
| 110610 | ISO-IRΔ10646Δ2000       | ISO/IEC 10646:2000            |
| 110611 | ISO-IRΔ10646Δ2000ΔUTF-8 | ISO/IEC 10646, UTF-8 encoding |
| 110612 | BINARY                  | None.                         |

110614 The encoding is included in an extended header for information only; when *pax* is  
 110615 used as described in POSIX.1-2024, it shall not translate the file data into any other  
 110616 encoding. The **BINARY** entry indicates unencoded binary data.

110617 When used in **write** or **copy** mode, it is implementation-defined whether *pax*  
 110618 includes a **charset** extended header record for a file.

110619 **comment** A series of characters used as a comment. All characters in the <value> field shall  
 110620 be ignored by *pax*.

110621 **gid** The group ID of the group that owns the file, expressed as a decimal number using  
 110622 digits from the ISO/IEC 646:1991 standard. This record shall override the *gid* field  
 110623 in the following header block(s). When used in **write** or **copy** mode, *pax* shall  
 110624 include a *gid* extended header record for each file whose group ID is greater than  
 110625 2 097 151 (octal 7 777 777).

110626 **gname** The group of the file(s), formatted as a group name in the group database. This  
 110627 record shall override the *gid* and *gname* fields in the following header block(s), and  
 110628 any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall  
 110629 translate the name from the encoding in the header record to the character set  
 110630 appropriate for the group database on the receiving system. If any of the characters  
 110631 cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the  
 110632 **-oinvalid=binary** option is specified, the results are implementation-defined.  
 110633 When used in **write** or **copy** mode, *pax* shall include a **gname** extended header  
 110634 record for each file whose group name cannot be represented entirely with the  
 110635 letters and digits of the portable character set.

110636 **hdrcharset** The name of the character set used to encode the value field of the **gname**,  
 110637 **linkpath**, **path**, and **uname** *pax* extended header records. The entries in the  
 110638 following table are defined to refer to known standards; additional names may be  
 110639 agreed between the originator and the recipient.

110640  
110641  
110642

| <value>                 | Formal Standard               |
|-------------------------|-------------------------------|
| ISO-IRΔ10646Δ2000ΔUTF-8 | ISO/IEC 10646, UTF-8 encoding |
| BINARY                  | None.                         |

110643  
110644  
110645

If no **hdrcharset** extended header record is specified, the default character set used to encode all values in extended header records shall be the ISO/IEC 10646:2020 standard UTF-8 encoding.

110646  
110647

The **BINARY** entry indicates that all values recorded in extended headers for affected files are unencoded binary data from the underlying system.

110648  
110649  
110650  
110651  
110652  
110653  
110654  
110655  
110656  
110657  
110658

#### **linkpath**

The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following **ustar** header block(s). The following **ustar** header block shall determine the type of link created. If *typeflag* of the following header block is 1, it shall be a hard link. If *typeflag* is 2, it shall be a symbolic link and the **linkpath** value shall be the contents of the symbolic link. The *pax* utility shall translate the name of the link (contents of the symbolic link) from the encoding in the header to the character set appropriate for the local file system. When used in **write** or **copy** mode, *pax* shall include a **linkpath** extended header record for each link whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

110659  
110660  
110661  
110662  
110663  
110664

#### **mtime**

The file modification time of the following file(s), equivalent to the value of the *st\_mtim* member of the **stat** structure for a file, as described in the *stat()* function. This record shall override the *mtime* field in the following header block(s). The modification time shall be restored if the process has appropriate privileges required to do so. The format of the <value> shall be as described in [pax Extended Header File Times](#) (on page 3267).

110665  
110666  
110667  
110668

#### **path**

The pathname of the following file(s). This record shall override the *name* and *prefix* fields in the following header block(s). The *pax* utility shall translate the pathname of the file from the encoding in the header to the character set appropriate for the local file system.

110669  
110670  
110671

When used in **write** or **copy** mode, *pax* shall include a *path* extended header record for each file whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

110672

**realtime.any** The keywords prefixed by ``realtime.'' are reserved for future standardization.

110673

**security.any** The keywords prefixed by ``security.'' are reserved for future standardization.

110674  
110675  
110676  
110677  
110678

#### **size**

The size of the file in octets, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *size* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *size* extended header record for each file with a size value greater than 8 589 934 591 (octal 77 777 777 777).

110679  
110680  
110681  
110682  
110683

#### **uid**

The user ID of the file owner, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *uid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *uid* extended header record for each file whose owner ID is greater than 2 097 151 (octal 7 777 777).

110684 **uname** The owner of the following file(s), formatted as a user name in the user database.  
 110685 This record shall override the *uid* and *uname* fields in the following header block(s),  
 110686 and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax*  
 110687 shall translate the name from the encoding in the header record to the character set  
 110688 appropriate for the user database on the receiving system. If any of the characters  
 110689 cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the  
 110690 **-oinvalid=binary** option is specified, the results are implementation-defined.  
 110691 When used in **write** or **copy** mode, *pax* shall include a **uname** extended header  
 110692 record for each file whose user name cannot be represented entirely with the letters  
 110693 and digits of the portable character set.

110694 If the *<value>* field is zero length, it shall delete any header block field, previously entered  
 110695 extended header value, or global extended header value of the same name.

110696 If a keyword in an extended header record (or in a **-o** option-argument) overrides or deletes a  
 110697 corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block  
 110698 field.

110699 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the  
 110700 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**  
 110701 header block fields in Table 3-15 (on page 3268) shall apply to the extended header records.

#### 110702 **pax Extended Header Keyword Precedence**

110703 This section describes the precedence in which the various header records and fields and  
 110704 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or  
 110705 **list** modes, it shall determine a file attribute in the following sequence:

- 110706 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step  
 110707 7., if applicable, or ignored otherwise.
- 110708 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
- 110709 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
- 110710 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the  
 110711 *<value>*. When extended header records conflict, the last one given in the header shall  
 110712 take precedence.
- 110713 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
- 110714 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be  
 110715 assigned the *<value>*. When global extended header records conflict, the last one given in  
 110716 the global header shall take precedence.
- 110717 7. Otherwise, the attribute shall be determined from the **ustar** header block.

#### 110718 **pax Extended Header File Times**

110719 The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's  
 110720 modification time cannot be represented exactly in the **ustar** header logical record described in  
 110721 [ustar Interchange Format](#) (on page 3268). This can occur if the time is out of **ustar** range, or if  
 110722 the file system of the underlying implementation supports non-integer time granularities and  
 110723 the time is not an integer. All of these time records shall be formatted as a decimal representation  
 110724 of the time in seconds since the Epoch. If a *<period>* ( ' . ' ) decimal point character is present,  
 110725 the digits to the right of the point shall represent the units of a subsecond timing granularity,  
 110726 where the first digit is tenths of a second and each subsequent digit is a tenth of the previous  
 110727 digit. In **read** or **copy** mode, the *pax* utility shall truncate the time of a file to the greatest value

110728 that is not greater than the input header file time. In **write** or **copy** mode, the *pax* utility shall  
 110729 output a time exactly if it can be represented exactly as a decimal number, and otherwise shall  
 110730 generate only enough digits so that the same time shall be recovered if the file is extracted on a  
 110731 system whose underlying implementation supports the same time granularity.

### 110732 **ustar Interchange Format**

110733 A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a  
 110734 fixed-size logical record of 512 octets (see below). Although this format may be thought of as  
 110735 being stored on 9-track industry-standard 12.7 mm (0.5 in) magnetic tape, other types of  
 110736 transportable media are not excluded. Each file archived shall be represented by a header logical  
 110737 record that describes the file, followed by zero or more logical records that give the contents of  
 110738 the file. At the end of the archive file there shall be two 512-octet logical records filled with  
 110739 binary zeros, interpreted as an end-of-archive indicator.

110740 The logical records may be grouped for physical I/O operations, as described under the  
 110741 **-bblocksize** and **-x ustar** options. Each group of logical records may be written with a single  
 110742 operation equivalent to the *write()* function. On magnetic tape, the result of this write shall be a  
 110743 single tape physical block. The last physical block shall always be the full size, so logical records  
 110744 after the two zero logical records may contain undefined data.

110745 The header logical record shall be structured as shown in the following table. All lengths and  
 110746 offsets are in decimal.

110747 **Table 3-15** ustar Header Block

| Field Name      | Octet Offset | Length (in Octets) |
|-----------------|--------------|--------------------|
| <i>name</i>     | 0            | 100                |
| <i>mode</i>     | 100          | 8                  |
| <i>uid</i>      | 108          | 8                  |
| <i>gid</i>      | 116          | 8                  |
| <i>size</i>     | 124          | 12                 |
| <i>mtime</i>    | 136          | 12                 |
| <i>chksum</i>   | 148          | 8                  |
| <i>typeflag</i> | 156          | 1                  |
| <i>linkname</i> | 157          | 100                |
| <i>magic</i>    | 257          | 6                  |
| <i>version</i>  | 263          | 2                  |
| <i>uname</i>    | 265          | 32                 |
| <i>gname</i>    | 297          | 32                 |
| <i>devmajor</i> | 329          | 8                  |
| <i>devminor</i> | 337          | 8                  |
| <i>prefix</i>   | 345          | 155                |

110765 All characters in the header logical record shall be represented in the coded character set of the  
 110766 ISO/IEC 646: 1991 standard. For maximum portability between implementations, names should  
 110767 be selected from characters represented by the portable filename character set as octets with the  
 110768 most significant bit zero. If an implementation supports the use of characters outside of <slash>  
 110769 and the portable filename character set in names for files, users, and groups, one or more  
 110770 implementation-defined encodings of these characters shall be provided for interchange  
 110771 purposes.

110772 However, the *pax* utility shall never create filenames on the local system that cannot be accessed

110773 via the procedures described in POSIX.1-2024. If a filename is found on the medium that would  
 110774 create an invalid filename, it is implementation-defined whether the data from the file is stored  
 110775 on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these  
 110776 files as long as it produces an error indicating that the file is being ignored.

110777 Each field within the header logical record is contiguous; that is, there is no padding used. Each  
 110778 character on the archive medium shall be stored contiguously.

110779 The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character.  
 110780 The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all  
 110781 characters in the array contain non-NUL characters including the last character. The *version* field  
 110782 is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character.  
 110783 All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991  
 110784 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

110785 The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be  
 110786 formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up  
 110787 to the first NUL character), a <slash> character, and *name*; otherwise, *name* is used alone. In  
 110788 either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it  
 110789 shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a  
 110790 pathname does not fit in the space provided, *pax* shall notify the user of the error, and shall not  
 110791 store any part of the file—header or data—on the medium.

110792 The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a  
 110793 *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall  
 110794 notify the user of the error, and shall not attempt to store the link on the medium.

110795 The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit  
 110796 representation. The encoded bits shall represent the following values:

110797 **Table 3-16** *ustar mode* Field

| 110798 | Bit Value | POSIX.1-2024 Bit | Description                                     |
|--------|-----------|------------------|-------------------------------------------------|
| 110799 | 04 000    | S_ISUID          | Set UID on execution.                           |
| 110800 | 02 000    | S_ISGID          | Set GID on execution.                           |
| 110801 | 01 000    | <reserved>       | Reserved for future standardization.            |
| 110802 | 00 400    | S_IRUSR          | Read permission for file owner class.           |
| 110803 | 00 200    | S_IWUSR          | Write permission for file owner class.          |
| 110804 | 00 100    | S_IXUSR          | Execute/search permission for file owner class. |
| 110805 | 00 040    | S_IRGRP          | Read permission for file group class.           |
| 110806 | 00 020    | S_IWGRP          | Write permission for file group class.          |
| 110807 | 00 010    | S_IXGRP          | Execute/search permission for file group class. |
| 110808 | 00 004    | S_IROTH          | Read permission for file other class.           |
| 110809 | 00 002    | S_IWOTH          | Write permission for file other class.          |
| 110810 | 00 001    | S_IXOTH          | Execute/search permission for file other class. |

110811 When appropriate privileges are required to set one of these mode bits, and the user restoring  
 110812 the files from the archive does not have appropriate privileges, the mode bits for which the user  
 110813 does not have appropriate privileges shall be ignored. Some of the mode bits in the archive  
 110814 format are not mentioned elsewhere in this volume of POSIX.1-2024. If the implementation does  
 110815 not support those bits, they may be ignored.

110816 The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

110817 The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type  
 110818 1 (a hard link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is

- 110819 set to specify a file of type 5 (directory), the *size* field shall be interpreted as described under the  
 110820 definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag*  
 110821 field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size*  
 110822 field is unspecified by this volume of POSIX.1-2024, and no data logical records shall be stored  
 110823 on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If the  
 110824 *typeflag* field is set to any other value, the number of logical records written following the header  
 110825 shall be  $(size+511)/512$ , ignoring any fraction in the result of the division.
- 110826 The *mtime* field shall be the modification time of the file at the time it was archived. It is the  
 110827 ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained  
 110828 from the *stat()* function.
- 110829 The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of  
 110830 the simple sum of all octets in the header logical record. Each octet in the header shall be treated  
 110831 as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the  
 110832 precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is  
 110833 treated as if it were all <space> characters.
- 110834 The *typeflag* field specifies the type of file archived. If a particular implementation does not  
 110835 recognize the type, or the user does not have appropriate privileges to create that type, the file  
 110836 shall be extracted as if it were a regular file if the file type is defined to have a meaning for the  
 110837 *size* field that could cause data logical records to be written on the medium (see the previous  
 110838 description for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error  
 110839 indicating that the conversion took place. All of the *typeflag* fields shall be coded in the  
 110840 ISO/IEC 646:1991 standard IRV:
- |        |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 110841 | 0    | Represents a regular file. For backwards-compatibility, a <i>typeflag</i> value of binary zero (' <code>\0</code> ') should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a <i>typeflag</i> value of the ISO/IEC 646:1991 standard IRV ' <code>0</code> '.                                       |
| 110842 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110843 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110844 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110845 | 1    | Represents a file linked to another file, of any type, previously archived. Such files are identified by having the same device and file serial numbers, and pathnames that refer to different directory entries. All such files shall be archived as linked files. The linked-to name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than 100 octets in length.                  |
| 110846 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110847 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110848 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110849 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110850 | 2    | Represents a symbolic link. The contents of the symbolic link shall be stored in the <i>linkname</i> field.                                                                                                                                                                                                                                                                                                                |
| 110851 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110852 | 3, 4 | Represent character special files and block special files respectively. In this case the <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format of which is unspecified by this volume of POSIX.1-2024. Implementations may map the device specifications to their own local specification or may ignore the entry.                                                          |
| 110853 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110854 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110855 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110856 | 5    | Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i> field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the <i>size</i> field. |
| 110857 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110858 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110859 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110860 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110861 | 6    | Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.                                                                                                                                                                                                                                                                                            |
| 110862 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110863 | 7    | Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0).                                                                                                                                                                                                                         |
| 110864 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 110865 |      |                                                                                                                                                                                                                                                                                                                                                                                                                            |

110866 A-Z The letters 'A' to 'Z', inclusive, are reserved for custom implementations. All other  
110867 values are reserved for future versions of this standard.

110868 It is unspecified whether files with pathnames that refer to the same directory entry are archived  
110869 as linked files or as separate files. If they are archived as linked files, this means that attempting  
110870 to extract both pathnames from the resulting archive always causes an error (unless the `-u`  
110871 option is used) because the link cannot be created.

110872 It is unspecified whether files with the same device and file serial numbers being appended to  
110873 an archive are treated as linked files to members that were in the archive before the append.

110874 Attempts to archive a socket shall produce a diagnostic message when **ustar** interchange format  
110875 is used, but may be allowed when **pax** interchange format is used. Handling of other file types is  
110876 implementation-defined.

110877 The *magic* field is the specification that this archive was output in this archive format. If this field  
110878 contains **ustar** (the five characters from the ISO/IEC 646:1991 standard IRV shown followed by  
110879 NUL), the *uname* and *gname* fields shall contain the ISO/IEC 646:1991 standard IRV  
110880 representation of the owner and group of the file, respectively (truncated to fit, if necessary).  
110881 When the file is restored by a privileged, protection-preserving version of the utility, the user  
110882 and group databases shall be scanned for these names. If found, the user and group IDs  
110883 contained within these files shall be used rather than the values contained within the *uid* and *gid*  
110884 fields.

#### 110885 **cpio Interchange Format**

110886 The octet-oriented **cpio** archive format shall be a series of entries, each comprising a header that  
110887 describes the file, the name of the file, and then the contents of the file.

110888 An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used  
110889 only to make physical I/O more efficient. The last group of blocks shall always be at the full  
110890 size.

110891 For the octet-oriented **cpio** archive format, the individual entry information shall be in the order  
110892 indicated and described by the following table; see also the **<cpio.h>** header.

110893

**Table 3-17** Octet-Oriented cpio Archive Entry

110894

| Header Field Name    | Length (in Octets) | Interpreted as  |
|----------------------|--------------------|-----------------|
| <i>c_magic</i>       | 6                  | Octal number    |
| <i>c_dev</i>         | 6                  | Octal number    |
| <i>c_ino</i>         | 6                  | Octal number    |
| <i>c_mode</i>        | 6                  | Octal number    |
| <i>c_uid</i>         | 6                  | Octal number    |
| <i>c_gid</i>         | 6                  | Octal number    |
| <i>c_nlink</i>       | 6                  | Octal number    |
| <i>c_rdev</i>        | 6                  | Octal number    |
| <i>c_mtime</i>       | 11                 | Octal number    |
| <i>c_namesize</i>    | 6                  | Octal number    |
| <i>c_filesize</i>    | 11                 | Octal number    |
| Filename Field Name  | Length             | Interpreted as  |
| <i>c_name</i>        | <i>c_namesize</i>  | Pathname string |
| File Data Field Name | Length             | Interpreted as  |
| <i>c_filedata</i>    | <i>c_filesize</i>  | Data            |

110895

110896

110897

110898

110899

110900

110901

110902

110903

110904

110905

110906

110907

110908

110909

110910

**cpio Header**

110911

110912

110913

110914

110915

110916

For each file in the archive, a header as defined previously shall be written. The information in the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted as octal numbers. The octal numbers shall be extended to the necessary length by appending the ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result is written to the most-significant digit of the stream of octets first. The fields shall be interpreted as follows:

110917

110918

*c\_magic* Identify the archive as being a transportable archive by containing the identifying value "070707".

110919

110920

110921

*c\_dev, c\_ino* Contains values that uniquely identify the file within the archive (that is, no files contain the same pair of *c\_dev* and *c\_ino* values unless they are links to the same file). The values shall be determined in an unspecified manner.

110922

*c\_mode* Contains the file type and access permissions as defined in the following table.



110923

Table 3-18 Values for `cpio c_mode` Field

110924

110925

110926

110927

110928

110929

110930

110931

110932

110933

110934

110935

110936

110937

110938

110939

110940

110941

110942

110943

110944

110945

| File Permissions Name | Value    | Indicates              |
|-----------------------|----------|------------------------|
| C_IRUSR               | 000 400  | Read by owner          |
| C_IWUSR               | 000 200  | Write by owner         |
| C_IXUSR               | 000 100  | Execute by owner       |
| C_IRGRP               | 000 040  | Read by group          |
| C_IWGRP               | 000 020  | Write by group         |
| C_IXGRP               | 000 010  | Execute by group       |
| C_IROTH               | 000 004  | Read by others         |
| C_IWOTH               | 000 002  | Write by others        |
| C_IXOTH               | 000 001  | Execute by others      |
| C_ISUID               | 004 000  | Set <i>uid</i>         |
| C_ISGID               | 002 000  | Set <i>gid</i>         |
| C_ISVTX               | 001 000  | Reserved               |
| File Type Name        | Value    | Indicates              |
| C_ISDIR               | 040 000  | Directory              |
| C_ISFIFO              | 010 000  | FIFO                   |
| C_ISREG               | 0100 000 | Regular file           |
| C_ISLNK               | 0120 000 | Symbolic link          |
| C_ISBLK               | 060 000  | Block special file     |
| C_ISCHR               | 020 000  | Character special file |
| C_ISSOCK              | 0140 000 | Socket                 |
| C_ISCTG               | 0110 000 | Reserved               |

110946

110947

110948

110949

110950

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of POSIX.1-2024; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

110951

*c\_uid*

Contains the user ID of the owner.

110952

*c\_gid*

Contains the group ID of the group.

110953

*c\_nlink*

Contains a number greater than or equal to the number of links in the archive referencing the file. If the `-a` option is used to append to a *cpio* archive, then the *pax* utility need not account for the files in the existing part of the archive when calculating the *c\_nlink* values for the appended part of the archive, and need not alter the *c\_nlink* values in the existing part of the archive if additional files with the same *c\_dev* and *c\_ino* values are appended to the archive.

110954

110955

110956

110957

110958

110959

*c\_rdev*

Contains implementation-defined information for character or block special files.

110960

*c\_mtime*

Contains the latest time of modification of the file at the time the archive was created.

110961

110962

*c\_namesize*

Contains the length of the pathname, including the terminating NUL character.

110963

*c\_filesiz*

Contains the length in octets of the data section following the header structure.

110964 **cpio Filename**

110965 The *c\_name* field shall contain the pathname of the file. The length of this field in octets is the  
110966 value of *c\_namesize*.

110967 If a filename is found on the medium that would create an invalid pathname, it is  
110968 implementation-defined whether the data from the file is stored on the file hierarchy and under  
110969 what name it is stored.

110970 All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum  
110971 portability between implementations, names should be selected from characters represented by  
110972 the portable filename character set as octets with the most significant bit zero. If an  
110973 implementation supports the use of characters outside the portable filename character set in  
110974 names for files, users, and groups, one or more implementation-defined encodings of these  
110975 characters shall be provided for interchange purposes. However, the *pax* utility shall never create  
110976 filenames on the local system that cannot be accessed via the procedures described previously in  
110977 this volume of POSIX.1-2024. If a filename is found on the medium that would create an invalid  
110978 filename, it is implementation-defined whether the data from the file is stored on the local file  
110979 system and under what name it is stored. The *pax* utility may choose to ignore these files as long  
110980 as it produces an error indicating that the file is being ignored.

110981 **cpio File Data**

110982 Following *c\_name*, there shall be *c\_filesiz*e octets of data. Interpretation of such data occurs in a  
110983 manner dependent on the file. For regular files, the data shall consist of the contents of the file.  
110984 For symbolic links, the data shall consist of the contents of the symbolic link. If *c\_filesiz*e is zero,  
110985 no data shall be contained in *c\_filedata*.

110986 When restoring from an archive:

- 110987 • If the user does not have appropriate privileges to create a file of the specified type, *pax*  
110988 shall ignore the entry and write an error message to standard error.
- 110989 • Only regular files and symbolic links have data to be restored. Presuming a regular file  
110990 meets any selection criteria that might be imposed on the format-reading utility by the  
110991 user, such data shall be restored.
- 110992 • If a user does not have appropriate privileges to set a particular mode flag, the flag shall be  
110993 ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this  
110994 volume of POSIX.1-2024. If the implementation does not support those flags, they may be  
110995 ignored.

110996 **cpio Special Entries**

110997 FIFO special files, directories, and the trailer shall be recorded with *c\_filesiz*e equal to zero.  
110998 Symbolic links shall be recorded with *c\_filesiz*e equal to the length of the contents of the symbolic  
110999 link. For other special files, *c\_filesiz*e is unspecified by this volume of POSIX.1-2024. The header  
111000 for the next file entry in the archive shall be written directly after the last octet of the file entry  
111001 preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive;  
111002 the contents of octets in the last block of the archive following such a header are undefined.

111003 **EXIT STATUS**

111004 The following exit values shall be returned:

- 111005 0 All files were processed successfully.

111006 >0 An error occurred.

### 111007 CONSEQUENCES OF ERRORS

111008 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an  
 111009 archive, or cannot preserve the user ID, group ID, or file mode when the **-p** option is specified, a  
 111010 diagnostic message shall be written to standard error and a non-zero exit status shall be  
 111011 returned, but processing shall continue. In the case where *pax* cannot create a hard link to a file,  
 111012 *pax* shall not, by default, create a second copy of the file.

111013 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may  
 111014 have only partially extracted the file or (if the **-n** option was not specified) may have extracted a  
 111015 file of the same name as that specified by the user, but which is not the file the user wanted.  
 111016 Additionally, the file modes of extracted directories may have additional bits from the S\_IRWXU  
 111017 mask set as well as incorrect modification and access times.

### 111018 APPLICATION USAGE

111019 Caution is advised when using the **-a** option to append to a *cpio* format archive. If any of the  
 111020 files being appended happen to be given the same *c\_dev* and *c\_ino* values as a file in the existing  
 111021 part of the archive, then they may be treated as links to that file on extraction. Thus, it is risky to  
 111022 use **-a** with *cpio* format except when it is done on the same system that the original archive was  
 111023 created on, and with the same *pax* utility, and in the knowledge that there has been little or no  
 111024 file system activity since the original archive was created that could lead to any of the files  
 111025 appended being given the same *c\_dev* and *c\_ino* values as an unrelated file in the existing part of  
 111026 the archive. Also, when (intentionally) appending additional links to a file in the existing part of  
 111027 the archive, the *c\_nlink* values in the modified archive can be smaller than the number of links to  
 111028 the file in the archive, which may mean that the links are not preserved on extraction.

111029 The **-p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio*  
 111030 implementations. In particular, the two utilities use **-m** in diametrically opposed ways. The **-p**  
 111031 option also provides a consistent means of extending the ways in which future file attributes can  
 111032 be addressed, such as for enhanced security systems or high-performance files. Although it may  
 111033 seem complex, there are really two modes that are most commonly used:

111034 **-p e** “Preserve everything”. This would be used by the historical superuser, someone with  
 111035 all appropriate privileges, to preserve all aspects of the files as they are recorded in the  
 111036 archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined attributes.

111037 **-p p** “Preserve” the file mode bits. This would be used by the user with regular privileges  
 111038 who wished to preserve aspects of the file other than the ownership. The file times are  
 111039 preserved by default, but two other flags are offered to disable these and use the time of  
 111040 extraction.

111041 The one pathname per line format of standard input precludes pathnames containing <newline>  
 111042 characters. Although such pathnames violate the portable filename guidelines, they may exist  
 111043 and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from  
 111044 historical archive programs. The problem can be avoided by listing filename arguments on the  
 111045 command line instead of on standard input.

111046 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this  
 111047 volume of POSIX.1-2024. Specifically, creating files of type block special or character special,  
 111048 restoring file access times unless the files are owned by the user (the **-t** option), or preserving file  
 111049 owner, group, and mode (the **-p** option) all probably require appropriate privileges.

111050 In **read** mode, implementations are permitted to overwrite files when the archive has multiple  
 111051 members with the same name. This may fail if permissions on the first version of the file do not  
 111052 permit it to be overwritten.

111053 The **cpio** and **ustar** formats can only support files up to 8 589 934 592 bytes ( $8 * 2^{30}$ ) in size.

111054 When archives containing binary header information are listed , the filenames printed may  
111055 cause strange behavior on some terminals.

111056 When all of the following are true:

- 111057 1. A file of type directory is being placed into an archive.
- 111058 2. The **ustar** archive format is being used.
- 111059 3. The pathname of the directory is less than or equal to 155 bytes long (it will fit in the *prefix*  
111060 field in the **ustar** header block).
- 111061 4. The last component of the pathname of the directory is longer than 100 bytes long (it will  
111062 not fit in the *name* field in the **ustar** header block).

111063 some implementations of the *pax* utility will place the entire directory pathname in the *prefix*  
111064 field, set the *name* field to an empty string, and place the directory in the archive. Other  
111065 implementations of the *pax* utility will give an error under these conditions because the *name*  
111066 field is not large enough to hold the last component of the directory name. This standard allows  
111067 either behavior. However, when extracting a directory from a **ustar** format archive, this standard  
111068 requires that all implementations be able to extract a directory even if the *name* field contains an  
111069 empty string as long as the *prefix* field does not also contain an empty string.

111070 When restricting file hierarchy traversal to one file system, it can sometimes be desirable for the  
111071 crossing points themselves to be processed (archived or copied) and sometimes for them not to  
111072 be processed. (Crossing points are mount points and, if the **-L** option is specified, symbolic links  
111073 to directories on other file systems.) With the **-X** option *pax* processes them, but there is no  
111074 standard way to have *pax* not process them. However, this can be achieved by using *find* to do  
111075 the hierarchy traversal and piping the output of *find* to *pax* (with the **-d** option); see the  
111076 APPLICATION USAGE for *find*.

#### 111077 EXAMPLES

111078 The following command:

```
111079 pax -w -f /dev/rmt/1m .
```

111080 copies the contents of the current directory to tape drive 1, medium density (assuming historical  
111081 System V device naming procedures—the historical BSD device name would be **/dev/rmt9**).

111082 The following commands:

```
111083 mkdir newdir
111084 pax -rw olddir newdir
```

111085 copy the *olddir* directory hierarchy to *newdir*.

```
111086 pax -r -s ',^//*usr//*,,' -f a.pax
```

111087 reads the archive **a.pax**, with all files rooted in **/usr** in the archive extracted relative to the current  
111088 directory.

111089 Using the option:

```
111090 -o listopt="%M %(atime)T %(size)D %(name)s"
```

111091 overrides the default output description in Standard Output and instead writes:

```
111092 -rw-rw--- Jan 12 15:53 2003 1492 /usr/foo/bar
```

111093 Using the options:

```
111094 -o listopt='%L\t%(size)D\n%.7' \
111095 -o listopt='(name)s\n%(atime)T\n%T'
```

111096 overrides the default output description in Standard Output and instead writes:

```
111097 /usr/foo/bar -> /tmp 1492
111098 /usr/fo
111099 Jan 12 15:53 1991
111100 Jan 31 15:53 2003
```

#### 111101 RATIONALE

111102 The *pax* utility was new for the ISO POSIX-2:1993 standard. It represents a peaceful compromise  
111103 between advocates of the historical *tar* and *cpio* utilities.

111104 A fundamental difference between *cpio* and *tar* was in the way directories were treated. The *cpio*  
111105 utility did not treat directories differently from other files, and to select a directory and its  
111106 contents required that each file in the hierarchy be explicitly specified. For *tar*, a directory  
111107 matched every file in the file hierarchy it rooted.

111108 The *pax* utility offers both interfaces; by default, directories map into the file hierarchy they root.  
111109 The `-d` option causes *pax* to skip any file not explicitly referenced, as *cpio* historically did. The *tar*  
111110 `-style` behavior was chosen as the default because it was believed that this was the more  
111111 common usage and because *tar* is the more commonly available interface, as it was historically  
111112 provided on both System V and BSD implementations.

111113 The data interchange format specification in this volume of POSIX.1-2024 requires that processes  
111114 with “appropriate privileges” shall always restore the ownership and permissions of extracted  
111115 files exactly as archived. If viewed from the historic equivalence between superuser and  
111116 “appropriate privileges”, there are two problems with this requirement. First, users running as  
111117 superusers may unknowingly set dangerous permissions on extracted files. Second, it is  
111118 needlessly limiting, in that superusers cannot extract files and own them as superuser unless the  
111119 archive was created by the superuser. (It should be noted that restoration of ownerships and  
111120 permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to  
111121 avoid these two problems, the *pax* specification has an additional “privilege” mechanism, the `-p`  
111122 option. Only a *pax* invocation with the privileges needed, and which has the `-p` option set using  
111123 the `e` specification character, has appropriate privileges to restore full ownership and permission  
111124 information.

111125 Note also that this volume of POSIX.1-2024 requires that the file ownership and access  
111126 permissions shall be set, on extraction, in the same fashion as the *creat()* function when provided  
111127 with the mode stored in the archive. This means that the file creation mask of the user is applied  
111128 to the file permissions.

111129 Users should note that directories may be created by *pax* while extracting files with permissions  
111130 that are different from those that existed at the time the archive was created. When extracting  
111131 sensitive information into a directory hierarchy that no longer exists, users are encouraged to set  
111132 their file creation mask appropriately to protect these files during extraction.

111133 The table of contents output is written to standard output to facilitate pipeline processing.

111134 An early proposal had hard links displaying for all pathnames. This was removed because it  
111135 complicates the output of the case where `-v` is not specified and does not match historical *cpio*  
111136 usage. The hard-link information is available in the `-v` display.

111137 The description of the `-l` option allows implementations to make hard links to symbolic links.  
111138 Earlier versions of this standard did not specify any way to create a hard link to a symbolic link,  
111139 but many implementations provided this capability as an extension. If there are hard links to

111140 symbolic links when an archive is created, the implementation is required to archive the hard  
111141 link in the archive (unless **-H** or **-L** is specified). When in **read** mode and in **copy** mode,  
111142 implementations supporting hard links to symbolic links should use them when appropriate.

111143 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that have  
111144 been brought along from historical usage. For example, there are restrictions on the length of  
111145 pathnames stored in the archive. When *pax* is used in **copy(-rw)** mode (copying directory  
111146 hierarchies), the ability to use extensions from the **-xpax** format overcomes these restrictions.

111147 The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard  
111148 block-size values for *cpio*, set when the **-B** option is specified. (The other default block-size value  
111149 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240  
111150 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The  
111151 maximum block size of 32 256 bytes ( $2^{15}$ -512 bytes) is the largest multiple of 512 bytes that fits  
111152 into a signed 16-bit tape controller transfer register. There are known limitations in some  
111153 historical systems that would prevent larger blocks from being accepted. Historical values were  
111154 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate  
111155 archives. Also, default block sizes for any file type other than character special file has been  
111156 deleted from this volume of POSIX.1-2024 as unimportant and not likely to affect the structure of  
111157 the resulting archive.

111158 Implementations are permitted to modify the block-size value based on the archive format or the  
111159 device to which the archive is being written. This is to provide implementations with the  
111160 opportunity to take advantage of special types of devices, and it should not be used without a  
111161 great deal of consideration as it almost certainly decreases archive portability.

111162 The intended use of the **-n** option was to permit extraction of one or more files from the archive  
111163 without processing the entire archive. This was viewed by the standard developers as offering  
111164 significant performance advantages over historical implementations. The **-n** option in early  
111165 proposals had three effects; the first was to cause special characters in patterns to not be treated  
111166 specially. The second was to cause only the first file that matched a pattern to be extracted. The  
111167 third was to cause *pax* to write a diagnostic message to standard error when no file was found  
111168 matching a specified pattern. Only the second behavior is retained by this volume of  
111169 POSIX.1-2024, for many reasons. First, it is in general not acceptable for a single option to have  
111170 multiple effects. Second, the ability to make pattern matching characters act as normal characters  
111171 is useful for parts of *pax* other than file extraction. Third, a finer degree of control over the  
111172 special characters is useful because users may wish to normalize only a single special character  
111173 in a single filename. Fourth, given a more general escape mechanism, the previous behavior of  
111174 the **-n** option can be easily obtained using the **-s** option or a *sed* script. Finally, writing a  
111175 diagnostic message when a pattern specified by the user is unmatched by any file is useful  
111176 behavior in all cases.

111177 In this version, the **-n** was removed from the **copy** mode synopsis of *pax*; it is inapplicable  
111178 because there are no pattern operands specified in this mode.

111179 There is another method than *pax* for copying subtrees in POSIX.1-2024 described as part of the  
111180 *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive interface,  
111181 while *pax* offers a finer granularity of control. Each provides additional functionality to the  
111182 other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not. It is  
111183 the intention of the standard developers that the results be similar (using appropriate option  
111184 combinations in both utilities). The results are not required to be identical; there seemed  
111185 insufficient gain to applications to balance the difficulty of implementations having to guarantee  
111186 that the results would be exactly identical.

111187 A single archive may span more than one file. It is suggested that implementations provide

111188 informative messages to the user on standard error whenever the archive file is changed.

111189 The **-d** option (do not create intermediate directories not listed in the archive) found in early  
111190 proposals was originally provided as a complement to the historic **-d** option of *cpio*. It has been  
111191 deleted.

111192 The **-s** option in early proposals specified a subset of the substitution command from the *ed*  
111193 utility. As there was no reason for only a subset to be supported, the **-s** option is now compatible  
111194 with the current *ed* specification. Since the delimiter can be any non-null character, the following  
111195 usage with single <space> characters is valid:

```
111196 pax -s " foo bar " . . .
```

111197 The **-t** description is worded so as to note that this may cause the access time update caused by  
111198 some other activity (which occurs while the file is being read) to be overwritten.

111199 The default behavior of *pax* with regard to file modification times is the same as historical  
111200 implementations of *tar*. It is not the historical behavior of *cpio*.

111201 Because the **-i** option uses */dev/tty*, utilities without a controlling terminal are not able to use  
111202 this option.

111203 The **-y** option, found in early proposals, has been deleted because a line containing a single  
111204 <period> for the **-i** option has equivalent functionality. The special lines for the **-i** option (a  
111205 single <period> and the empty line) are historical practice in *cpio*.

111206 In early drafts, a **-echarmap** option was included to increase portability of files between systems  
111207 using different coded character sets. This option was omitted because it was apparent that  
111208 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate  
111209 substitute.

111210 The ISO POSIX-2:1993 standard and ISO POSIX-1 standard requirements for *pax*, however,  
111211 made it very difficult to create a single archive containing files created using extended characters  
111212 provided by different locales. This version adds the **hdrcharset** keyword to make it possible to  
111213 archive files in these cases without dropping files due to translation errors.

111214 Translating filenames and other attributes from a locale's encoding to UTF-8 and then back again  
111215 can lose information, as the resulting filename might not be byte-for-byte equivalent to the  
111216 original. To avoid this problem, users can specify the **-o hdrcharset=binary** option, which will  
111217 cause the resulting archive to use binary format for all names and attributes. Such archives are  
111218 not portable among hosts that use different native encodings (e.g., EBCDIC *versus* ASCII-based  
111219 encodings), but they will allow interchange among the vast majority of POSIX file systems in  
111220 practical use. Also, the **-o hdrcharset=binary** option will cause *pax* in **copy** mode to behave  
111221 more like other standard utilities such as *cp*.

111222 If the values specified by the **-o exthdr.name=value**, **-o globexthdr.name=value**, or by  
111223 **\$TMPDIR** (if **-o globexthdr.name** is not specified) require a character encoding other than that  
111224 described in the ISO/IEC 646:1991 standard, a **path** extended header record will have to be  
111225 created for the file. If a **hdrcharset** extended header record is active for such headers, it will  
111226 determine the codeset used for the value field in these extended **path** header records. These **path**  
111227 extended header records always need to be created when writing an archive even if  
111228 **hdrcharset=binary** has been specified and would contain the same (binary) data that appears in  
111229 the **ustar** header record prefix and *name* fields. (In other words, an extended header **path** record  
111230 is always required to be generated if the *prefix* or *name* fields contain non-ASCII characters even  
111231 when **hdrcharset=binary** is also in effect for that file.)

111232 The **-k** option was added to address international concerns about the dangers involved in the  
111233 character set transformations of **-e** (if the target character set were different from the source, the

filenames might be transformed into names matching existing files) and also was made more general to protect files transferred between file systems with different {NAME\_MAX} values (truncating a filename on a smaller system might also inadvertently overwrite existing files). As stated, it prevents any overwriting, even if the target file is older than the source. This version adds more granularity of options to solve this problem by introducing the **-oinvalid=option**—specifically the **UTF-8** and **binary** actions. (Note that an existing file is still subject to overwriting in this case. The **-k** option closes that loophole.)

Some of the file characteristics referenced in this volume of POSIX.1-2024 might not be supported by some archive formats. For example, neither the **tar** nor **cpio** formats contain the file access time. For this reason, the **e** specification character has been provided, intended to cause all file characteristics specified in the archive to be retained.

It is required that extracted directories, by default, have their access and modification times and permissions set to the values specified in the archive. This has obvious problems in that the directories are almost certainly modified after being extracted and that directory permissions may not permit file creation. One possible solution is to create directories with the mode specified in the archive, as modified by the *umask* of the user, with sufficient permissions to allow file creation. After all files have been extracted, *pax* would then reset the access and modification times and permissions as necessary.

The list-mode formatting description borrows heavily from the one defined by the *printf* utility. However, since there is no separate operand list to get conversion arguments, the format was extended to allow specifying the name of the conversion argument as part of the conversion specification.

The **T** conversion specifier allows time fields to be displayed in any of the date formats. Unlike the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past. This makes parsing the output more predictable.

The **D** conversion specifier handles the ability to display the major/minor or file size, as with *ls*, by using **%-8(size)D**.

The **L** conversion specifier handles the *ls* display for symbolic links.

Conversion specifiers were added to generate existing known types used for *ls*.

### 111263 **pax Interchange Format**

The new POSIX data interchange format was developed primarily to satisfy international concerns that the **ustar** and **cpio** formats did not provide for file, user, and group names encoded in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers realized that this new POSIX data interchange format should be very extensible because there were other requirements they foresaw in the near future:

- Support international character encodings and locale information
- Support security information (ACLs, and so on)
- Support future file types, such as realtime or contiguous files
- Include data areas for implementation use
- Support systems with words larger than 32 bits and timers with subsecond granularity

The following were not goals for this format because these are better handled by separate utilities or are inappropriate for a portable format:



- 111276 • Encryption
- 111277 • Compression
- 111278 • Data translation between locales and codesets
- 111279 • *inode* storage

111280 The format chosen to support the goals is an extension of the **ustar** format. Of the two formats  
111281 previously available, only the **ustar** format was selected for extensions because:

- 111282 • It was easier to extend in an upwards-compatible way. It offered version flags and header  
111283 block type fields with room for future standardization. The **cpio** format, while possessing a  
111284 more flexible file naming methodology, could not be extended without breaking some  
111285 theoretical implementation or using a dummy filename that could be a legitimate filename.
- 111286 • Industry experience since the original “*tar wars*” fought in developing the ISO POSIX-1  
111287 standard has clearly been in favor of the **ustar** format, which is generally the default  
111288 output format selected for *pax* implementations on new systems.

111289 The new format was designed with one additional goal in mind: reasonable behavior when an  
111290 older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated  
111291 that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this  
111292 allowed the format to include all the extended information in a pseudo-regular file that  
111293 preceded each real file. An option is given that allows the archive creator to set up reasonable  
111294 names for these files on the older systems. Also, the normative text suggests that reasonable file  
111295 access values be used for this **ustar** header block. Making these header files inaccessible for  
111296 convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are  
111297 suggested.

111298 The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format  
111299 rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous  
111300 version of *pax*), mandated the behavior of the format-reading utility when it encountered an  
111301 unknown *typeflag*, but was silent about the other two fields.

111302 Early proposals for the first version of this standard contained a proposed archive format that  
111303 was based on compatibility with the standard for tape files (ISO 1001, similar to the format used  
111304 historically on many mainframes and minicomputers). This format was overly complex and  
111305 required considerable overhead in volume and header records. Furthermore, the standard  
111306 developers felt that it would not be acceptable to the community of POSIX developers, so it was  
111307 later changed to be a format more closely related to historical practice on POSIX systems.

111308 The prefix and name split of pathnames in **ustar** was replaced by the single path extended  
111309 header record for simplicity.

111310 The concept of a global extended header (*typeflag***g**) was controversial. If this were applied to an  
111311 archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape  
111312 could be a serious problem; a utility attempting to extract as many files as possible from a  
111313 damaged archive could lose a large percentage of file header information in this case. However,  
111314 if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers  
111315 considerable potential size reductions by eliminating redundant information. Thus, the text  
111316 warns against using the global method for unreliable media and provides a method for  
111317 implanting global information in the extended header for each file, rather than in the *typeflag* **g**  
111318 records.

111319 No facility for data translation or filtering on a per-file basis is included because the standard  
111320 developers could not invent an interface that would allow this in an efficient manner. If a filter,  
111321 such as encryption or compression, is to be applied to all the files, it is more efficient to apply the

111322 filter to the entire archive as a single file. The standard developers considered interfaces that  
111323 would invoke a shell script for each file going into or out of the archive, but the system overhead  
111324 in this approach was considered to be too high.

111325 One such approach would be to have **filter=** records that give a pathname for an executable.  
111326 When the program is invoked, the file and archive would be open for standard input/output  
111327 and all the header fields would be available as environment variables or command-line  
111328 arguments. The standard developers did discuss such schemes, but they were omitted from  
111329 POSIX.1-2024 due to concerns about excessive overhead. Also, the program itself would need to  
111330 be in the archive if it were to be used portably.

111331 There is currently no portable means of identifying the character set(s) used for a file in the file  
111332 system. Therefore, *pax* has not been given a mechanism to generate charset records  
111333 automatically. The only portable means of doing this is for the user to write the archive using the  
111334 **-ocharset=string** command line option. This assumes that all of the files in the archive use the  
111335 same encoding. The “implementation-defined” text is included to allow for a system that can  
111336 identify the encodings used for each of its files.

111337 The table of standards that accompanies the charset record description is acknowledged to be  
111338 very limited. Only a limited number of character set standards is reasonable for maximal  
111339 interchange. Any character set is, of course, possible by prior agreement. It was suggested that  
111340 EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal  
111341 standards, and then only those with reasonably large followings, can be included here, simply as  
111342 a matter of practicality. The *<value>*s represent names of officially registered character sets in the  
111343 format required by the ISO 2375:1985 standard.

111344 The normal *<comma>* or *<blank>*-separated list rules are not followed in the case of keyword  
111345 options to allow ease of argument parsing for *getopts*.

111346 Further information on character encodings is in [pax Archive Character Set Encoding/Decoding](#)  
111347 (on page 3284).

111348 The standard developers have reserved keyword name space for vendor extensions. It is  
111349 suggested that the format to be used is:

111350 *VENDOR.keyword*

111351 where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further  
111352 suggested that the keyword following the *<period>* be named differently than any of the  
111353 standard keywords so that it could be used for future standardization, if appropriate, by  
111354 omitting the *VENDOR* prefix.

111355 The *<length>* field in the extended header record was included to make it simpler to step  
111356 through the records, even if a record contains an unknown format (to a particular *pax*) with  
111357 complex interactions of special characters. It also provides a minor integrity checkpoint within  
111358 the records to aid a program attempting to recover files from a damaged archive.

111359 There are no extended header versions of the *devmajor* and *devminor* fields because the  
111360 unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific  
111361 extended keywords (such as *VENDOR.devmajor*) should be used.

111362 Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly  
111363 on a symbolic name basis, as in **ustar**.

111364 Just as with the **ustar** format descriptions, the new format makes no special arrangements for  
111365 multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file  
111366 and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing  
111367 their labels, and mounting each in the proper sequence are considered to be implementation

- 111368 details that cannot be described portably.
- 111369 The **pax** format is intended for interchange, not only for backup on a single (family of) systems.  
111370 It is not as densely packed as might be possible for backup:
- 111371 • It contains information as coded characters that could be coded in binary.
  - 111372 • It identifies extended records with name fields that could be omitted in favor of a fixed-  
111373 field layout.
  - 111374 • It translates names into a portable character set and identifies locale-related information,  
111375 both of which are probably unnecessary for backup.
- 111376 The requirements on restoring from an archive are slightly different from the historical wording,  
111377 allowing for non-monolithic privilege to bring forward as much as possible. In particular,  
111378 attributes such as “high performance file” might be broadly but not universally granted while  
111379 set-user-ID or *chown()* might be much more restricted. There is no implication in POSIX.1-2024  
111380 that the security information be honored after it is restored to the file hierarchy, in spite of what  
111381 might be improperly inferred by the silence on that topic. That is a topic for another standard.
- 111382 Hard links are recorded in the fashion described here because a hard link can be to any file type.  
111383 It is desirable in general to be able to restore part of an archive selectively and restore all of those  
111384 files completely. If the data is not associated with each hard link, it is not possible to do this.  
111385 However, the data associated with a file can be large, and when selective restoration is not  
111386 needed, this can be a significant burden. The archive is structured so that files that have no  
111387 associated data can always be restored by the name of any link name of any hard link, and the  
111388 user can choose whether data is recorded with each instance of a file that contains data. The  
111389 format permits mixing of hard links with data and hard links without data in a single archive;  
111390 this can be done for special needs, and *pax* is expected to interpret such archives on input  
111391 properly, despite the fact that there is no *pax* option that would force this mixed case on output.  
111392 (When **-o linkdata** is used, the output must contain the duplicate data, but the implementation  
111393 is free to include it or omit it when **-o linkdata** is not used.)
- 111394 The time values are included as extended header records for those implementations needing  
111395 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be  
111396 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject  
111397 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a  
111398 leading '-'. Even though some implementations can support finer file-time granularities than  
111399 seconds, the normative text requires support only for seconds since the Epoch because the  
111400 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new  
111401 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will  
111402 be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification  
111403 time) is described with appropriate privileges so that it can be ignored when writing to the file  
111404 system. POSIX does not provide a portable means to change file creation time. Nothing is  
111405 intended to prevent a non-portable implementation of *pax* from restoring the value.
- 111406 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the  
111407 sizes specified in the regular *tar* header. New file system architectures are emerging that will  
111408 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits  
111409 for user and group IDs, but the extended header values were included for completeness,  
111410 allowing overrides for all of the decimal values in the *tar* header.
- 111411 The standard developers intended to describe the effective results of *pax* with regard to file  
111412 ownerships and permissions; implementations are not restricted in timing or sequencing the  
111413 restoration of such, provided the results are as specified.
- 111414 Much of the text describing the extended headers refers to use in “**write** or **copy** modes”. The

111415 **copy** mode references are due to the normative text: “The effect of the copy shall be as if the  
111416 copied files were written to an archive file and then subsequently extracted ...”. There is  
111417 certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode,  
111418 but the effects must be as if it had.

#### 111419 **pax Archive Character Set Encoding/Decoding**

111420 There is a need to exchange archives of files between systems of different native codesets.  
111421 Filenames, group names, and user names must be preserved to the fullest extent possible when  
111422 an archive is read on the receiving platform. Translation of the contents of files is not within the  
111423 scope of the *pax* utility.

111424 There will also be the need to represent characters that are not available on the receiving  
111425 platform. These unsupported characters cannot be automatically folded to the local set of  
111426 characters due to the chance of collisions. This could result in overwriting previous extracted  
111427 files from the archive or pre-existing files on the system.

111428 For these reasons, the codeset used to represent characters within the extended header records of  
111429 the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields  
111430 requiring translation include, at a minimum, filenames, user names, group names, and link  
111431 pathnames. Implementations may wish to have localized extended keywords that use non-  
111432 portable characters.

111433 The standard developers considered the following options:

- 111434 • The archive creator specifies the well-defined name of the source codeset. The receiver  
111435 must then recognize the codeset name and perform the appropriate translations to the  
111436 destination codeset.
- 111437 • The archive creator includes within the archive the character mapping table for the source  
111438 codeset used to encode extended header records. The receiver must then read the  
111439 character mapping table and perform the appropriate translations to the destination  
111440 codeset.
- 111441 • The archive creator translates the extended header records in the source codeset into a  
111442 canonical form. The receiver must then perform the appropriate translations to the  
111443 destination codeset.

111444 The approach that incorporates the name of the source codeset poses the problem of codeset  
111445 name registration, and makes the archive useless to *pax* archive decoders that do not recognize  
111446 that codeset.

111447 Because parts of an archive may be corrupted, the standard developers felt that including the  
111448 character map of the source codeset was too fragile. The loss of this one key component could  
111449 result in making the entire archive useless. (The difference between this and the global extended  
111450 header decision was that the latter has a workaround—duplicating extended header records on  
111451 unreliable media—but this would be too burdensome for large character set maps.)

111452 Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the  
111453 cross-product of all source and destination codesets.

111454 To simplify the translation from the source codeset to the canonical form and from the canonical  
111455 form to the destination codeset, the standard developers decided that the internal representation  
111456 should be a stateless encoding. A stateless encoding is one where each codepoint has the same  
111457 meaning, without regard to the decoder being in a specific state. An example of a stateful  
111458 encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the  
111459 ISO/IEC 646: 1991 standard (equivalent to 7-bit ASCII).

111460 For these reasons, the standard developers decided to adopt a canonical format for the  
 111461 representation of file information strings. The obvious, well-endorsed candidate is the  
 111462 ISO/IEC 10646:2020 standard (based in part on Unicode), which can be used to represent the  
 111463 characters of virtually all standardized character sets. The standard developers initially agreed  
 111464 upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters  
 111465 provides a sufficiently rich set to represent all commonly-used codesets.

111466 However, the standard developers found that the 16-bit Unicode representation had some  
 111467 problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character  
 111468 made the extended header records twice as long for the case of strings coded entirely from  
 111469 historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the  
 111470 ISO/IEC 10646:2020 standard. This multi-byte representation encodes UCS2 or UCS4 characters  
 111471 reliably and deterministically, eliminating the need for a canonical byte ordering. In addition,  
 111472 NUL octets and other characters possibly confusing to POSIX file systems do not appear, except  
 111473 to represent themselves. It was realized that certain national codesets take up more space after  
 111474 the encoding, due to their placement within the UCS range; it was felt that the usefulness of the  
 111475 encoding of the names outweighs the disadvantage of size increase for file, user, and group  
 111476 names.

111477 The encoding of UTF-8 is as follows:

| 111478 | UCS4 Hex Encoding   | UTF-8 Binary Encoding                                 |
|--------|---------------------|-------------------------------------------------------|
| 111479 | 00000000–0000007F   | 0xxxxxxx                                              |
| 111480 | 00000080–000007FF   | 110xxxxx 10xxxxxx                                     |
| 111481 | 00000800–0000FFFF   | 1110xxxx 10xxxxxx 10xxxxxx                            |
| 111482 | 00010000–001FFFFFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx                   |
| 111483 | 00200000–03FFFFFFF  | 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx          |
| 111484 | 04000000–7FFFFFFF   | 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx |

111485 where each 'x' represents a bit value from the character being translated.

### 111486 **ustar Interchange Format**

111487 The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of  
 111488 the historical *tar* utility. The goal of these changes was not only to provide the functional  
 111489 enhancements desired, but also to retain compatibility between new and old versions. This  
 111490 compatibility has been retained. Archives written using the old archive format are compatible  
 111491 with the new format.

111492 Implementors should be aware that the previous file format did not include a mechanism to  
 111493 archive directory type files. For this reason, the convention of using a filename ending with  
 111494 <slash> was adopted to specify a directory on the archive.

111495 The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for  
 111496 {PATH\_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname  
 111497 be stored there without the use of the *prefix* field. Although the name field is known to be too  
 111498 small to contain {PATH\_MAX} characters, the value was not changed in this version of the  
 111499 archive file format to retain backwards-compatibility, and instead the prefix was introduced.  
 111500 Also, because of the earlier version of the format, there is no way to remove the restriction on the  
 111501 *linkname* field being limited in size to just that of the *name* field.

111502 The *size* field is required to be meaningful in all implementation extensions, although it could be  
 111503 zero. This is required so that the data blocks can always be properly counted.

111504 It is suggested that if device special files need to be represented that cannot be represented in the  
 111505 standard format, that one of the extension types (A-Z) be used, and that the additional

111506 information for the special file be represented as data and be reflected in the *size* field.

111507 Attempting to restore a special file type, where it is converted to ordinary data and conflicts with  
 111508 an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax*  
 111509 should not be able to overwrite the entries in, for example, */dev* in any case (whether the file is  
 111510 converted to another type or not). If run as a privileged user, it should be able to do so, and it  
 111511 would be considered a bug if it did not. The same is true of ordinary data files and similarly  
 111512 named special files; it is impossible to anticipate the needs of the user (who could really intend  
 111513 to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the  
 111514 protection system as required.

111515 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a  
 111516 **ustar** archive. POSIX.1-2024 does not require the contiguous file extension, but does define a  
 111517 standard way of archiving such files so that all conforming systems can interpret these file types  
 111518 in a meaningful and consistent manner. On a system that does not support extended file types,  
 111519 the *pax* utility should do the best it can with the file and go on to the next.

111520 The file protection modes are those conventionally used by the *ls* utility. This is extended beyond  
 111521 the usage in the ISO POSIX-2 standard to support the “shared text” or “sticky” bit. It is intended  
 111522 that the conformance document should not document anything beyond the existence of and  
 111523 support of such a mode. Further extensions are expected to these bits, particularly with  
 111524 overloading the set-user-ID and set-group-ID flags.

#### 111525 **cpio Interchange Format**

111526 The reference to appropriate privileges in the **cpio** format refers to an error on standard output;  
 111527 the **ustar** format does not make comparable statements.

111528 The model for this format was the historical System V *cpio-c* data interchange format. This  
 111529 model documents the portable version of the **cpio** format and not the binary version. It has the  
 111530 flexibility to transfer data of any type described within POSIX.1-2024, yet is extensible to transfer  
 111531 data types specific to extensions beyond POSIX.1-2024 (for example, contiguous files). Because it  
 111532 describes existing practice, there is no question of maintaining upwards-compatibility.

#### 111533 **cpio Header**

111534 There has been some concern that the size of the *c\_ino* field of the header is too small to handle  
 111535 those systems that have very large *inode* numbers. However, the *c\_ino* field in the header is used  
 111536 strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as  
 111537 the *inode* number of the file in the location from which that file is extracted.

111538 The name *c\_magic* is based on historical usage.

#### 111539 **cpio Filename**

111540 For most historical implementations of the *cpio* utility, {PATH\_MAX} octets can be used to  
 111541 describe the pathname without the addition of any other header fields (the NUL character  
 111542 would be included in this count). {PATH\_MAX} is the minimum value for pathname size,  
 111543 documented as 256 bytes. However, an implementation may use *c\_namesize* to determine the  
 111544 exact length of the pathname. With the current description of the **<cpio.h>** header, this  
 111545 pathname size can be as large as a number that is described in six octal digits.

111546 Two values are documented under the *c\_mode* field values to provide for extensibility for known  
 111547 file types:

111548 **0110 000** Reserved for contiguous files. The implementation may treat the rest of the  
 111549 information for this archive like a regular file. If this file type is undefined, the  
 111550 implementation may create the file as a regular file.

111551 This provides for extensibility of the **cpio** format while allowing for the ability to read old  
 111552 archives. Files of an unknown type may be read as “regular files” on some implementations. On  
 111553 a system that does not support extended file types, the *pax* utility should do the best it can with  
 111554 the file and go on to the next.

#### 111555 **FUTURE DIRECTIONS**

111556 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 111557 value of a <newline> character when <newline> is a terminator or separator in the output  
 111558 format being used, implementations are encouraged to treat this as an error. A future version of  
 111559 this standard may require implementations to treat this as an error.

111560 If this utility is directed to create a new directory entry that contains any bytes that have the  
 111561 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 111562 error. A future version of this standard may require implementations to treat this as an error.

#### 111563 **SEE ALSO**

111564 [Chapter 2](#) (on page 2472), *cp*, *ed*, *getopts*, *ls*, *printf*

111565 [XBD Section 3.145](#) (on page 52), [Chapter 5](#) (on page 113), [Chapter 8](#) (on page 167), [Section 12.2](#)  
 111566 (on page 215), [<cpio.h>](#), [<tar.h>](#)

111567 XSH *chown()*, *creat()*, *fstatat()*, *futimens()*, *mkdir()*, *mkfifo()*, *write()*

#### 111568 **CHANGE HISTORY**

111569 First released in Issue 4.

#### 111570 **Issue 5**

111571 A note is added to the APPLICATION USAGE indicating that the **cpio** and **tar** formats can only  
 111572 support files up to 8 gigabytes in size.

#### 111573 **Issue 6**

111574 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- 111575 • Support has been added for symbolic links in the options and interchange formats.
- 111576 • A new format has been devised, based on extensions to **ustar**.
- 111577 • References to the “extended” **tar** and **cpio** formats derived from the POSIX.1-1990  
 111578 standard have been changed to remove the “extended” adjective because this could cause  
 111579 confusion with the extended **tar** header added in this version. (All references to **tar** are  
 111580 actually to **ustar**.)

111581 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

111582 IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir()* and *mkfifo()* calls can  
 111583 ignore an [EEXIST] error when extracting an archive.

111584 IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when  
 111585 in **read** mode.

111586 IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **-t** option.

111587 IEEE PASC Interpretation 1003.2 #195 is applied.

111588 IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **-H**, **-L**,  
 111589 and **-I** options.

- 111590 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/35 is applied, adding the process ID of  
 111591 the *pax* process into certain fields. This change provides a method for the implementation to  
 111592 ensure that different instances of *pax* extracting a file named */a/b/foo* will not collide when  
 111593 processing the extended header information associated with **foo**.
- 111594 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/36 is applied, changing `-x B` to `-x pax` in  
 111595 the OPTIONS section.
- 111596 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/20 is applied, updating the SYNOPSIS to  
 111597 be consistent with the normative text.
- 111598 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/21 is applied, updating the  
 111599 DESCRIPTION to describe the behavior when files to be linked are symbolic links and the  
 111600 system is not capable of making hard links to symbolic links.
- 111601 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/22 is applied, updating the OPTIONS  
 111602 section to describe the behavior for how multiple `-odelete=pattern` options are to be handled.
- 111603 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/23 is applied, updating the **write** option  
 111604 within the OPTIONS section.
- 111605 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/24 is applied, adding a paragraph into  
 111606 the OPTIONS section that states that specifying more than one of the mutually-exclusive options  
 111607 (`-H` and `-L`) is not considered an error and that the last option specified will determine the  
 111608 behavior of the utility.
- 111609 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/25 is applied, removing the *ctime*  
 111610 paragraph within the EXTENDED DESCRIPTION. There is a contradiction in the definition of  
 111611 the *ctime* keyword for the *pax* extended header, in that the *st\_ctime* member of the **stat**  
 111612 structure does not refer to a file creation time. No field in the standard **stat** structure from `<sys/stat.h>`  
 111613 includes a file creation time.
- 111614 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/26 is applied, making it clear that *typeflag*  
 111615 1 (**ustar** Interchange Format) applies not only to files that are hard-linked, but also to files that  
 111616 are aliased via symbolic links.
- 111617 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/27 is applied, clarifying the *cpio c\_nlink*  
 111618 field.
- 111619 **Issue 7**
- 111620 Austin Group Interpretations 1003.1-2001 #011, #036, #086, and #109 are applied.
- 111621 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
 111622 *LC\_MESSAGES* environment variable.
- 111623 SD5-XCU-ERN-2 is applied, making `-c` and `-n` mutually-exclusive in the SYNOPSIS.
- 111624 SD5-XCU-ERN-3 is applied, revising the default behavior of `-H` and `-L`.
- 111625 SD5-XCU-ERN-5, SD5-XCU-ERN-6, SD5-XCU-ERN-7, SD5-XCU-ERN-60 are applied.
- 111626 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 111627 The *pax* utility is no longer allowed to create separate identical symbolic links when extracting  
 111628 linked symbolic links from an archive.
- 111629 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0128 [260], XCU/TC1-2008/0129  
 111630 [261], XCU/TC1-2008/0130 [261], XCU/TC1-2008/0131 [313], and XCU/TC1-2008/0132 [233]  
 111631 are applied.
- 111632 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0152 [886], XCU/TC2-2008/0153



- 111633 [814], XCU/TC2-2008/0154 [886], and XCU/TC2-2008/0155 [707] are applied.
- 111634 **Issue 8**
- 111635 Austin Group Defect 251 is applied, encouraging implementations to behave as follows:
- 111636 a. Report an error if a utility is directed to display a pathname that contains any bytes that  
111637 have the encoded value of a <newline> character when <newline> is a terminator or  
111638 separator in the output format being used.
  - 111639 b. Disallow the creation of filenames containing any bytes that have the encoded value of a  
111640 <newline> character.
- 111641 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 111642 Austin Group Defect 1133 is applied, clarifying the *-X* option and adding a paragraph to the  
111643 APPLICATION USAGE section.
- 111644 Austin Group Defect 1270 is applied, removing the *-n* option from the copy mode SYNOPSIS  
111645 line.
- 111646 Austin Group Defect 1278 is applied, removing mention of the *-n* option in connection with  
111647 write mode.
- 111648 Austin Group Defect 1330 is applied, removing obsolescent interfaces.
- 111649 Austin Group Defect 1331 is applied, changing ```st_atime``` to ```st_atim``` and ```st_mtime``` to  
111650 ```st_mtim```.
- 111651 Austin Group Defect 1379 is applied, changing the ENVIRONMENT VARIABLES section.
- 111652 Austin Group Defect 1380 is applied, changing text using the term ```link``` in line with its  
111653 updated definition and changing the description of the *-u* option.
- 111654 Austin Group Defect 1618 is applied, adding optional trailing 's' and 'S' characters to the  
111655 option-argument of the *-s* option.

111656 **NAME**

111657 pr — print files

111658 **SYNOPSIS**

```
111659 XSI pr [+page] [-column] [-adfFmprt] [-e[char][gap]] [-h header]
111660 [-i[char][gap]] [-l lines] [-n[char][width]] [-o offset] [-s[char]]
111661 [-w width] [file...]
```

111662 **DESCRIPTION**

111663 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be  
 111664 read, formatted, and written to standard output. By default, the input shall be separated into  
 111665 66-line pages, each with:

- 111666 • A 5-line header that includes the page number, date, time, and the pathname of the file
- 111667 • A 5-line trailer consisting of blank lines

111668 If standard output is associated with a terminal, diagnostic messages shall be deferred until the  
 111669 *pr* utility has completed processing.

111670 When options specifying multi-column output are specified, output text columns shall be of  
 111671 equal width; input lines that do not fit into a text column shall be truncated. By default, text  
 111672 columns shall be separated with at least one <blank>.

111673 **OPTIONS**

111674 The *pr* utility shall conform to XBD [Section 12.2](#) (on page 215), except that: the *page* option has a  
 111675 '+' delimiter; *page* and *column* can be multi-digit numbers; some of the option-arguments are  
 111676 optional; and some of the option-arguments cannot be specified as separate arguments from the  
 111677 preceding option letter. In particular, the *-s* option does not allow the option letter to be  
 111678 separated from its argument, and the options *-e*, *-i*, and *-n* require that both arguments, if  
 111679 present, not be separated from the option letter.

111680 The following options shall be supported. In the following option descriptions, *column*, *lines*,  
 111681 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

- 111682 **+page** Begin output at page number *page* of the formatted input.
- 111683 **-column** Produce multi-column output that is arranged in *column* columns (the default shall  
 111684 be 1) and is written down each column in the order in which the text is received  
 111685 from the input file. This option should not be used with *-m*. The options *-e* and *-i*  
 111686 shall be assumed for multiple text-column output. Whether or not text columns are  
 111687 produced with identical vertical lengths is unspecified, but a text column shall  
 111688 never exceed the length of the page (see the *-l* option). When used with *-t*, use the  
 111689 minimum number of lines to write the output.
- 111690 **-a** Modify the effect of the *-column* option so that the columns are filled across the  
 111691 page in a round-robin order (for example, when *column* is 2, the first input line  
 111692 heads column 1, the second heads column 2, the third is the second line in column  
 111693 1, and so on).
- 111694 **-d** Produce output that is double-spaced; append an extra <newline> following every  
 111695 <newline> found in the input.
- 111696 **-e[char][gap]**  
 111697 Expand each input <tab> to the next greater column position specified by the  
 111698 formula  $n*gap+1$ , where *n* is an integer > 0. If *gap* is zero or is omitted, it shall  
 111699 default to 8. All <tab> characters in the input shall be expanded into the  
 111700 appropriate number of <space> characters. If any non-digit character, *char*, is  
 111701 specified, it shall be used as the input <tab>. If the first character of the *-e* option-

- 111702 argument is a digit, the entire option-argument shall be assumed to be *gap*.
- 111703 XSI **-f** Use a <form-feed> for new pages, instead of the default behavior that uses a  
111704 sequence of <newline> characters. Pause before beginning the first page if the  
111705 standard output is associated with a terminal.
- 111706 **-F** Use a <form-feed> for new pages, instead of the default behavior that uses a  
111707 sequence of <newline> characters.
- 111708 **-h header** Use the string *header* to replace the contents of the *file* operand in the page header.
- 111709 **-i[*char*][*gap*]** In output, replace <space> characters with <tab> characters wherever one or more  
111710 adjacent <space> characters reach column positions  $gap+1$ ,  $2 * gap+1$ ,  $3 * gap+1$ , and  
111711 so on. If *gap* is zero or is omitted, default tab settings at every eighth column  
111712 position shall be assumed. If any non-digit character, *char*, is specified, it shall be  
111713 used as the output <tab>. If the first character of the **-i** option-argument is a digit,  
111714 the entire option-argument shall be assumed to be *gap*.
- 111715 **-l lines** Override the 66-line default and reset the page length to *lines*. If *lines* is not greater  
111716 than the sum of both the header and trailer depths (in lines), the *pr* utility shall  
111717 suppress both the header and trailer, as if the **-t** option were in effect.
- 111718 **-m** Merge files. Standard output shall be formatted so the *pr* utility writes one line  
111719 from each file specified by a *file* operand, side by side into text columns of equal  
111720 fixed widths, in terms of the number of column positions. Implementations shall  
111721 support merging of at least nine *file* operands.
- 111722 **-n[*char*][*width*]**  
111723 Provide *width*-digit line numbering (default for *width* shall be 5). The number shall  
111724 occupy the first *width* column positions of each text column of default output or  
111725 each line of **-m** output. If *char* (any non-digit character) is given, it shall be  
111726 appended to the line number to separate it from whatever follows (default for *char*  
111727 is a <tab>).
- 111728 **-o offset** Each line of output shall be preceded by offset <space> characters. If the **-o** option  
111729 is not specified, the default offset shall be zero. The space taken is in addition to the  
111730 output line width (see the **-w** option below).
- 111731 **-p** Pause before beginning each page if the standard output is directed to a terminal;  
111732 *pr* shall write an <alert> to standard error and wait for a <newline> to be read on  
111733 **/dev/tty**.
- 111734 **-r** Write no diagnostic reports on failure to open files.
- 111735 **-s[*char*]** Separate text columns by the single character *char* instead of by the appropriate  
111736 number of <space> characters (default for *char* shall be <tab>).
- 111737 **-t** Write neither the five-line identifying header nor the five-line trailer usually  
111738 supplied for each page. Quit writing after the last line of each file without spacing  
111739 to the end of the page.
- 111740 **-w width** Set the width of the line to *width* column positions for multiple text-column output  
111741 only. If the **-w** option is not specified and the **-s** option is not specified, the default  
111742 width shall be 72. If the **-w** option is not specified and the **-s** option is specified,  
111743 the default width shall be 512.
- 111744 For single column output, input lines shall not be truncated.

111745 **OPERANDS**

111746 The following operand shall be supported:

111747 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*  
 111748 operand is '-', the standard input shall be used.

111749 **STDIN**

111750 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
 111751 See the INPUT FILES section.

111752 **INPUT FILES**

111753 The input files shall be text files. If the **-m** option is not specified, an empty input file may, but  
 111754 should not, be treated as an error.

111755 The file **/dev/tty** shall be used to read responses required by the **-p** option.

111756 **ENVIRONMENT VARIABLES**

111757 The following environment variables shall affect the execution of *pr*:

111758 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 111759 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables  
 111760 used to determine the values of locale categories.)

111761 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 111762 internationalization variables.

111763 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 111764 characters (for example, single-byte as opposed to multi-byte characters in  
 111765 arguments and input files) and which characters are defined as printable (character  
 111766 class **print**). Non-printable characters are still written to standard output, but are  
 111767 not counted for the purpose for column-width and line-length calculations.

111768 *LC\_MESSAGES*

111769 Determine the locale that should be used to affect the format and contents of  
 111770 diagnostic messages written to standard error.

111771 *LC\_TIME* Determine the format of the date and time for use in writing header lines.

111772 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

111773 *TZ* Determine the timezone used to calculate date and time strings written in header  
 111774 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

111775 **ASYNCHRONOUS EVENTS**

111776 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error  
 111777 messages to the screen before terminating.

111778 **STDOUT**

111779 If the **-m** option is not specified, the *pr* utility output shall be as follows:

- 111780 • If an input file is empty and the implementation does not treat this as an error, no output  
 111781 shall be written for that file and this shall be considered to be successful completion of the  
 111782 processing for that file.

- 111783 • For each non-empty input file, the output shall be a paginated version of the original file.

111784 If the **-m** option is specified, the *pr* utility output shall be a paginated version of the merged file  
 111785 contents, as described in OPTIONS.

111786 In both cases, the pagination shall be accomplished using either <form-feed> characters or a

111787 XSI sequence of <newline> characters, as controlled by the **-F** or **-f** option. Page headers shall be  
 111788 generated unless the **-t** option is specified, the **-l** option is specified with too small a value (see  
 111789 OPTIONS), or the **-m** option is specified and all of the input files are empty. The page headers  
 111790 shall be of the form:

111791 "\n\n%s %s Page %d\n\n\n", <output of date>, <file>, <page number>

111792 In the POSIX locale, the <output of date> field shall be equivalent to the output of the following  
 111793 command:

111794 date "+%b %e %H:%M %Y"

111795 without the trailing <newline>, as it would appear if executed at the current time if the **-m**  
 111796 option is specified, or at the following time otherwise:

- 111797 • The current time on pages being written from standard input.
- 111798 • The modification time of the file named by the corresponding *file* operand on pages not  
 111799 being written from standard input.

111800 When the *LC\_TIME* locale category is not set to the POSIX locale, a different format and order of  
 111801 presentation of this field may be used.

111802 If the **-h** option is specified, the <file> field shall be replaced by the *header* argument. Otherwise:

- 111803 • If the **-m** option is specified, the <file> field shall be replaced by a null string on all pages.
- 111804 • If the **-m** option is not specified, the <file> field shall be replaced by a null string on pages  
 111805 containing output that was read from standard input.

#### 111806 **STDERR**

111807 The standard error shall be used for diagnostic messages and for alerting the terminal when **-p**  
 111808 is specified.

#### 111809 **OUTPUT FILES**

111810 None.

#### 111811 **EXTENDED DESCRIPTION**

111812 None.

#### 111813 **EXIT STATUS**

111814 The following exit values shall be returned:

- 111815 0 Successful completion.
- 111816 >0 An error occurred.

#### 111817 **CONSEQUENCES OF ERRORS**

111818 Default.

#### 111819 **APPLICATION USAGE**

111820 A conforming application must protect its first operand, if it starts with a <plus-sign>, by  
 111821 preceding it with the "--" argument that denotes the end of the options. For example, *pr+x*  
 111822 could be interpreted as an invalid page number or a *file* operand.

111823 If a *file* operand contains <newline>, <form-feed>, or <vertical-tab> characters, or is overly long,  
 111824 and the *pr* utility is instructed to include the pathname of that file in the header, pagination may  
 111825 not be handled correctly. Applications can guard against this by using the **-h** option (for  
 111826 example, passing a sanitized, truncated form of the pathname with **-h**).

111827 **EXAMPLES**

- 111828           1. Print a numbered list of all files in the current directory:
- 111829                 `ls -a | pr -n -h "Files in $(pwd) ."`
- 111830           2. Print **file1** and **file2** as a double-spaced, three-column listing headed by ```file list```:
- 111831                 `pr -3d -h "file list" file1 file2`
- 111832           3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:
- 111833                 `pr -e9 -t <file1 >file2`

111834 **RATIONALE**

111835           This utility is one of those that does not follow the Utility Syntax Guidelines because of its  
 111836           historical origins. The standard developers could have added new options that obeyed the  
 111837           guidelines (and marked the old options obsolescent) or devised an entirely new utility; there are  
 111838           examples of both actions in this volume of POSIX.1-2024. Because of its widespread use by  
 111839           historical applications, the standard developers decided to exempt this version of *pr* from many  
 111840           of the guidelines.

111841           Implementations are required to accept option-arguments to the `-h`, `-l`, `-o`, and `-w` options  
 111842           whether presented as part of the same argument or as a separate argument to *pr*, as suggested by  
 111843           the Utility Syntax Guidelines. The `-n` and `-s` options, however, are specified as in historical  
 111844           practice because they are frequently specified without their optional arguments. If a `<blank>`  
 111845           were allowed before the option-argument in these cases, a *file* operand could mistakenly be  
 111846           interpreted as an option-argument in historical applications.

111847           The text about the minimum number of lines in multi-column output was included to ensure  
 111848           that a best effort is made in balancing the length of the columns. There are known historical  
 111849           implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines  
 111850           and a second of 4. Although this is not a problem when a full page with headers and trailers is  
 111851           produced, it would be relatively useless when used with `-t`.

111852           Historical implementations of the *pr* utility have differed in the action taken for the `-f` option.  
 111853           BSD uses it as described here for the `-F` option; System V uses it to change trailing `<newline>`  
 111854           characters on each page to a `<form-feed>` and, if standard output is a TTY device, sends an  
 111855           `<alert>` to standard error and reads a line from `/dev/tty` before the first page. There were strong  
 111856           arguments from both sides of this issue concerning historical practice and as a result the `-F`  
 111857           option was added. XSI-conformant systems support the System V historical actions for the `-f`  
 111858           option.

111859           The `<output of date>` field in the `-l` format is specified only for the POSIX locale. As noted, the  
 111860           format can be different in other locales. No mechanism for defining this is present in this volume  
 111861           of POSIX.1-2024, as the appropriate vehicle is a message catalog; that is, the format should be  
 111862           specified as a ```message```.

111863           Some implementations of *pr* treat an empty file as an error when `-m` is not specified, but not  
 111864           when `-m` is specified (even if there is only one input file). Implementations are encouraged to  
 111865           eliminate this inconsistency by never treating an empty file as an error.

111866 **FUTURE DIRECTIONS**

111867           A future version of this standard may require that an empty file is never treated as an error.

111868 **SEE ALSO**111869 *expand, lp*

111870 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

111871 **CHANGE HISTORY**

111872 First released in Issue 2.

111873 **Issue 6**111874 The following new requirements on POSIX implementations derive from alignment with the  
111875 Single UNIX Specification:

- 111876
- The `-p` option is added.

111877 The normative text is reworded to avoid use of the term “must” for application requirements.

111878 **Issue 7**

111879 PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied.

111880 Austin Group Interpretation 1003.1-2001 #093 is applied.

111881 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

111882 **Issue 8**111883 Austin Group Defect 251 is applied, adding a paragraph about problematic pathnames to the  
111884 APPLICATION USAGE section.111885 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.111886 Austin Group Defect 1433 is applied, changing `<carriage-return>` to `<newline>` in the  
111887 description of the `-p` option.111888 Austin Group Defect 1434 is applied, combining the two option groups in the SYNOPSIS into  
111889 one.111890 Austin Group Defect 1590 is applied, clarifying the requirements when an input file is empty  
111891 and changing the STDOUT section.

111892 **NAME**

111893 printf — write formatted output

111894 **SYNOPSIS**111895 `printf format [argument...]`111896 **DESCRIPTION**111897 The *printf* utility shall write formatted operands to the standard output. The *argument* operands  
111898 shall be formatted under control of the *format* operand.111899 **OPTIONS**

111900 None.

111901 **OPERANDS**

111902 The following operands shall be supported:

111903 *format* A character string describing the format to use to write the remaining operands.  
111904 See the EXTENDED DESCRIPTION section.111905 *argument* The values to be written to standard output, under the control of *format*. See the  
111906 EXTENDED DESCRIPTION section.111907 **STDIN**

111908 Not used.

111909 **INPUT FILES**

111910 None.

111911 **ENVIRONMENT VARIABLES**111912 The following environment variables shall affect the execution of *printf*:111913 *LANG* Provide a default value for the internationalization variables that are unset or null.  
111914 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables  
111915 used to determine the values of locale categories.)111916 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
111917 internationalization variables.111918 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
111919 characters (for example, single-byte as opposed to multi-byte characters in  
111920 arguments).111921 *LC\_MESSAGES*111922 Determine the locale that should be used to affect the format and contents of  
111923 diagnostic messages written to standard error.111924 *LC\_NUMERIC*111925 Determine the locale for numeric formatting. It shall affect the format of numbers  
111926 written using the *e*, *E*, *f*, *g*, and *G* conversion specifier characters (if supported).111927 *XSHELL* *NLSPATH* Determine the location of messages objects and message catalogs.111928 **ASYNCHRONOUS EVENTS**

111929 Default.

111930 **STDOUT**

111931 See the EXTENDED DESCRIPTION section.



111932 **STDERR**

111933 The standard error shall be used only for diagnostic messages.

111934 **OUTPUT FILES**

111935 None.

111936 **EXTENDED DESCRIPTION**

111937 The application shall ensure that the *format* operand is a character string, beginning and ending  
 111938 in its initial shift state, if any. The *format* operand shall be used as the format string described in  
 111939 XBD Chapter 5 (on page 113) with the following exceptions:

- 111940 1. A <space> in the format string, in any context other than a flag of a conversion  
 111941 specification, shall be treated as an ordinary character that is copied to the output.
- 111942 2. A ' $\Delta$ ' character in the format string shall be treated as a ' $\Delta$ ' character, not as a <space>.
- 111943 3. In addition to the escape sequences shown in XBD Chapter 5 (on page 113) ('\\', '\a',  
 111944 '\b', '\f', '\n', '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit  
 111945 octal number, shall be written as a byte with the numeric value specified by the octal  
 111946 number.
- 111947 4. The implementation shall not precede or follow output from the *d* or *u* conversion  
 111948 specifiers with <blank> characters not specified by the *format* operand.
- 111949 5. The implementation shall not precede output from the *o* conversion specifier with zeros  
 111950 not specified by the *format* operand.
- 111951 6. The *a*, *A*, *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers need not be supported.
- 111952 7. An additional conversion specifier character, *b*, shall be supported as follows. The  
 111953 argument shall be taken to be a string that can contain <backslash>-escape sequences.  
 111954 The following <backslash>-escape sequences shall be supported:
  - 111955 — The escape sequences listed in XBD Chapter 5 (on page 113) ('\\', '\a', '\b',  
 111956 '\f', '\n', '\r', '\t', '\v'), which shall be converted to the characters they  
 111957 represent.
  - 111958 — "\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be  
 111959 converted to a byte with the numeric value specified by the octal number.
  - 111960 — '\c', which shall not be written and shall cause *printf* to ignore any remaining  
 111961 characters in the string operand containing it, any remaining string operands, and  
 111962 any additional characters in the *format* operand. If a precision is specified and the  
 111963 argument contains a '\c' after the point at which the number of bytes indicated by  
 111964 the precision specification have been written, it is unspecified whether the '\c'  
 111965 takes effect.
- 111966 The interpretation of a <backslash> followed by any other sequence of characters is  
 111967 unspecified.
- 111968 Bytes from the converted string shall be written until the end of the string or the number  
 111969 of bytes indicated by the precision specification is reached. If the precision is omitted, it  
 111970 shall be taken to be infinite, so all bytes up to the end of the converted string shall be  
 111971 written.
- 111972 8. Conversions can be applied to the *n*th *argument* operand rather than to the next *argument*  
 111973 operand. In this case, the conversion specifier character '%' is replaced by the sequence  
 111974 "%n\$", where *n* is a decimal integer in the range [1,{NL\_ARGMAX}], giving the *argument*  
 111975 operand number. This feature provides for the definition of format strings that select

- 111976 arguments in an order appropriate to specific languages.
- 111977 The format can contain either numbered argument conversion specifications (that is, ones  
111978 beginning with "%n\$"), or unnumbered argument conversion specifications, but not  
111979 both. The only exception to this is that "%%" can be mixed with the "%n\$" form. The  
111980 results of mixing numbered and unnumbered argument specifications that consume an  
111981 argument are unspecified.
- 111982 9. For each conversion specification that consumes an argument, an *argument* operand shall  
111983 be evaluated and converted to the appropriate type for the conversion as specified below.  
111984 The operand to be evaluated shall be determined as follows:
- 111985 • If the conversion specification begins with a "%n\$" sequence, the *nth argument*  
111986 operand shall be evaluated.
  - 111987 • Otherwise, the evaluated operand shall be the next *argument* operand after the one  
111988 evaluated by the previous conversion specification that consumed an argument; if  
111989 there is no such previous conversion specification the first *argument* operand shall  
111990 be evaluated.
- 111991 If the *format* operand contains no conversion specifications that consume an argument  
111992 and there are *argument* operands present, the results are unspecified.
- 111993 10. The *format* operand shall be reused as often as necessary to satisfy the *argument* operands.  
111994 If conversion specifications beginning with a "%n\$" sequence are used, on format reuse  
111995 the value of *n* shall refer to the *nth argument* operand following the highest numbered  
111996 *argument* operand consumed by the previous use of the *format* operand.
- 111997 11. If an *argument* operand to be consumed by a conversion specification does not exist:
- 111998 • If it is a numbered argument conversion specification, *printf* should write a  
111999 diagnostic message to standard error and exit with non-zero status, but may behave  
112000 as for an unnumbered argument conversion specification.
  - 112001 • If it is an unnumbered argument conversion specification, any extra *b*, *c*, or *s*  
112002 conversion specifiers shall be evaluated as if a null string argument were supplied  
112003 and any other extra conversion specifiers shall be evaluated as if a zero argument  
112004 were supplied.
- 112005 12. If a character sequence in the *format* operand begins with a '%' character, but does not  
112006 form a valid conversion specification, the behavior is unspecified.
- 112007 13. The argument to the *c* conversion specifier can be a string containing zero or more bytes.  
112008 If it contains one or more bytes, the first byte shall be written and any additional bytes  
112009 shall be ignored. If the argument is an empty string, it is unspecified whether nothing is  
112010 written or a null byte is written.
- 112011 The *argument* operands shall be treated as strings if the corresponding conversion specifier is *b*,  
112012 *c*, or *s*, and shall be evaluated as if by the *strtod()* function if the corresponding conversion  
112013 specifier is *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G*. Otherwise, they shall be evaluated as unsuffixed C integer  
112014 constants, as described by the ISO C standard, with the following extensions:
- 112015 • A leading <plus-sign> or <hyphen-minus> shall be allowed.
  - 112016 • If the leading character is a single-quote or double-quote, the value shall be the numeric  
112017 value in the underlying codeset of the character following the single-quote or double-  
112018 quote.

- 112019           • Suffixed integer constants may be allowed.
- 112020           If an *argument* operand cannot be completely converted into an internal value appropriate to the  
112021           corresponding conversion specification, a diagnostic message shall be written to standard error  
112022           and the utility shall not exit with a zero exit status, but shall continue processing any remaining  
112023           operands and shall write the value accumulated at the time the error was detected to standard  
112024           output.
- 112025           It shall not be considered an error if an *argument* operand is not completely used for a *b*, *c*, or *s*  
112026           conversion.
- 112027 **EXIT STATUS**
- 112028           The following exit values shall be returned:
- 112029           0 Successful completion.
  - 112030           >0 An error occurred.
- 112031 **CONSEQUENCES OF ERRORS**
- 112032           Default.
- 112033 **APPLICATION USAGE**
- 112034           The floating-point formatting conversion specifications of *printf()* are not required because all  
112035           arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations  
112036           and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-  
112037           point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility  
112038           cannot really be used to format *bc* output; it does not support arbitrary precision.)  
112039           Implementations are encouraged to support the floating-point conversions as an extension.
- 112040           Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of  
112041           POSIX.1-2024 on which it is based, makes no special provision for dealing with multi-byte  
112042           characters when using the *%c* conversion specification or when a precision is specified in a *%b* or  
112043           *%s* conversion specification. Applications should be extremely cautious using either of these  
112044           features when there are multi-byte characters in the character set.
- 112045           No provision is made in this volume of POSIX.1-2024 which allows field widths and precisions  
112046           to be specified as '\*' since the '\*' can be replaced directly in the *format* operand using shell  
112047           variable substitution. Implementations can also provide this feature as an extension if they so  
112048           choose.
- 112049           Hexadecimal character constants as defined in the ISO C standard are not recognized in the  
112050           *format* operand because there is no consistent way to detect the end of the constant. Octal  
112051           character constants are limited to, at most, three octal digits, but hexadecimal character constants  
112052           are only terminated by a non-hex-digit character. In the ISO C standard, string literal  
112053           concatenation can be used to terminate a constant and follow it with a hexadecimal character to  
112054           be written. In the shell, similar concatenation can be done using '\$'...' so that the shell  
112055           converts the hexadecimal sequence before it executes *printf*.
- 112056           The *%b* conversion specification is not part of the ISO C standard; it has been added here as a  
112057           portable way to process <backslash>-escapes expanded in string operands as provided by the  
112058           *echo* utility. See also the APPLICATION USAGE section of *echo* (on page 2811) for ways to use  
112059           *printf* as a replacement for all of the traditional versions of the *echo* utility.
- 112060           If an argument cannot be parsed correctly for the corresponding conversion specification, the  
112061           *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end  
112062           of an argument being used for a numeric conversion shall be reported as errors.
- 112063           Unlike the *printf()* function, when numbered conversion specifications are used, specifying the

112064 Nth argument does not require that all the leading arguments, from the first to the (N-1)th, are  
 112065 specified in the format string. For example, "%3\$s %1\$d\n" is an acceptable *format* operand  
 112066 which evaluates the first and third *argument* operands but not the second.

### 112067 EXAMPLES

112068 To alert the user and then print and read a series of prompts:

```
112069 printf "\aPlease fill in the following: \nName: "  
112070 read name  
112071 printf "Phone number: "  
112072 read phone
```

112073 To read out a list of right and wrong answers from a file, calculate the percentage correctly, and  
 112074 print them out. The numbers are right-justified and separated by a single <tab>. The percentage  
 112075 is written to one decimal place of accuracy:

```
112076 while read right wrong ; do  
112077     percent=$(echo "scale=1; ($right*100)/($right+$wrong)" | bc)  
112078     printf "%2d right\t%2d wrong\t(%s%%)\n" \  
112079         $right $wrong $percent  
112080 done < database_file
```

112081 The command:

```
112082 printf "%5d%4d\n" 1 21 321 4321 54321
```

112083 produces:

```
112084     1  21  
112085     3214321  
112086 54321  0
```

112087 Note that the *format* operand is used three times to print all of the given strings and that a '0'  
 112088 was supplied by *printf* to satisfy the last %4d conversion specification.

112089 The command:

```
112090 printf '%d\n' 10 010 0x10
```

112091 produces:

| Output Line | Explanation                                                                      |
|-------------|----------------------------------------------------------------------------------|
| 10          | Decimal representation of the numeric value of decimal integer constant 10       |
| 8           | Decimal representation of the numeric value of octal integer constant 010        |
| 16          | Decimal representation of the numeric value of hexadecimal integer constant 0x10 |

112098 If the implementation supports floating-point conversions, the command:

```
112099 LC_ALL=C printf "%.2f\n" 10 010 0x10 10.1e2 010.1e2 0x10.1p2
```

112100 produces:

| Output Line                          | Explanation                                                                                                           |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| 112101<br>112102<br>112103<br>112104 | 10.00<br>The string "10" interpreted as a decimal value, with 2 digits of precision                                   |
| 112105<br>112106                     | 10.00<br>The string "010" interpreted as a decimal (not octal) value, with 2 digits of precision                      |
| 112107<br>112108                     | 16.00<br>The string "0x10" interpreted as a hexadecimal value, with 2 digits of precision                             |
| 112109<br>112110                     | 1010.00<br>The string "10.1e2" interpreted as a decimal floating-point value, with 2 digits of precision              |
| 112111<br>112112                     | 1010.00<br>The string "010.1e2" interpreted as a decimal (not octal) floating-point value, with 2 digits of precision |
|                                      | 64.25<br>The string "0x10.1p2" interpreted as a hexadecimal floating-point value, with 2 digits of precision          |

112113 The *printf* utility is required to notify the user when conversion errors are detected while  
 112114 producing numeric output; thus, the following results would be expected on an implementation  
 112115 with 32-bit two's-complement integers when %d is specified as the *format* operand:

| Argument         | Standard Output | Diagnostic Output                                |
|------------------|-----------------|--------------------------------------------------|
| 112116<br>112117 | 5a              | 5                                                |
| 112118           | 9999999999      | <b>printf: "5a" not completely converted</b>     |
| 112119           | -9999999999     | <b>printf: "9999999999" arithmetic overflow</b>  |
| 112120           | ABC             | <b>printf: "-9999999999" arithmetic overflow</b> |
| 112121           |                 | <b>printf: "ABC" expected numeric value</b>      |

112122 The diagnostic message format is not specified, but these examples convey the type of  
 112123 information that should be reported. Note that the value shown on standard output is what  
 112124 would be expected as the return value from the *strtol()* function as defined in the System  
 112125 Interfaces volume of POSIX.1-2024. A similar correspondence exists between %u and *strtoul()*  
 112126 and %e, %f, and %g (if the implementation supports floating-point conversions) and *strtod()*.

112127 In a locale that uses a codeset based on the ISO/IEC 646:1991 standard, the command:

```
112128 printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"
```

112129 produces:

| Output Line | Explanation |
|-------------|-------------|
| 112130      |             |
| 112131      | 3           |
| 112132      | 3           |
| 112133      | -3          |
| 112134      | 51          |
| 112135      | 43          |
| 112136      | 43          |
| 112137      | 45          |
| 112138      | 45          |
| 112139      |             |

112140 Since the last two arguments contain more characters than used for the conversion, a diagnostic  
 112141 message is generated and the exit status is non-zero. Note that in a locale with multi-byte  
 112142 characters, the value of a character is intended to be the value of the equivalent of the `wchar_t`  
 112143 representation of the character as described in the System Interfaces volume of POSIX.1-2024.

#### 112144 RATIONALE

112145 The *printf* utility was added to provide functionality that has historically been provided by *echo*.  
 112146 However, due to irreconcilable differences in the various versions of *echo* extant, the version has  
 112147 few special features, leaving those to this new *printf* utility, which is based on one in the Ninth  
 112148 Edition system.

112149 The format strings for the *printf* utility are handled differently than for the *printf()* function in  
 112150 several respects. Notable differences include:

- 112151 • Reuse of the format until all arguments have been consumed.
- 112152 • No provision for obtaining field width and precision from argument values.
- 112153 • No requirement to support floating-point conversion specifiers.
- 112154 • An additional `b` conversion specifier.
- 112155 • Special handling of leading single-quote or double-quote for integer conversion specifiers.
- 112156 • Hexadecimal character constants are not recognized in the format.
- 112157 • Formats that use numbered argument conversion specifications can have gaps in the  
 112158 argument numbers.

112159 Although *printf* implementations have no difficulty handling formats with mixed numbered and  
 112160 unnumbered conversion specifications (unlike the *printf()* function where it is undefined  
 112161 behavior), existing implementations differ in behavior. Given the format operand  
 112162 "`%2$d %d\n`", with some implementations the "`%d`" evaluates the first argument and with  
 112163 others it evaluates the third. Consequently this standard leaves the behavior unspecified (as  
 112164 opposed to undefined).

112165 Earlier versions of this standard specified that arguments for all conversions other than `b`, `c`, and  
 112166 `s` were evaluated in the same way (as C constants, but with stated exceptions). For  
 112167 implementations supporting the floating-point conversions it was not clear whether integer  
 112168 conversions need only accept integer constants and floating-point conversions need only accept  
 112169 floating-point constants, or whether both types of conversions should accept both types of  
 112170 constants. Also by not distinguishing between them, the requirement relating to a leading  
 112171 single-quote or double-quote applied to floating-point conversions even though this provided  
 112172 no useful functionality to applications that was not already available through the integer  
 112173 conversions. The current standard clarifies the situation by specifying that the arguments for  
 112174 floating-point conversions are evaluated as if by *strtod()*, and the arguments for integer  
 112175 conversions are evaluated as C integer constants, with the special treatment of leading single-

112176 quote and double-quote applying only to integer conversions.

#### 112177 FUTURE DIRECTIONS

112178 A future version of this standard may require that a missing *argument* operand to be consumed  
112179 by a numbered argument conversion specification is treated as an error.

112180 A future version of this standard is expected to add a %b conversion to the *printf()* function for  
112181 binary conversion of integers, in alignment with the next version of the ISO C standard. This  
112182 will result in an inconsistency between the *printf* utility and *printf()* function for format strings  
112183 containing %b. Implementors are encouraged to collaborate on a way to address this which  
112184 could then be adopted in a future version of this standard. For example, the *printf* utility could  
112185 add a -C option to make the format string behave in the same way, as far as possible, as the  
112186 *printf()* function.

112187 A future version of this standard may add a %q conversion to convert a string argument to a  
112188 quoted output format that can be reused as shell input.

#### 112189 SEE ALSO

112190 *awk, bc, echo*

112191 XBD [Chapter 5](#) (on page 113), [Chapter 8](#) (on page 167)

112192 XSH *fprintf(), strtod()*

#### 112193 CHANGE HISTORY

112194 First released in Issue 4.

#### 112195 Issue 7

112196 Austin Group Interpretations 1003.1-2001 #175 and #177 are applied.

112197 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112198 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0156 [727], XCU/TC2-2008/0157  
112199 [727,932], XCU/TC2-2008/0158 [584], and XCU/TC2-2008/0159 [727] are applied.

#### 112200 Issue 8

112201 Austin Group Defect 1108 is applied, changing ``twos'' to ``two's''.

112202 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

112203 Austin Group Defect 1202 is applied, changing the description of how '\c' is handled by the b  
112204 conversion specifier.

112205 Austin Group Defects 1209 and 1476 are applied, changing the EXAMPLES section.

112206 Austin Group Defect 1413 is applied, changing the APPLICATION USAGE section.

112207 Austin Group Defect 1562 is applied, clarifying that it is the application's responsibility to  
112208 ensure that the format is a character string, beginning and ending in its initial shift state, if any.

112209 Austin Group Defect 1592 is applied, adding support for numbered conversion specifications.

112210 Austin Group Defect 1771 is applied, changing the FUTURE DIRECTIONS section.

112211 **NAME**112212 prs — print an SCCS file (**DEVELOPMENT**)112213 **SYNOPSIS**112214 XSI prs [-a] [-d *dataspec*] [-r[*SID*]] *file...*112215 prs [-e|-l] -c *cutoff* [-d *dataspec*] *file...*112216 prs [-e|-l] -r[*SID*] [-d *dataspec*] *file...*112217 **DESCRIPTION**112218 The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied  
112219 format.112220 **OPTIONS**112221 The *prs* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the `-r` option has an  
112222 optional option-argument. This optional option-argument cannot be presented as a separate  
112223 argument. The following options shall be supported:112224 `-d dataspec` Specify the output data specification. The *dataspec* shall be a string consisting of  
112225 SCCS file *data keywords* (see [Data Keywords](#), on page 3305) interspersed with  
112226 optional user-supplied text.112227 `-r[SID]` Specify the SCCS identification string (SID) of a delta for which information is  
112228 desired. If no *SID* option-argument is specified, the SID of the most recently  
112229 created delta shall be assumed.112230 `-e` Request information for all deltas created earlier than and including the delta  
112231 designated via the `-r` option or the date-time given by the `-c` option.112232 `-l` Request information for all deltas created later than and including the delta  
112233 designated via the `-r` option or the date-time given by the `-c` option.112234 `-c cutoff` Indicate the *cutoff* date-time, in the form:112235 `YY[MM[DD[HH[MM[SS]]]]]`112236 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999  
112237 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.112238 **Note:** It is expected that in a future version of this standard the default century inferred  
112239 from a 2-digit year will change. (This would apply to all commands accepting a  
112240 2-digit year as input.)112241 No changes (deltas) to the SCCS file that were created after the specified *cutoff*  
112242 date-time shall be included in the output. Units omitted from the date-time default  
112243 to their maximum possible values; for example, `-c 7502` is equivalent to  
112244 `-c 750228235959`.112245 `-a` Request writing of information for both removed—that is, *delta type=R* (see  
112246 [rmdel](#))—and existing—that is, *delta type=D<sub>r</sub>*—deltas. If the `-a` option is not  
112247 specified, information for existing deltas only shall be provided.112248 **OPERANDS**

112249 The following operand shall be supported:

112250 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs*  
112251 utility shall behave as though each file in the directory were specified as a named  
112252 file, except that non-SCCS files (last component of the pathname does not begin  
112253 with **s**.) and unreadable files shall be silently ignored.



112254 If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
 112255 each line of the standard input shall be taken to be the name of an SCCS file to be  
 112256 processed. Non-SCCS files and unreadable files shall be silently ignored.

#### 112257 STDIN

112258 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
 112259 line of the text file shall be interpreted as an SCCS pathname.

#### 112260 INPUT FILES

112261 Any SCCS files displayed are files of an unspecified format.

#### 112262 ENVIRONMENT VARIABLES

112263 The following environment variables shall affect the execution of *prs*:

112264 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 112265 (See XBD Section 8.2 (on page 169) the precedence of internationalization variables  
 112266 used to determine the values of locale categories.)

112267 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 112268 internationalization variables.

112269 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 112270 characters (for example, single-byte as opposed to multi-byte characters in  
 112271 arguments and input files).

112272 *LC\_MESSAGES*

112273 Determine the locale that should be used to affect the format and contents of  
 112274 diagnostic messages written to standard error.

112275 *NLSPATH* Determine the location of messages objects and message catalogs.

#### 112276 ASYNCHRONOUS EVENTS

112277 Default.

#### 112278 STDOUT

112279 The standard output shall be a text file whose format is dependent on the data keywords  
 112280 specified with the **-d** option.

#### 112281 Data Keywords

112282 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an  
 112283 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple  
 112284 times.

112285 The information written by *prs* shall consist of:

- 112286 1. The user-supplied text
- 112287 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data  
 112288 keywords in the order of appearance in the *dataspec*

112289 The format of a data keyword value shall either be simple ('S'), in which keyword substitution  
 112290 is direct, or multi-line ('M').

112291 User-supplied text shall be any text other than recognized data keywords. A <tab> shall be  
 112292 specified by '\t' and <newline> by '\n'. When the **-r** option is not specified, the default  
 112293 *dataspec* shall be:

112294 :PN::\n\n

112295 and the following *dataspec* shall be used for each selected delta:

112296 :Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:

112297

112298

112299

112300

112301

112302

112303

112304

112305

112306

112307

112308

112309

112310

112311

112312

112313

112314

112315

112316

112317

112318

112319

112320

112321

112322

112323

112324

112325

112326

112327

112328

112329

112330

112331

112332

112333

112334

112335

112336

112337

112338

112339

112340

112341

112342

112343

112344

| SCCS File Data Keywords |                                                          |              |                |        |
|-------------------------|----------------------------------------------------------|--------------|----------------|--------|
| Keyword                 | Data Item                                                | File Section | Value          | Format |
| :Dt:                    | Delta information                                        | Delta Table  | See below*     | S      |
| :DL:                    | Delta line statistics                                    | "            | :Li:/:Ld:/:Lu: | S      |
| :Li:                    | Lines inserted by Delta                                  | "            | nnnnn***       | S      |
| :Ld:                    | Lines deleted by Delta                                   | "            | nnnnn***       | S      |
| :Lu:                    | Lines unchanged by Delta                                 | "            | nnnnn***       | S      |
| :DT:                    | Delta type                                               | "            | D or R         | S      |
| :I:                     | SCCS ID string (SID)                                     | "            | See below**    | S      |
| :R:                     | Release number                                           | "            | nnnn           | S      |
| :L:                     | Level number                                             | "            | nnnn           | S      |
| :B:                     | Branch number                                            | "            | nnnn           | S      |
| :S:                     | Sequence number                                          | "            | nnnn           | S      |
| :D:                     | Date delta created                                       | "            | :Dy:/:Dm:/:Dd: | S      |
| :Dy:                    | Year delta created                                       | "            | nn             | S      |
| :Dm:                    | Month delta created                                      | "            | nn             | S      |
| :Dd:                    | Day delta created                                        | "            | nn             | S      |
| :T:                     | Time delta created                                       | "            | :Th:::Tm:::Ts: | S      |
| :Th:                    | Hour delta created                                       | "            | nn             | S      |
| :Tm:                    | Minutes delta created                                    | "            | nn             | S      |
| :Ts:                    | Seconds delta created                                    | "            | nn             | S      |
| :P:                     | Programmer who created Delta                             | "            | logname        | S      |
| :DS:                    | Delta sequence number                                    | "            | nnnn           | S      |
| :DP:                    | Predecessor Delta sequence number                        | "            | nnnn           | S      |
| :DI:                    | Sequence number of deltas included, excluded, or ignored | "            | :Dn:/:Dx:/:Dg: | S      |
| :Dn:                    | Deltas included (sequence #)                             | "            | :DS: :DS: ...  | S      |
| :Dx:                    | Deltas excluded (sequence #)                             | "            | :DS: :DS: ...  | S      |
| :Dg:                    | Deltas ignored (sequence #)                              | "            | :DS: :DS: ...  | S      |
| :MR:                    | MR numbers for delta                                     | "            | text           | M      |
| :C:                     | Comments for delta                                       | "            | text           | M      |
| :UN:                    | User names                                               | User Names   | text           | M      |
| :FL:                    | Flag list                                                | Flags        | text           | M      |
| :Y:                     | Module type flag                                         | "            | text           | S      |
| :MF:                    | MR validation flag                                       | "            | yes or no      | S      |
| :MP:                    | MR validation program name                               | "            | text           | S      |
| :KF:                    | Keyword error, warning flag                              | "            | yes or no      | S      |
| :BF:                    | Branch flag                                              | "            | yes or no      | S      |
| :J:                     | Joint edit flag                                          | "            | yes or no      | S      |
| :LK:                    | Locked releases                                          | "            | :R: ...        | S      |
| :Q:                     | User-defined keyword                                     | "            | text           | S      |
| :M:                     | Module name                                              | "            | text           | S      |
| :FB:                    | Floor boundary                                           | "            | :R:            | S      |
| :CB:                    | Ceiling boundary                                         | "            | :R:            | S      |
| :Ds:                    | Default SID                                              | "            | :I:            | S      |
| :ND:                    | Null delta flag                                          | "            | yes or no      | S      |
| :FD:                    | File descriptive text                                    | Comments     | text           | M      |

| SCCS File Data Keywords |                               |                                                                                                                                                           |       |                            |
|-------------------------|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-------|----------------------------|
| Keyword                 | Data Item                     | File Section                                                                                                                                              | Value | Format                     |
| 112345                  |                               |                                                                                                                                                           |       |                            |
| 112346                  |                               |                                                                                                                                                           |       |                            |
| 112347                  | <b>:BD:</b>                   | Body                                                                                                                                                      | Body  | <i>text</i> M              |
| 112348                  | <b>:GB:</b>                   | Gotten body                                                                                                                                               | "     | <i>text</i> M              |
| 112349                  | <b>:W:</b>                    | A form of <i>what</i> string                                                                                                                              | N/A   | <b>:Z::M:\t:I:</b> S       |
| 112350                  | <b>:A:</b>                    | A form of <i>what</i> string                                                                                                                              | N/A   | <b>:Z::Y: :M: :I::Z:</b> S |
| 112351                  | <b>:Z:</b>                    | <i>what</i> string delimiter                                                                                                                              | N/A   | @ (#) S                    |
| 112352                  | <b>:F:</b>                    | SCCS filename                                                                                                                                             | N/A   | <i>text</i> S              |
| 112353                  | <b>:PN:</b>                   | SCCS file pathname                                                                                                                                        | N/A   | <i>text</i> S              |
| 112354                  | *                             | <b>:Dt::DT: :I: :D: :T: :P: :DS: :DP:</b>                                                                                                                 |       |                            |
| 112355                  | **                            | <b>:R::L::B::S:</b> if the delta is a branch delta ( <b>:BF:==yes</b> )                                                                                   |       |                            |
| 112356                  |                               | <b>:R::L:</b> if the delta is not a branch delta ( <b>:BF:==no</b> )                                                                                      |       |                            |
| 112357                  | ***                           | The line statistics are capped at 99 999. For example, if 100 000 lines were unchanged in a certain revision, <b>:Lu:</b> shall produce the value 99 999. |       |                            |
| 112358                  |                               |                                                                                                                                                           |       |                            |
| 112359                  | <b>STDERR</b>                 |                                                                                                                                                           |       |                            |
| 112360                  |                               | The standard error shall be used only for diagnostic messages.                                                                                            |       |                            |
| 112361                  | <b>OUTPUT FILES</b>           |                                                                                                                                                           |       |                            |
| 112362                  |                               | None.                                                                                                                                                     |       |                            |
| 112363                  | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                           |       |                            |
| 112364                  |                               | None.                                                                                                                                                     |       |                            |
| 112365                  | <b>EXIT STATUS</b>            |                                                                                                                                                           |       |                            |
| 112366                  |                               | The following exit values shall be returned:                                                                                                              |       |                            |
| 112367                  |                               | 0 Successful completion.                                                                                                                                  |       |                            |
| 112368                  |                               | >0 An error occurred.                                                                                                                                     |       |                            |
| 112369                  | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                           |       |                            |
| 112370                  |                               | Default.                                                                                                                                                  |       |                            |
| 112371                  | <b>APPLICATION USAGE</b>      |                                                                                                                                                           |       |                            |
| 112372                  |                               | None.                                                                                                                                                     |       |                            |
| 112373                  | <b>EXAMPLES</b>               |                                                                                                                                                           |       |                            |
| 112374                  |                               | 1. The following example:                                                                                                                                 |       |                            |
| 112375                  |                               | <code>prs -d "User Names for :F: are:\n:UN:" s.file</code>                                                                                                |       |                            |
| 112376                  |                               | might write to standard output:                                                                                                                           |       |                            |
| 112377                  |                               | User Names for s.file are:                                                                                                                                |       |                            |
| 112378                  |                               | xyz                                                                                                                                                       |       |                            |
| 112379                  |                               | 131                                                                                                                                                       |       |                            |
| 112380                  |                               | abc                                                                                                                                                       |       |                            |
| 112381                  |                               | 2. The following example:                                                                                                                                 |       |                            |
| 112382                  |                               | <code>prs -d "Delta for pgm :M:: :I: - :D: By :P:" -r s.file</code>                                                                                       |       |                            |
| 112383                  |                               | might write to standard output:                                                                                                                           |       |                            |
| 112384                  |                               | Delta for pgm main.c: 3.7 - 77/12/01 By cas                                                                                                               |       |                            |

112385 3. As a special case:  
 112386 prs s.file  
 112387 might write to standard output:  
 112388 s.file:  
 112389 <blank line>  
 112390 D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000  
 112391 MRs:  
 112392 b178-12345  
 112393 b179-54321  
 112394 COMMENTS:  
 112395 this is the comment line for s.file initial delta  
 112396 <blank line>

112397 for each delta table entry of the **D** type. The only option allowed to be used with this  
 112398 special case is the **-a** option.

#### 112399 RATIONALE

112400 None.

#### 112401 FUTURE DIRECTIONS

112402 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 112403 value of a <newline> character when <newline> is a terminator or separator in the output  
 112404 format being used, implementations are encouraged to treat this as an error. A future version of  
 112405 this standard may require implementations to treat this as an error.

#### 112406 SEE ALSO

112407 *admin, delta, get, what*

112408 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

#### 112409 CHANGE HISTORY

112410 First released in Issue 2.

#### 112411 Issue 5

112412 The phrase “in which keyword substitution is followed by a <newline>” is deleted from the end  
 112413 of the second paragraph of [Data Keywords](#) (on page 3305).

112414 The interpretation of the *YY* component of the **-c cutoff** argument is noted.

#### 112415 Issue 6

112416 The normative text is reworded to emphasize the term “shall” for implementation requirements.

112417 The Open Group Base Resolution bwg2001-007 is applied, updating the table in STDOUT with a  
 112418 note that line statistics are capped at 99 999 for the **:Li:**, **:Ld:**, **:Lu:**, and **:DL:** keywords.

112419 The Open Group Interpretation PIN4C.00009 is applied.

#### 112420 Issue 7

112421 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

#### 112422 Issue 8

112423 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
 112424 directed to display a pathname that contains any bytes that have the encoded value of a  
 112425 <newline> character when <newline> is a terminator or separator in the output format being  
 112426 used.

112427 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

112428

Austin Group Defect 1452 is applied, deleting **:KV:** from the list of keywords.

112429

Austin Group Defect 1570 is applied, removing extra spacing in "==".

112430 **NAME**

112431 ps — report process status

112432 **SYNOPSIS**

```
112433 XSI ps [-aAw] [-defl] [-g grouplist] [-G grouplist]
112434 [-n namelist] [-o format]... [-p proclist] [-t termlist]
112435 [-u userlist] [-U userlist]
```

112436 **DESCRIPTION**

112437 The *ps* utility shall write information about processes, subject to having appropriate privileges to  
 112438 obtain information about those processes.

112439 By default, *ps* shall select all processes with the same effective user ID as the current user and the  
 112440 same controlling terminal as the invoker.

112441 **OPTIONS**

112442 The *ps* utility shall conform to XBD [Section 12.2](#) (on page 215).

112443 The following options shall be supported:

- 112444 **-a** Write information for all processes associated with terminals. Implementations  
 112445 may omit session leaders from this list.
- 112446 **-A** Write information for all processes.
- 112447 XSI **-d** Write information for all processes, except session leaders.
- 112448 XSI **-e** Write information for all processes. (Equivalent to **-A**.)
- 112449 XSI **-f** Generate a **full** listing. (See the **STDOUT** section for the contents of a **full** listing.)
- 112450 XSI **-g grouplist** Write information for processes whose session leaders are given in *grouplist*. The  
 112451 application shall ensure that the *grouplist* is a single argument in the form of a  
 112452 <blank> or <comma>-separated list.
- 112453 **-G grouplist** Write information for processes whose real group ID numbers are given in  
 112454 *grouplist*. The application shall ensure that the *grouplist* is a single argument in the  
 112455 form of a <blank> or <comma>-separated list.
- 112456 XSI **-l** Generate a **long** listing. (See **STDOUT** for the contents of a **long** listing.)
- 112457 XSI **-n namelist** Specify the name of an alternative system *namelist* file in place of the default. The  
 112458 name of the default file and the format of a *namelist* file are unspecified.
- 112459 **-o format** Write information according to the format specification given in *format*. This is  
 112460 fully described in the **STDOUT** section. Multiple **-o** options can be specified; the  
 112461 format specification shall be interpreted as the <space>-separated concatenation of  
 112462 all the *format* option-arguments.
- 112463 **-p proclist** Write information for processes whose process ID numbers are given in *proclist*.  
 112464 The application shall ensure that the *proclist* is a single argument in the form of a  
 112465 <blank> or <comma>-separated list.
- 112466 **-t termlist** Write information for processes associated with terminals given in *termlist*. The  
 112467 application shall ensure that the *termlist* is a single argument in the form of a  
 112468 <blank> or <comma>-separated list. Terminal identifiers shall be given in an  
 112469 XSI implementation-defined format. On XSI-conformant systems, they shall be given  
 112470 in one of two forms: the device's filename (for example, **tty04**) or, if the device's  
 112471 filename starts with **tty**, just the identifier following the characters **tty** (for example,  
 112472 **"04"**).

- 112473 XSI **-u *userlist*** Write information for processes whose user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is a single argument in the form of a <blank> or <comma>-separated list. In the listing, the numerical user ID shall be written unless the **-f** option is used, in which case the login name shall be written.
- 112474
- 112475
- 112476
- 112477
- 112478 **-U *userlist*** Write information for processes whose real user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is a single argument in the form of a <blank> or <comma>-separated list.
- 112479
- 112480
- 112481 **-w** Behave as if the *COLUMNS* environment variable had a value of at least 132. If the **-w** option is not specified or is specified exactly once, all output lines shall contain no more than the greater of {*LINE\_MAX*} and *COLUMNS* bytes provided that no format name is specified multiple times.
- 112482
- 112483
- 112484
- 112485 XSI With the exception of **-f**, **-l**, **-n *namelist***, and **-o *format***, all of the options shown are used to select processes. If any are specified, the default list shall be ignored and *ps* shall select the processes represented by the inclusive OR of all the selection-criteria options.
- 112486
- 112487
- 112488 **OPERANDS**
- 112489 None.
- 112490 **STDIN**
- 112491 Not used.
- 112492 **INPUT FILES**
- 112493 None.
- 112494 **ENVIRONMENT VARIABLES**
- 112495 The following environment variables shall affect the execution of *ps*:
- 112496 ***COLUMNS*** Override the system-selected horizontal display line size, used to determine the number of text columns to display. See XBD [Chapter 8](#) (on page 167) for valid values and results when it is unset or null.
- 112497
- 112498
- 112499 ***LANG*** Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 169) the precedence of internationalization variables used to determine the values of locale categories.)
- 112500
- 112501
- 112502 ***LC\_ALL*** If set to a non-empty string value, override the values of all the other internationalization variables.
- 112503
- 112504 ***LC\_CTYPE*** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 112505
- 112506
- 112507 ***LC\_MESSAGES***
- 112508 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
- 112509
- 112510
- 112511 ***LC\_TIME*** Determine the format and contents of the date and time strings displayed.
- 112512 XSI ***NLSPATH*** Determine the location of messages objects and message catalogs.
- 112513 ***TZ*** Determine the timezone used to calculate date and time strings displayed. If *TZ* is unset or null, an unspecified default timezone shall be used.
- 112514

112515 **ASYNCHRONOUS EVENTS**

112516 Default.

112517 **STDOUT**112518 When the **-o** option is not specified, the standard output format is unspecified.

112519 XSI On XSI-conformant systems, the output format shall be as follows. The column headings and descriptions of the columns in a *ps* listing are given below. The precise meanings of these fields are implementation-defined. The letters 'f' and 'l' (below) indicate the option (**full** or **long**) that shall cause the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes are listed.

|        |              |       |                                                                                                                                    |
|--------|--------------|-------|------------------------------------------------------------------------------------------------------------------------------------|
| 112525 | <b>F</b>     | (l)   | Flags (octal and additive) associated with the process.                                                                            |
| 112526 | <b>S</b>     | (l)   | The state of the process.                                                                                                          |
| 112527 | <b>UID</b>   | (f,l) | The user ID number of the process owner; the login name is printed under the <b>-f</b> option.                                     |
| 112528 |              |       |                                                                                                                                    |
| 112529 | <b>PID</b>   | (all) | The process ID of the process; it is possible to kill a process if this datum is known.                                            |
| 112530 |              |       |                                                                                                                                    |
| 112531 | <b>PPID</b>  | (f,l) | The process ID of the parent process.                                                                                              |
| 112532 | <b>C</b>     | (f,l) | Processor utilization for scheduling.                                                                                              |
| 112533 | <b>PRI</b>   | (l)   | The priority of the process; higher numbers mean lower priority.                                                                   |
| 112534 | <b>NI</b>    | (l)   | Nice value; used in priority computation.                                                                                          |
| 112535 | <b>ADDR</b>  | (l)   | The address of the process.                                                                                                        |
| 112536 | <b>SZ</b>    | (l)   | The size in pages of the total memory requirements of the process, including text, data, stack, mapped memory and other resources. |
| 112537 |              |       |                                                                                                                                    |
| 112538 | <b>WCHAN</b> | (l)   | The event for which the process is waiting or sleeping; if blank, the process is running.                                          |
| 112539 |              |       |                                                                                                                                    |
| 112540 | <b>STIME</b> | (f)   | Starting time of the process.                                                                                                      |
| 112541 | <b>TTY</b>   | (all) | The controlling terminal for the process.                                                                                          |
| 112542 | <b>TIME</b>  | (all) | The cumulative execution time for the process.                                                                                     |
| 112543 | <b>CMD</b>   | (all) | The command name; the full command name and its arguments are written under the <b>-f</b> option.                                  |
| 112544 |              |       |                                                                                                                                    |

112545 A process that has exited and has a parent, but has not yet been waited for by the parent, shall be marked **defunct**.

112546

112547 Under the option **-f**, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the option **-f**, is written in square brackets.

112548

112549

112550 The **-o** option allows the output format to be specified under user control.

112551 The application shall ensure that the format specification is a list of names presented as a single argument, <blank> or <comma>-separated. Each variable has a default header. The default header can be overridden by appending an <equals-sign> and the new text of the header. The rest of the characters in the argument shall be used as the header text. The fields specified shall be written in the order specified on the command line, and should be arranged in columns in the output. The field widths shall be selected by the system to be at least as wide as the header text (default or overridden value). If the header text is null, such as **-o user=**, the field width shall be at least as wide as the default header text. If all header text fields are null, no header line shall be written.

112552 The following names are recognized in the POSIX locale:

112553

112554

112555

112556

112557

112558

112559



|        |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 112561 | <b>ruser</b>  | The real user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                            |
| 112562 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112563 | <b>user</b>   | The effective user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                       |
| 112564 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112565 | <b>rgroup</b> | The real group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                          |
| 112566 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112567 | <b>group</b>  | The effective group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.                                                                                                                                                                                                                                                                                                                                                                                     |
| 112568 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112569 | <b>pid</b>    | The decimal value of the process ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 112570 | <b>ppid</b>   | The decimal value of the parent process ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 112571 | <b>pgid</b>   | The decimal value of the process group ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 112572 | <b>pcpu</b>   | The ratio of CPU time used recently to CPU time available in the same period, expressed as a percentage. The meaning of “recently” in this context is unspecified. The CPU time available is determined in an unspecified manner.                                                                                                                                                                                                                                                                                                                        |
| 112573 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112574 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112575 | <b>vsz</b>    | The size of the process in (virtual) memory in 1 024 byte units as a decimal integer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 112576 | <b>nice</b>   | The decimal value of the nice value of the process; see <i>nice</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 112577 | <b>etime</b>  | In the POSIX locale, the elapsed time since the process was started, in the form:                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 112578 |               | [ [ <i>dd</i> -] <i>hh</i> : ] <i>mm</i> : <i>ss</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 112579 |               | where <i>dd</i> shall represent the number of days, <i>hh</i> the number of hours, <i>mm</i> the number of minutes, and <i>ss</i> the number of seconds. The <i>dd</i> field shall be a decimal integer. The <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be two-digit decimal integers padded on the left with zeros.                                                                                                                                                                                                                             |
| 112580 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112581 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112582 | <b>time</b>   | In the POSIX locale, the cumulative CPU time of the process in the form:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 112583 |               | [ [ <i>dd</i> -] <i>hh</i> : <i>mm</i> : <i>ss</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 112584 |               | The <i>dd</i> , <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be as described in the <b>etime</b> specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 112585 | <b>tty</b>    | The name of the controlling terminal of the process (if any) in the same format used by the <i>who</i> utility.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112586 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112587 | <b>comm</b>   | The name of the command being executed ( <i>argv</i> [0] value) as a string.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 112588 | <b>args</b>   | The command with all its arguments as a string. The implementation may truncate this value to the field width; it is implementation-defined whether any further truncation occurs. It is unspecified whether the string represented is a version of the argument list as it was passed to the command when it started, or is a version of the arguments as they may have been modified by the application. Applications cannot depend on being able to modify their argument list and having that modification be reflected in the output of <i>ps</i> . |
| 112589 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112590 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112591 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112592 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112593 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112594 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112595 |               | Any field need not be meaningful in all implementations. In such a case a <hyphen-minus>                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 112596 |               | (' - ') should be output in place of the field value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 112597 |               | Only <b>comm</b> and <b>args</b> shall be allowed to contain <blank> characters; all others shall not. Any                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 112598 |               | implementation-defined variables shall be specified in the system documentation along with the                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 112599 |               | default header and indicating whether the field may contain <blank> characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 112600 |               | The following table specifies the default header to be used in the POSIX locale corresponding to                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 112601 |               | each format specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

112602

Table 3-19 Variable Names and Default Headers in *ps*

112603

| Format Specifier | Default Header | Format Specifier | Default Header |
|------------------|----------------|------------------|----------------|
| <b>args</b>      | <b>COMMAND</b> | <b>ppid</b>      | <b>PPID</b>    |
| <b>comm</b>      | <b>COMMAND</b> | <b>rgroup</b>    | <b>RGROUP</b>  |
| <b>etime</b>     | <b>ELAPSED</b> | <b>ruser</b>     | <b>RUSER</b>   |
| <b>group</b>     | <b>GROUP</b>   | <b>time</b>      | <b>TIME</b>    |
| <b>nice</b>      | <b>NI</b>      | <b>tty</b>       | <b>TT</b>      |
| <b>pcpu</b>      | <b>%CPU</b>    | <b>user</b>      | <b>USER</b>    |
| <b>pgid</b>      | <b>PGID</b>    | <b>vsz</b>       | <b>VSZ</b>     |
| <b>pid</b>       | <b>PID</b>     |                  |                |

112604

112605

112606

112607

112608

112609

112610

112611

**112612 STDERR**

112613 The standard error shall be used only for diagnostic messages.

**112614 OUTPUT FILES**

112615 None.

**112616 EXTENDED DESCRIPTION**

112617 None.

**112618 EXIT STATUS**

112619 The following exit values shall be returned:

112620 0 Successful completion.

112621 >0 An error occurred.

**112622 CONSEQUENCES OF ERRORS**

112623 Default.

**112624 APPLICATION USAGE**

112625 Things can change while *ps* is running; the snapshot it gives is only true for an instant, and  
112626 might not be accurate by the time it is displayed.

112627 The **args** format specifier is allowed to produce a truncated version of the command arguments.  
112628 In some implementations, this information is no longer available when the *ps* utility is executed.

112629 If the field width is too narrow to display a textual ID, the system may use a numeric version.  
112630 Normally, the system would be expected to choose large enough field widths, but if a large  
112631 number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on  
112632 one line. One way to ensure adequate width for the textual IDs is to override the default header  
112633 for a field to make it larger than most or all user or group names.

112634 Portable applications should not set *COLUMNS* to a value greater than {*LINE\_MAX*} and should  
112635 not specify the **-w** option more than once if the output will be used as input for a utility that  
112636 requires a text file as input because lines containing more than {*LINE\_MAX*} bytes may cause  
112637 undefined behavior in some implementations of the utility.

112638 There is no special quoting mechanism for header text. The header text is the rest of the  
112639 argument. If multiple header changes are needed, multiple **-o** options can be used, such as:

```
112640 ps -o "user=User Name" -o pid=Process\ ID
```

112641 On some implementations, especially multi-level secure systems, *ps* may be severely restricted  
112642 and produce information only about child processes owned by the user.

112643 **EXAMPLES**

112644 The command:

```
112645 ps -o user,pid,ppid=MOM -o args
```

112646 writes at least the following in the POSIX locale:

```
112647     USER    PID    MOM    COMMAND
112648 helene     34     12    ps -o user,pid,ppid=MOM -o args
```

112649 The contents of the **COMMAND** field need not be the same in all implementations, due to  
112650 possible truncation.

112651 **RATIONALE**

112652 There is very little commonality between BSD and System V implementations of *ps*. Many  
112653 options conflict or have subtly different usages. The standard developers attempted to select a  
112654 set of options for the base standard that were useful on a wide range of systems and selected  
112655 options that either can be implemented on both BSD and System V-based systems without  
112656 breaking the current implementations or where the options are sufficiently similar that any  
112657 changes would not be unduly problematic for users or implementors.

112658 It is recognized that on some implementations, especially multi-level secure systems, *ps* may be  
112659 nearly useless. The default output has therefore been chosen such that it does not break  
112660 historical implementations and also is likely to provide at least some useful information on most  
112661 systems.

112662 The major change is the addition of the format specification capability. The motivation for this  
112663 invention is to provide a mechanism for users to access a wider range of system information, if  
112664 the system permits it, in a portable manner. The fields chosen to appear in this volume of  
112665 POSIX.1-2024 were arrived at after considering what concepts were likely to be both reasonably  
112666 useful to the “average” user and had a reasonable chance of being implemented on a wide range  
112667 of systems. Again it is recognized that not all systems are able to provide all the information  
112668 and, conversely, some may wish to provide more. It is hoped that the approach adopted will be  
112669 sufficiently flexible and extensible to accommodate most systems. Implementations may be  
112670 expected to introduce new format specifiers.

112671 The default output should consist of a short listing containing the process ID, terminal name,  
112672 cumulative execution time, and command name of each process.

112673 The preference of the standard developers would have been to make the format specification an  
112674 operand of the *ps* command. Unfortunately, BSD usage precluded this.

112675 At one time a format was included to display the environment array of the process. This was  
112676 deleted because there is no portable way to display it.

112677 The **-A** option is equivalent to the BSD **-g** and the SVID **-e**. Because the two systems differed, a  
112678 mnemonic compromise was selected.

112679 The **-a** option is described with some optional behavior because the SVID omits session leaders,  
112680 but BSD does not.

112681 In an early proposal, format specifiers appeared for priority and start time. The former was not  
112682 defined adequately in this volume of POSIX.1-2024 and was removed in deference to the defined  
112683 nice value; the latter because elapsed time was considered to be more useful.

112684 In a new BSD version of *ps*, a **-O** option can be used to write all of the default information,  
112685 followed by additional format specifiers. This was not adopted because the default output is  
112686 implementation-defined. Nevertheless, this is a useful option that should be reserved for that  
112687 purpose. In the **-o** option for the POSIX Shell and Utilities *ps*, the format is the concatenation of

112688 each **-o**. Therefore, the user can have an alias or function that defines the beginning of their  
112689 desired format and add more fields to the end of the output in certain cases where that would be  
112690 useful.

112691 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
112692 require that they all use the same format.

112693 The **pcpu** field indicates that the CPU time available is determined in an unspecified manner.  
112694 This is because it is difficult to express an algorithm that is useful across all possible machine  
112695 architectures. Historical counterparts to this value have attempted to show percentage of use in  
112696 the recent past, such as the preceding minute. Frequently, these values for all processes did not  
112697 add up to 100%. Implementations are encouraged to provide data in this field to users that will  
112698 help them identify processes currently affecting the performance of the system.

#### 112699 **FUTURE DIRECTIONS**

112700 None.

#### 112701 **SEE ALSO**

112702 *kill*, *nice*, *renice*

112703 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 112704 **CHANGE HISTORY**

112705 First released in Issue 2.

#### 112706 **Issue 6**

112707 This utility is marked as part of the User Portability Utilities option.

112708 The normative text is reworded to avoid use of the term “must” for application requirements.

112709 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

#### 112710 **Issue 7**

112711 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112712 SD5-XCU-ERN-148 is applied, updating the OPTIONS section.

112713 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0160 [584] is applied.

#### 112714 **Issue 8**

112715 Austin Group Defect 905 is applied, adding the **-w** option.

112716 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

112717 Austin Group Defect 1141 is applied, changing the description of **SZ** in the STDOUT section.

112718 Austin Group Defect 1175 is applied, changing “uid” to “user” in the EXAMPLES section.

112719 **NAME**112720 `pwd` — return working directory name112721 **SYNOPSIS**112722 `pwd [-L|-P]`112723 **DESCRIPTION**112724 The *pwd* utility shall write to standard output an absolute pathname of the current working  
112725 directory, which does not contain the filenames dot or dot-dot.112726 **OPTIONS**112727 The *pwd* utility shall conform to XBD [Section 12.2](#) (on page 215).

112728 The following options shall be supported by the implementation:

112729 **-L** If the *PWD* environment variable contains an absolute pathname of the current  
112730 directory and the pathname does not contain any components that are dot or dot-  
112731 dot, *pwd* shall write this pathname to standard output, except that if the *PWD*  
112732 environment variable is longer than {PATH\_MAX} bytes including the terminating  
112733 null, it is unspecified whether *pwd* writes this pathname to standard output or  
112734 behaves as if the **-P** option had been specified. Otherwise, the **-L** option shall  
112735 behave as the **-P** option.

112736 **-P** The pathname written to standard output shall not contain any components that  
112737 refer to files of type symbolic link. If there are multiple pathnames that the *pwd*  
112738 utility could write to standard output, one beginning with a single <slash>  
112739 character and one or more beginning with two <slash> characters, then it shall  
112740 write the pathname beginning with a single <slash> character. The pathname shall  
112741 not contain any unnecessary <slash> characters after the leading one or two  
112742 <slash> characters.

112743 If both **-L** and **-P** are specified, the last one shall apply. If neither **-L** nor **-P** is specified, the *pwd*  
112744 utility shall behave as if **-L** had been specified.

112745 **OPERANDS**

112746 None.

112747 **STDIN**

112748 Not used.

112749 **INPUT FILES**

112750 None.

112751 **ENVIRONMENT VARIABLES**112752 The following environment variables shall affect the execution of *pwd*:

112753 **LANG** Provide a default value for the internationalization variables that are unset or null.  
112754 (See XBD [Section 8.2](#) (on page 169) the precedence of internationalization variables  
112755 used to determine the values of locale categories.)

112756 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
112757 internationalization variables.

112758 **LC\_MESSAGES**

112759 Determine the locale that should be used to affect the format and contents of  
112760 diagnostic messages written to standard error.

112761 **XSI** **NLSPATH** Determine the location of messages objects and message catalogs.

112762 *PWD* An absolute pathname of the current working directory. If an application sets or  
112763 unsets the value of *PWD*, the behavior of *pwd* is unspecified.

#### 112764 **ASYNCHRONOUS EVENTS**

112765 Default.

#### 112766 **STDOUT**

112767 The *pwd* utility output is an absolute pathname of the current working directory:

112768 "%s\n", <directory pathname>

#### 112769 **STDERR**

112770 The standard error shall be used only for diagnostic messages.

#### 112771 **OUTPUT FILES**

112772 None.

#### 112773 **EXTENDED DESCRIPTION**

112774 None.

#### 112775 **EXIT STATUS**

112776 The following exit values shall be returned:

112777 0 Successful completion.

112778 >0 An error occurred.

#### 112779 **CONSEQUENCES OF ERRORS**

112780 If an error is detected other than a write error when writing to standard output, no output shall  
112781 be written to standard output, a diagnostic message shall be written to standard error, and the  
112782 exit status shall be non-zero.

#### 112783 **APPLICATION USAGE**

112784 If the pathname obtained from *pwd* is longer than {PATH\_MAX} bytes, it could produce an error  
112785 if passed to *cd*. Therefore, in order to return to that directory it may be necessary to break the  
112786 pathname into sections shorter than {PATH\_MAX} and call *cd* on each section in turn (the first  
112787 section being an absolute pathname and subsequent sections being relative pathnames).

#### 112788 **EXAMPLES**

112789 None.

#### 112790 **RATIONALE**

112791 Some implementations have historically provided *pwd* as a shell special built-in command.

112792 In most utilities, if an error occurs, partial output may be written to standard output. This does  
112793 not happen in historical implementations of *pwd* (unless an error condition causes a partial  
112794 write). Because *pwd* is frequently used in historical shell scripts without checking the exit status,  
112795 it is important that the historical behavior is required here; therefore, the CONSEQUENCES OF  
112796 ERRORS section specifically disallows any partial output being written to standard output,  
112797 except when a write error occurs when writing to standard output.

112798 An earlier version of this standard stated that the *PWD* environment variable was affected when  
112799 the *-P* option was in effect. This was incorrect; conforming implementations do not do this.

#### 112800 **FUTURE DIRECTIONS**

112801 If this utility is directed to display a pathname that contains any bytes that have the encoded  
112802 value of a <newline> character when <newline> is a terminator or separator in the output  
112803 format being used, implementations are encouraged to treat this as an error. A future version of  
112804 this standard may require implementations to treat this as an error.

112805 **SEE ALSO**

- 112806 *cd*
- 112807 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)
- 112808 XSH *getcwd()*

112809 **CHANGE HISTORY**

- 112810 First released in Issue 2.
- 112811 **Issue 6**
- 112812 The **-P** and **-L** options are added to describe actions relating to symbolic links as specified in the
- 112813 IEEE P1003.2b draft standard.
- 112814 **Issue 7**
- 112815 Austin Group Interpretation 1003.1-2001 #097 is applied.
- 112816 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 112817 Changes to the *pwd* utility and *PWD* environment variable have been made to match the
- 112818 changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.
- 112819 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0161 [471] is applied.
- 112820 **Issue 8**
- 112821 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is
- 112822 directed to display a pathname that contains any bytes that have the encoded value of a
- 112823 <newline> character when <newline> is a terminator or separator in the output format being
- 112824 used.
- 112825 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 112826 Austin Group Defect 1488 is applied, clarifying the behavior when a write error occurs when
- 112827 writing to standard output.

112828 **NAME**

112829 read — read from standard input into shell variables

112830 **SYNOPSIS**112831 read [-r] [-d *delim*] *var*...112832 **DESCRIPTION**112833 The *read* utility shall read a single logical line from standard input into one or more shell  
112834 variables.

112835 If the *-r* option is not specified, <backslash> shall act as an escape character. An unescaped  
112836 <backslash> shall preserve the literal value of a following <backslash> and shall prevent a  
112837 following byte (if any) from being used to split fields, with the exception of either <newline> or  
112838 the logical line delimiter specified with the *-d delim* option (if it is used and *delim* is not  
112839 <newline>); it is unspecified which. If this excepted character follows the <backslash>, the *read*  
112840 utility shall interpret this as line continuation. The <backslash> and the excepted character shall  
112841 be removed before splitting the input into fields. All other unescaped <backslash> characters  
112842 shall be removed after splitting the input into fields.

112843 If standard input is a terminal device and the invoking shell is interactive, *read* shall prompt for a  
112844 continuation line when it reads an input line ending with a <backslash> <newline>, unless the  
112845 *-r* option is specified.

112846 The terminating logical line delimiter (if any) shall be removed from the input. Then, if the shell  
112847 variable *IFS* (see [Section 2.5.3](#), on page 2481) is set, and its value is an empty string, the resulting  
112848 data shall be assigned to the variable named by the first *var* operand, and the variables named  
112849 by other *var* operands (if any) shall be set to the empty string. No other processing shall be  
112850 performed in this case.

112851 If *IFS* is unset, or is set to any non-empty value, then a modified version of the field splitting  
112852 algorithm specified in [Section 2.6.5](#) (on page 2491) shall be applied, with the modifications as  
112853 follows:

- 112854 1. The input to the algorithm shall be the logical line (minus terminating delimiter) that was  
112855 read from standard input, and shall be considered as a single initial field, all of which  
112856 resulted from expansions, with any escaped byte and the preceding <backslash> escape  
112857 character treated as if they were the result of a quoted expansion, and all other bytes  
112858 treated as if they were the results of unquoted expansions.
- 112859 2. The loop over the contents of that initial field shall cease when either the input is empty  
112860 or *n* output fields have been generated, where *n* is one less than the number of *var*  
112861 operands passed to the *read* utility. Any remaining input in the original field being  
112862 processed shall be returned to the *read* utility “unsplit”; that is, unmodified except that  
112863 any leading or trailing *IFS* white space, as defined in [Section 2.6.5](#) (on page 2491), shall be  
112864 removed.

112865 The specified *var* operands shall be processed in the order they appear on the command line,  
112866 and the output fields generated by the field splitting algorithm shall be used in the order they  
112867 were generated, by repeating the following checks until neither is true:

- 112868 • If more than one *var* operand is yet to be processed and one or more output fields are yet to  
112869 be used, the variable named by the first unprocessed *var* operand shall be assigned the  
112870 value of the first unused output field.
- 112871 • If exactly one *var* operand is yet to be processed and there was some remaining unsplit  
112872 input returned from the modified field splitting algorithm, the variable named by the  
112873 unprocessed *var* operand shall be assigned the unsplit input.



112874 If there are still one or more unprocessed *var* operands, each of the variables names by those  
112875 operands shall be assigned an empty string.

112876 Note that in the case where just one *var* operand is given on the *read* command line, the modified  
112877 field splitting algorithm ceases after producing zero output fields and simply returns the  
112878 original input field, with any leading and trailing *IFS* white space removed, as unsplit input.  
112879 This unsplit input is assigned to the variable named by the *var* operand.

112880 The setting of variables specified by the *var* operands shall affect the current shell execution  
112881 environment; see [Section 2.13](#) (on page 2522). An error in setting any variable (such as if a *var*  
112882 has previously been marked *readonly*) shall be considered an error of *read* processing, and shall  
112883 result in a return value greater than one. Variables named before the one generating the error  
112884 shall be set as described above; it is unspecified whether variables named later shall be set as  
112885 above, or *read* simply ceases processing when the error occurs, leaving later named variables  
112886 unaltered. If *read* is called in a subshell or separate utility execution environment, such as one of  
112887 the following:

```
112888 (read foo)
112889 nohup read ...
112890 find . -exec read ... \;
```

112891 it shall not affect the shell variables in the caller's environment.

112892 If end-of-file is detected before a terminating logical line delimiter is encountered, the variables  
112893 specified by the *var* operands shall be set as described above and the exit status shall be 1.

#### 112894 OPTIONS

112895 The *read* utility shall conform to XBD [Section 12.2](#) (on page 215).

112896 The following options shall be supported:

112897 **-d** *delim* If *delim* consists of one single-byte character, that byte shall be used as the logical  
112898 line delimiter. If *delim* is the null string, the logical line delimiter shall be the null  
112899 byte. Otherwise, the behavior is unspecified.

112900 **-r** Do not treat a <backslash> character in any special way. Consider each  
112901 <backslash> to be part of the input line.

#### 112902 OPERANDS

112903 The following operand shall be supported:

112904 *var* The name of an existing or nonexisting shell variable. If a *var* operand names the  
112905 variable *IFS*, the behavior is unspecified.

112906 If a *var* operand names one of the variables *LANG*, *LC\_CTYPE*, or *LC\_ALL* and the  
112907 new value assigned to the variable would change how the bytes in *IFS* form  
112908 characters, or which characters in *IFS* are considered to be *IFS* white space (see  
112909 [Section 2.6.5](#), on page 2491), it is unspecified what effects, if any, the change has on  
112910 how *read* performs field splitting.

#### 112911 STDIN

112912 If the **-d** *delim* option is not specified, or if it is specified and *delim* is not the null string, the  
112913 standard input shall contain zero or more bytes (which need not form valid characters) and shall  
112914 not contain any null bytes.

112915 If the **-d** *delim* option is specified and *delim* is the null string, the standard input shall contain  
112916 zero or more bytes (which need not form valid characters).

112917 **INPUT FILES**

112918 None.

112919 **ENVIRONMENT VARIABLES**112920 The following environment variables shall affect the execution of *read*:112921 *IFS* Determine the internal field separators used to delimit fields; see [Section 2.5.3](#) (on  
112922 page 2481).112923 *LANG* Provide a default value for the internationalization variables that are unset or null.  
112924 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
112925 variables used to determine the values of locale categories.)112926 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
112927 internationalization variables.112928 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
112929 characters (for example, single-byte as opposed to multi-byte characters in  
112930 arguments).112931 *LC\_MESSAGES*112932 Determine the locale that should be used to affect the format and contents of  
112933 diagnostic messages written to standard error.112934 XSI *NLSPATH* Determine the location of messages objects and message catalogs.112935 *PS2* Provide the prompt string that an interactive shell shall write to standard error  
112936 when a line ending with a <backslash> <newline> is read and the *-r* option was  
112937 not specified.112938 **ASYNCHRONOUS EVENTS**

112939 Default.

112940 **STDOUT**

112941 Not used.

112942 **STDERR**

112943 The standard error shall be used for diagnostic messages and prompts for continued input.

112944 **OUTPUT FILES**

112945 None.

112946 **EXTENDED DESCRIPTION**

112947 None.

112948 **EXIT STATUS**

112949 The following exit values shall be returned:

112950 0 Successful completion.

112951 1 End-of-file was detected.

112952 &gt;1 An error occurred.

112953 **CONSEQUENCES OF ERRORS**

112954 Default.

112955 **APPLICATION USAGE**

112956 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

112957 The `-r` option is included to enable `read` to subsume the purpose of the `line` utility, which is not  
112958 included in POSIX.1-2024.

112959 The `-d delim` option enables reading up to an arbitrary single-byte delimiter. When *delim* is the  
112960 null string, the delimiter is the null byte and this allows `read` to be used to process null-  
112961 terminated lists of pathnames (as produced by the `find -print0` primary), with correct handling  
112962 of pathnames that contain `<newline>` characters. Note that in order to specify the null string as  
112963 the delimiter, `-d` and *delim* need to be specified as two separate arguments. Implementations  
112964 differ in their handling of `<backslash>` for line continuation when `-d delim` is specified (and  
112965 *delim* is not `<newline>`); some treat `<backslash>delim` (or `<backslash><NUL>` if *delim* is the null  
112966 string) as a line continuation, whereas others still treat `<backslash><newline>` as a line  
112967 continuation. Consequently, portable applications need to specify `-r` whenever they specify `-d  
112968 delim` (and *delim* is not `<newline>`).

112969 When reading a pathname it is inadvisable to use the contents of the first *var* operand, if non-  
112970 empty, when the exit status of `read` is 1, as it is likely the result of the command used to generate  
112971 the list of pathnames (for example `find` with `-print` or `-print0`) being terminated after it has  
112972 written a partial pathname, and consequently using it could result in the wrong pathname being  
112973 processed.

112974 Since the *var* operands are processed in the order specified on the command line, if any variable  
112975 name is specified more than once as a *var* operand, the last assignment made is the one that is in  
112976 effect when `read` returns, including when an empty string is assigned because no field data was  
112977 available.

112978 **EXAMPLES**

112979 The following command:

```
112980 while read -r xx yy
112981 do
112982     printf "%s %s\n" "$yy" "$xx"
112983 done < input_file
```

112984 prints a file with the first field of each line moved to the end of the line.

112985 **RATIONALE**

112986 The `read` utility historically has been a shell built-in. It was separated off into its own utility to  
112987 take advantage of the richer description of functionality introduced by this volume of  
112988 POSIX.1-2024.

112989 Since `read` affects the current shell execution environment, it is required to be intrinsic. If it is  
112990 called in a subshell or separate utility execution environment, such as one of the following:

```
112991 (read foo)
112992 nohup read ...
112993 find . -exec read ... \;
```

112994 it does not affect the shell variables in the environment of the caller.

112995 Earlier versions of this standard required the standard input to be a text file, and therefore the  
112996 results were undefined if the input was not empty and end-of-file was detected before a  
112997 `<newline>` character was encountered. However, all of the most popular shell implementations  
112998 have been found to have consistent behavior in this case, and so the behavior is now specified  
112999 and the requirement for standard input to be a text file has been relaxed to allow non-empty  
113000 input that does not end with a `<newline>`.

113001 **FUTURE DIRECTIONS**

113002 None.

113003 **SEE ALSO**113004 [Chapter 2](#) (on page 2472)113005 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)113006 **CHANGE HISTORY**

113007 First released in Issue 2.

113008 **Issue 7**

113009 Austin Group Interpretation 1003.1-2001 #194 is applied, clarifying the handling of the &lt;backslash&gt; escape character.

113011 SD5-XCU-ERN-126 is applied, clarifying that input lines end with a &lt;newline&gt;.

113012 The description of here-documents is removed from the *read* reference page.

113013 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0162 [958] is applied.

113014 **Issue 8**113015 Austin Group Defect 243 is applied, adding the `-d` option and relaxing the requirement for standard input to be a text file.113017 Austin Group Defect 367 is applied, requiring that *read* distinguishes between detecting end-of-file and an error occurring, setting its exit status to one and greater than one, respectively.

113019 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that this utility is required to be intrinsic.

113021 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

113022 Austin Group Defect 1778 is applied, clarifying how field splitting is performed.

113023 Austin Group Defect 1779 is applied, clarifying how an error in setting any variable affects the processing of variables named before or after the one generating the error.

113024

113025 **NAME**

113026 readlink — display the contents of a symbolic link

113027 **SYNOPSIS**113028 readlink [-n] *file*113029 **DESCRIPTION**

113030 If the *file* operand names a symbolic link, the *readlink* utility shall not follow the symbolic link  
 113031 when resolving *file* and shall write the contents of the symbolic link to standard output. If the **-n**  
 113032 option is not specified, the output to standard output shall be followed by a <newline>  
 113033 character.

113034 If *file* does not name a symbolic link, *readlink* shall write a diagnostic message to standard error  
 113035 and exit with non-zero status.

113036 **OPTIONS**113037 The *readlink* utility shall conform to XBD [Section 12.2](#) (on page 215).

113038 The following option shall be supported:

113039 **-n** Do not output a trailing <newline> character.113040 **OPERANDS**

113041 The following operand shall be supported:

113042 *file* A pathname of a symbolic link to be read.113043 **STDIN**

113044 Not used.

113045 **INPUT FILES**

113046 None.

113047 **ENVIRONMENT VARIABLES**113048 The following environment variables shall affect the execution of *readlink*:

113049 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 113050 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 113051 variables used to determine the values of locale categories.)

113052 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 113053 internationalization variables.

113054 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 113055 characters (for example, single-byte as opposed to multi-byte characters in  
 113056 arguments and input files).

113057 **LC\_MESSAGES**

113058 Determine the locale that should be used to affect the format and contents of  
 113059 diagnostic messages written to standard error.

113060 **XSI** **NLSPATH** Determine the location of messages objects and message catalogs.113061 **ASYNCHRONOUS EVENTS**

113062 Default.

113063 **STDOUT**

113064 See DESCRIPTION.

- 113065 **STDERR**  
113066       The standard error shall be used only for diagnostic messages.
- 113067 **OUTPUT FILES**  
113068       None.
- 113069 **EXTENDED DESCRIPTION**  
113070       None.
- 113071 **EXIT STATUS**  
113072       The following exit values shall be returned:  
113073       0 Successful completion.  
113074       >0 An error occurred.
- 113075 **CONSEQUENCES OF ERRORS**  
113076       Default.
- 113077 **APPLICATION USAGE**  
113078       None.
- 113079 **EXAMPLES**  
113080       None.
- 113081 **RATIONALE**  
113082       The *readlink* utility was added because using *ls -l* to obtain the contents of a symbolic link is difficult if the output includes more than one occurrence of the string " -> ".  
113083  
113084       The *-f* option found in many implementations was not included, as the *realpath* utility provides equivalent functionality with a choice of behaviors.  
113085
- 113086 **FUTURE DIRECTIONS**  
113087       None.
- 113088 **SEE ALSO**  
113089       *ln, ls, realpath*  
113090       XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)  
113091       XSH [readlink\(\)](#)
- 113092 **CHANGE HISTORY**  
113093       First released in Issue 8.

113094 **NAME**113095        *realpath* — resolve a pathname113096 **SYNOPSIS**113097        *realpath* [-E|-e] *file*113098 **DESCRIPTION**113099        The *realpath* utility shall canonicalize the pathname specified by the *file* operand as follows:113100        If a call to the *realpath*() function with the specified pathname as its first argument would  
113101        succeed, the canonicalized pathname shall be the pathname that would be returned by that  
113102        *realpath*() call. Otherwise:

- 113103           • If the **-e** option is specified, the canonicalization shall fail.
- 113104           • If the **-E** option is specified, then if a call to the *realpath*() function with the specified  
113105           pathname as its first argument would encounter an error condition other than [ENOENT],  
113106           the canonicalization shall fail; if the call would encounter an [ENOENT] error, *realpath* shall  
113107           expand all symbolic links that would be encountered in an attempt to resolve the specified  
113108           pathname using the algorithm specified in XBD [Section 4.16](#) (on page 105), except that any  
113109           trailing <slash> characters that are not also leading <slash> characters shall be ignored. If  
113110           this expansion succeeds and the path prefix of the expanded pathname resolves to an  
113111           existing directory, the canonicalized pathname shall be the expanded pathname. In all  
113112           other cases, the canonicalization shall fail. If the expanded pathname is not empty, does not  
113113           begin with a <slash>, and has exactly one pathname component, it shall be treated as if it  
113114           had a path prefix of ". /".
- 113115           • If no options are specified, *realpath* shall canonicalize the specified pathname in an  
113116           unspecified manner such that the resulting absolute pathname does not contain any  
113117           components that refer to files of type symbolic link and does not contain any components  
113118           that are dot or dot-dot.

113119        Upon successful canonicalization, *realpath* shall write the canonicalized pathname, followed by a  
113120        <newline> character, to standard output.113121        If canonicalization fails, or the canonicalized pathname is empty, nothing shall be written to  
113122        standard output, a diagnostic message shall be written to standard error, and *realpath* shall exit  
113123        with non-zero status.113124 **OPTIONS**113125        The *realpath* utility shall conform to XBD [Section 12.2](#) (on page 215).

113126        The following options shall be supported:

113127        **-E**        Do not treat it as an error if attempting to resolve the last component of the  
113128        canonicalized form of the *file* operand results in an [ENOENT] error condition.113129        **-e**        Treat it as an error if attempting to resolve the last component of the canonicalized  
113130        form of the *file* operand results in an [ENOENT] error condition.113131        Specifying more than one of the mutually-exclusive options **-E** and **-e** shall not be considered an  
113132        error. The last option specified shall determine the behavior of the utility.113133 **OPERANDS**

113134        The following operand shall be supported:

113135        *file*        A pathname to be canonicalized.

113136 **STDIN**

113137 Not used.

113138 **INPUT FILES**

113139 None.

113140 **ENVIRONMENT VARIABLES**113141 The following environment variables shall affect the execution of *realpath*:

113142 *LANG* Provide a default value for the internationalization variables that are unset or null.  
113143 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
113144 variables used to determine the values of locale categories.)

113145 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
113146 internationalization variables.

113147 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
113148 characters (for example, single-byte as opposed to multi-byte characters in  
113149 arguments and input files).

113150 *LC\_MESSAGES*

113151 Determine the locale that should be used to affect the format and contents of  
113152 diagnostic messages written to standard error.

113153 *XSI NLSPATH* Determine the location of messages objects and message catalogs.

113154 **ASYNCHRONOUS EVENTS**

113155 Default.

113156 **STDOUT**

113157 See DESCRIPTION.

113158 **STDERR**

113159 The standard error shall be used only for diagnostic messages.

113160 **OUTPUT FILES**

113161 None.

113162 **EXTENDED DESCRIPTION**

113163 None.

113164 **EXIT STATUS**

113165 The following exit values shall be returned:

113166 0 Successful completion.

113167 &gt;0 An error occurred.

113168 **CONSEQUENCES OF ERRORS**

113169 Default.



113170 **APPLICATION USAGE**

113171 If neither the `-e` nor the `-E` option is specified, some implementations behave as if `-e` had been  
 113172 specified and others as if `-E` had been specified, but there are also implementations where the  
 113173 behavior differs from both of these. For example, the *mksh* shell has an internal implementation  
 113174 of *realpath* that canonicalizes `/dir/regular_file/..` to `/dir`, whereas the *realpath()* function would  
 113175 return an [ENOTDIR] error in this case. Portable applications should always specify either `-e` or  
 113176 `-E`.

113177 **EXAMPLES**

113178 None.

113179 **RATIONALE**

113180 The *realpath* utility was added in preference to a `-f` option found in some implementations of the  
 113181 *readlink* utility because it allows the application to specify whether or not a missing final  
 113182 component is to be treated as an error.

113183 The behavior with the `-E` option when *file* does not resolve (with symbolic links followed) to an  
 113184 existing file is not the same as simply calling *realpath()* with the path prefix of the *file* operand  
 113185 and writing the resulting pathname, a `<slash>`, and the last component of *file* to standard output.  
 113186 For example, if `/tmp/nofile` does not exist, and *file* is `A/B` where `A` is an existing directory and `B`  
 113187 is a symbolic link to `/tmp/nofile`, *realpath* with `-E` will output `/tmp/nofile`, but if `B` is a symbolic  
 113188 link to `/tmp/nofile/foo`, *realpath* with `-E` will treat this as an error. In both cases  
 113189 *realpath("A/B")* would fail with *errno* set to [ENOENT]. Even though *realpath("A")*  
 113190 would succeed, in neither case is anything ending `/B` the result.

113191 Trailing `<slash>` characters (that follow a non-`<slash>`) are handled differently with `-E` than with  
 113192 `-e`. With `-e` they are handled as for the *realpath()* function. With `-E` they are sometimes  
 113193 effectively ignored, and they are never included in the output. For example, if `/tmp/nofile` does  
 113194 not exist and `/tmp/regfile` is an existing regular file:

```
113195 $ realpath -E /tmp/nofile/  
113196 /tmp/nofile  
113197 $ realpath -E /tmp/regfile/  
113198 realpath: /tmp/regfile/: Not a directory
```

113199 **FUTURE DIRECTIONS**

113200 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 113201 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
 113202 format being used, implementations are encouraged to treat this as an error. A future version of  
 113203 this standard may require implementations to treat this as an error.

113204 **SEE ALSO**

113205 *ln*, *ls*, *pwd*, *readlink*

113206 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

113207 XSH Section 2.3 (on page 507), *realpath()*

113208 **CHANGE HISTORY**

113209 First released in Issue 8.

113210 **NAME**

113211 renice — set nice values of running processes

113212 **SYNOPSIS**113213 renice [-g|-p|-u] -n *increment* *ID*...113214 **DESCRIPTION**

113215 The *renice* utility shall request that the nice values (see XBD [Section 3.225](#), on page 64) of one or  
 113216 more running processes be changed. By default, the applicable processes are specified by their  
 113217 process IDs. When a process group is specified (see **-g**), the request shall apply to all processes  
 113218 in the process group.

113219 The nice value shall be bounded in an implementation-defined manner. If the requested  
 113220 *increment* would raise or lower the nice value of the executed utility beyond implementation-  
 113221 defined limits, then the limit whose value was exceeded shall be used.

113222 When a user is *reniced*, the request applies to all processes whose saved set-user-ID matches the  
 113223 user ID corresponding to the user.

113224 Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values  
 113225 of any process unless the user requesting such a change has appropriate privileges to do so for  
 113226 the specified process. If the user lacks appropriate privileges to perform the requested action, the  
 113227 utility shall return an error status.

113228 The saved set-user-ID of the user's process shall be checked instead of its effective user ID when  
 113229 *renice* attempts to determine the user ID of the process in order to determine whether the user  
 113230 has appropriate privileges.

113231 **OPTIONS**113232 The *renice* utility shall conform to XBD [Section 12.2](#) (on page 215), except for Guideline 9.

113233 The following options shall be supported:

113234 **-g** Interpret the following operands as unsigned decimal integer process group IDs.

113235 **-n** *increment* Specify how the nice value of the specified process or processes is to be adjusted.  
 113236 The *increment* option-argument is a positive or negative decimal integer that shall  
 113237 be used to modify the nice value of the specified process or processes. Negative  
 113238 *increment* values may require appropriate privileges.

113239 **-p** Interpret the following operands as unsigned decimal integer process IDs. The **-p**  
 113240 option is the default if no options are specified.

113241 **-u** Interpret the following operands as users. If a user exists with a user name equal to  
 113242 the operand, then the user ID of that user is used in further processing. Otherwise,  
 113243 if the operand represents an unsigned decimal integer, it shall be used as the  
 113244 numeric user ID of the user.

113245 **OPERANDS**

113246 The following operands shall be supported:

113247 *ID* A process ID, process group ID, or user name/user ID, depending on the option  
 113248 selected.

113249 **STDIN**

113250 Not used.

113251 **INPUT FILES**

113252 None.

113253 **ENVIRONMENT VARIABLES**113254 The following environment variables shall affect the execution of *renice*:

113255 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 113256 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 113257 variables used to determine the values of locale categories.)

113258 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 113259 internationalization variables.

113260 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 113261 characters (for example, single-byte as opposed to multi-byte characters in  
 113262 arguments).

113263 *LC\_MESSAGES*

113264 Determine the locale that should be used to affect the format and contents of  
 113265 diagnostic messages written to standard error.

113266 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

113267 **ASYNCHRONOUS EVENTS**

113268 Default.

113269 **STDOUT**

113270 Not used.

113271 **STDERR**

113272 The standard error shall be used only for diagnostic messages.

113273 **OUTPUT FILES**

113274 None.

113275 **EXTENDED DESCRIPTION**

113276 None.

113277 **EXIT STATUS**

113278 The following exit values shall be returned:

113279 0 Successful completion.

113280 &gt;0 An error occurred.

113281 **CONSEQUENCES OF ERRORS**

113282 Default.

113283 **APPLICATION USAGE**

113284 None.

113285 **EXAMPLES**

113286 1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

113287 `renice -n 5 -p 987 32`

113288 2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the  
 113289 user has appropriate privileges to do so:

113290 `renice -n -4 -g 324 76`

113291 3. Adjust the nice value so that numeric user ID 8 and user **sas** would have a lower nice  
113292 value:

```
113293 renice -n 4 -u 8 sas
```

113294 Useful nice value increments on historical systems include 19 or 20 (the affected processes run  
113295 only when nothing else in the system attempts to run) and any negative number (to make  
113296 processes run faster).

#### 113297 RATIONALE

113298 The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-  
113299 argument. However, for clarity, they have been included in the OPTIONS section, rather than  
113300 the OPERANDS section.

113301 The definition of nice value is not intended to suggest that all processes in a system have  
113302 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the  
113303 System Interfaces volume of POSIX.1-2024 make the notion of a single underlying priority for all  
113304 scheduling policies problematic. Some implementations may implement the *nice*-related features  
113305 to affect all processes on the system, others to affect just the general time-sharing activities  
113306 implied by this volume of POSIX.1-2024, and others may have no effect at all. Because of the use  
113307 of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are  
113308 possible.

113309 Originally, this utility was written in the historical manner, using the term “nice value”. This  
113310 was always a point of concern with users because it was never intuitively obvious what this  
113311 meant. With a newer version of *renice*, which used the term “system scheduling priority”, it was  
113312 hoped that novice users could better understand what this utility was meant to do. Also, it  
113313 would be easier to document what the utility was meant to do. Unfortunately, the addition of  
113314 the POSIX realtime scheduling capabilities introduced the concepts of process and thread  
113315 scheduling priorities that were totally unaffected by the *nice/renice* utilities or the  
113316 *nice()/setpriority()* functions. Continuing to use the term “system scheduling priority” would  
113317 have incorrectly suggested that these utilities and functions were indeed affecting these realtime  
113318 priorities. It was decided to revert to the historical term “nice value” to reference this unrelated  
113319 process attribute.

113320 Although this utility has use by system administrators (and in fact appears in the system  
113321 administration portion of the BSD documentation), the standard developers considered that it  
113322 was very useful for individual end users to control their own processes.

113323 Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
113324 renice nice_value[-p] pid...[-g gid...][-p pid...][-u user...]
```

```
113325 renice nice_value -g gid...[-g gid...]-p pid...[-u user...]
```

```
113326 renice nice_value -u user...[-g gid...]-p pid...[-u user...]
```

113327 These forms are no longer specified by POSIX.1-2024 but may be present in some  
113328 implementations.

#### 113329 FUTURE DIRECTIONS

113330 None.

#### 113331 SEE ALSO

113332 [nice](#)

113333 [XBD Section 3.225](#) (on page 64), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

113334 **CHANGE HISTORY**

113335 First released in Issue 4.

113336 **Issue 5**

113337 In the SYNOPSIS, an ellipsis is added to the `-u` option in all three obsolescent forms.

113338 **Issue 6**

113339 This utility is marked as part of the User Portability Utilities option.

113340 The APPLICATION USAGE section is added.

113341 The obsolescent forms of the SYNOPSIS are removed.

113342 Text previously conditional on POSIX\_SAVED\_IDS is mandatory in this version. This is a FIPS  
113343 requirement.

113344 **Issue 7**

113345 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility  
113346 Syntax Guidelines does not apply.

113347 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

113348 The *renice* utility is moved from the User Portability Utilities option to the Base. User Portability  
113349 Utilities is now an option for interactive utilities.

113350 **Issue 8**

113351 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

113352 Austin Group Defect 1286 is applied, changing the description of the `-n` option.

113353 **NAME**113354 `rm` — remove directory entries113355 **SYNOPSIS**113356 `rm [-diRrv] file...`113357 `rm -f [-diRrv] [file...]`113358 **DESCRIPTION**113359 The `rm` utility shall remove the directory entry specified by each *file* argument.

113360 If either of the files `dot` or `dot-dot` are specified as the basename portion of an operand (that is,  
113361 the final pathname component) or if an operand resolves to the root directory, `rm` shall write a  
113362 diagnostic message to standard error and do nothing more with such operands.

113363 For each *file* the following steps shall be taken:

- 113364 1. If the *file* does not exist:
  - 113365 a. If the `-f` option is not specified, `rm` shall write a diagnostic message to standard  
113366 error.
  - 113367 b. Go on to any remaining *files*.
- 113368 2. If *file* is of type directory, the following steps shall be taken:
  - 113369 a. If neither the `-R` option nor the `-r` option is specified, but `-d` is specified, `rm` shall  
113370 proceed with step 3 for the current file. If none of `-r`, `-R`, or `-d` is specified, `rm` shall  
113371 write a diagnostic message to standard error, do nothing more with *file*, and go on  
113372 to any remaining files.
  - 113373 b. If *file* is an empty directory, `rm` may skip to step 2d. If the `-f` option is not specified,  
113374 and either the permissions of *file* do not permit writing and the standard input is a  
113375 terminal or the `-i` option is specified, `rm` shall write a prompt to standard error and  
113376 read a line from the standard input. If the response is not affirmative, `rm` shall do  
113377 nothing more with the current file and go on to any remaining files.
  - 113378 c. For each entry contained in *file*, other than `dot` or `dot-dot`, the four steps listed here  
113379 (1 to 4) shall be taken with the entry as if it were a *file* operand. The `rm` utility shall  
113380 not traverse directories by following symbolic links into other parts of the  
113381 hierarchy, but shall remove the links themselves.
  - 113382 d. If the `-i` option is specified, `rm` shall write a prompt to standard error and read a  
113383 line from the standard input. If the response is not affirmative, `rm` shall do nothing  
113384 more with the current file, and go on to any remaining files.
  - 113385 e. `rm` shall proceed with step 4 for the current file.
- 113386 3. If the `-f` option is not specified, and either the permissions of *file* do not permit writing  
113387 and the standard input is a terminal or the `-i` option is specified, `rm` shall write a prompt  
113388 to the standard error and read a line from the standard input. If the response is not  
113389 affirmative, `rm` shall do nothing more with the current file and go on to any remaining  
113390 files.
- 113391 4. `rm` shall perform actions equivalent to the `remove()` function defined in the System  
113392 Interfaces volume of POSIX.1-2024 called with a pathname of the current file used as the  
113393 *path* argument.

113394 If `rm` successfully performed the above actions on the current file, and the `-v` option is  
113395 specified, `rm` shall write a message containing the pathname of the current file to the  
113396 standard output. If the actions fail for any reason, `rm` shall write a diagnostic message to

- 113397 standard error, do nothing more with the current file, and go on to any remaining files.
- 113398 The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail  
 113399 due to path length limitations (unless an operand specified by the user exceeds system  
 113400 limitations).
- 113401 **OPTIONS**
- 113402 The *rm* utility shall conform to XBD [Section 12.2](#) (on page 215).
- 113403 The following options shall be supported:
- 113404 **-d** Remove empty directories. See the DESCRIPTION.
- 113405 **-f** Do not prompt for confirmation. Do not write diagnostic messages or modify the  
 113406 exit status in the case of no file operands, or in the case of operands that do not  
 113407 exist. Any previous occurrences of the **-i** option shall be ignored.
- 113408 **-i** Prompt for confirmation as described previously. Any previous occurrences of the  
 113409 **-f** option shall be ignored.
- 113410 **-R** Remove file hierarchies. See the DESCRIPTION.
- 113411 **-r** Equivalent to **-R**.
- 113412 **-v** After each file has been removed, write a message to standard output indicating  
 113413 that it has been removed.
- 113414 **OPERANDS**
- 113415 The following operand shall be supported:
- 113416 *file* A pathname of a directory entry to be removed.
- 113417 **STDIN**
- 113418 The standard input shall be used to read an input line in response to each prompt specified in  
 113419 the STDOUT section. Otherwise, the standard input shall not be used.
- 113420 **INPUT FILES**
- 113421 None.
- 113422 **ENVIRONMENT VARIABLES**
- 113423 The following environment variables shall affect the execution of *rm*:
- 113424 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 113425 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 113426 variables used to determine the values of locale categories.)
- 113427 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 113428 internationalization variables.
- 113429 **LC\_COLLATE**
- 113430 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 113431 character collating elements used in the extended regular expression defined for  
 113432 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.
- 113433 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 113434 characters (for example, single-byte as opposed to multi-byte characters in  
 113435 arguments) and the behavior of character classes within regular expressions used  
 113436 in the extended regular expression defined for the **yesexpr** locale keyword in the  
 113437 *LC\_MESSAGES* category.

- 113438 *LC\_MESSAGES*
- 113439 Determine the locale used to process affirmative responses, and the locale used to
- 113440 affect the format and contents of diagnostic messages and prompts written to
- 113441 standard error.
- 
- 113442 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 113443 **ASYNCHRONOUS EVENTS**
- 113444 Default.
- 113445 **STDOUT**
- 113446 If the `-v` option is specified, information about each removed file shall be written to standard
- 113447 output in an unspecified format.
- 113448 **STDERR**
- 113449 Prompts shall be written to standard error under the conditions specified in the **DESCRIPTION**
- 113450 and **OPTIONS** sections. The prompts shall contain the *file* pathname, but their format is
- 113451 otherwise unspecified. The standard error also shall be used for diagnostic messages.
- 113452 **OUTPUT FILES**
- 113453 None.
- 113454 **EXTENDED DESCRIPTION**
- 113455 None.
- 113456 **EXIT STATUS**
- 113457 The following exit values shall be returned:
- 113458 0 All requested directory entries (excluding directory entries where a non-affirmative
- 113459 response was given to a request for confirmation) were successfully deleted. In addition, if
- 113460 the `-v` option is specified, information about each removed directory entry was successfully
- 113461 written to standard output.
- 113462 >0 An error occurred.
- 113463 **CONSEQUENCES OF ERRORS**
- 113464 Default.
- 113465 **APPLICATION USAGE**
- 113466 The *rm* utility is forbidden to remove the names `dot` and `dot-dot` in order to avoid the
- 113467 consequences of inadvertently doing something like:
- 113468 `rm -r .*`
- 113469 Some implementations do not permit the removal of the last hard link to an executable binary
- 113470 file that is being executed; see the [EBUSY] error in the *unlink()* function defined in the System
- 113471 Interfaces volume of POSIX.1-2024. Thus, the *rm* utility can fail to remove such files.
- 113472 The `-i` option causes *rm* to prompt and read the standard input even if the standard input is not
- 113473 a terminal, but in the absence of `-i` the mode prompting is not done when the standard input is
- 113474 not a terminal.
- 113475 **EXAMPLES**
- 113476 1. The following command:
- 113477 `rm a.out core`
- 113478 removes the directory entries: **a.out** and **core**, or issues an error if either directory entry is
- 113479 itself a directory or does not exist.



- 113480           2. The following command:
- 113481            `rm -Rf junk`
- 113482           removes the directory **junk** and all its contents, without prompting.
- 113483           3. The following command:
- 113484            `rm -d name`
- 113485           behaves like
- 113486            `rmdir name`
- 113487           if **name** is a directory (including failing if **name** is not empty), as if by the `rmdir()`
- 113488           function; and behaves like
- 113489            `rm name`
- 113490           if **name** is not a directory, as if by the `unlink()` function.

#### 113491 RATIONALE

113492           For absolute clarity, paragraphs (2b) and (3) in the DESCRIPTION of *rm* describing the behavior

113493           when prompting for confirmation, should be interpreted in the following manner:

```
113494           if ((NOT f_option) AND
113495            ((not_writable AND input_is_terminal) OR i_option))
```

113496           The exact format of the interactive prompts is unspecified. Only the general nature of the

113497           contents of prompts are specified because implementations may desire more descriptive

113498           prompts than those used on historical implementations. Therefore, an application not using the

113499           **-f** option, or using the **-i** option, relies on the system to provide the most suitable dialog directly

113500           with the user, based on the behavior specified.

113501           The **-r** option is historical practice on all known systems. The synonym **-R** option is provided

113502           for consistency with the other utilities in this volume of POSIX.1-2024 that provide options

113503           requesting recursive descent through the file hierarchy.

113504           The behavior of the **-f** option in historical versions of *rm* is inconsistent. In general, along with

113505           “forcing” the unlink without prompting for permission, it always causes diagnostic messages to

113506           be suppressed and the exit status to be unmodified for nonexistent operands and files that

113507           cannot be unlinked. In some versions, however, the **-f** option suppresses usage messages and

113508           system errors as well. Suppressing such messages is not a service to either shell scripts or users.

113509           It is less clear that error messages regarding files that cannot be unlinked (removed) should be

113510           suppressed. Although this is historical practice, this volume of POSIX.1-2024 does not permit the

113511           **-f** option to suppress such messages.

113512           When given the **-r** and **-i** options, historical versions of *rm* prompt the user twice for each

113513           directory, once before removing its contents and once before actually attempting to delete the

113514           directory entry that names it. This allows the user to “prune” the file hierarchy walk. Historical

113515           versions of *rm* were inconsistent in that some did not do the former prompt for directories

113516           named on the command line and others had obscure prompting behavior when the **-i** option

113517           was specified and the permissions of the file did not permit writing. The POSIX Shell and

113518           Utilities *rm* differs little from historic practice, but does require that prompts be consistent.

113519           Historical versions of *rm* were also inconsistent in that prompts were done to both standard

113520           output and standard error. This volume of POSIX.1-2024 requires that prompts be done to

113521           standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that

113522           provide an option to list deleted files on standard output.

113523 The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be  
113524 deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its  
113525 descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the  
113526 historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted  
113527 to fail because of path length restrictions, unless an operand specified by the user is longer than  
113528 {PATH\_MAX}.

113529 The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of  
113530 the dependence on the *unlink()* functionality, per the DESCRIPTION. When removing  
113531 hierarchies with *-r* or *-R*, the prohibition on following symbolic links has to be made explicit.

113532 The addition of the *-d* option allows the use of *rm* to delete either a file or an empty directory  
113533 without the risk of recursion into a non-empty directory, and without the inherent race between  
113534 determining a file's type and deciding what action to attempt on that file. If either the *-r* or *-R*  
113535 option is specified, the use of recursion takes precedence.

113536 The addition of the *-v* option allows a user of *rm* to see which files have been deleted.

#### 113537 FUTURE DIRECTIONS

113538 If this utility is directed to display a pathname that contains any bytes that have the encoded  
113539 value of a <newline> character when <newline> is a terminator or separator in the output  
113540 format being used, implementations are encouraged to treat this as an error. A future version of  
113541 this standard may require implementations to treat this as an error.

#### 113542 SEE ALSO

113543 *rmdir*

113544 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

113545 XSH *remove()*, *rmdir()*, *unlink()*

#### 113546 CHANGE HISTORY

113547 First released in Issue 2.

#### 113548 Issue 5

113549 The FUTURE DIRECTIONS section is added.

#### 113550 Issue 6

113551 Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft  
113552 standard.

#### 113553 Issue 7

113554 Austin Group Interpretations 1003.1-2001 #019 and #091 are applied.

113555 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
113556 *LC\_MESSAGES* environment variable.

113557 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0163 [542], XCU/TC2-2008/0164  
113558 [819], and XCU/TC2-2008/0165 [542] are applied.

#### 113559 Issue 8

113560 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
113561 directed to display a pathname that contains any bytes that have the encoded value of a  
113562 <newline> character when <newline> is a terminator or separator in the output format being  
113563 used.

113564 Austin Group Defect 802 is applied, adding the *-d* option.

113565 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

- 113566 Austin Group Defects 1154, 1365, and 1487 are applied, adding the `-v` option.
- 113567 Austin Group Defect 1380 is applied, changing ```last link``` to ```last hard link```.
- 113568 Austin Group Defect 1732 is applied, changing the EXIT STATUS section.

113569 **NAME**113570 rmdel — remove a delta from an SCCS file (**DEVELOPMENT**)113571 **SYNOPSIS**113572 XSI `rmdel -r SID file...`113573 **DESCRIPTION**

113574 The *rmdel* utility shall remove the delta specified by the *SID* from each named SCCS file. The  
 113575 delta to be removed shall be the most recent delta in its branch in the delta chain of each named  
 113576 SCCS file. In addition, the application shall ensure that the *SID* specified is not that of a version  
 113577 being edited for the purpose of making a delta; that is, if a *p-file* (see *get*) exists for the named  
 113578 SCCS file, the *SID* specified shall not appear in any entry of the *p-file*.

113579 Removal of a delta shall be restricted to:

- 113580 1. The user who made the delta
- 113581 2. The owner of the SCCS file
- 113582 3. The owner of the directory containing the SCCS file

113583 **OPTIONS**113584 The *rmdel* utility shall conform to XBD [Section 12.2](#) (on page 215).

113585 The following option shall be supported:

113586 **-r** *SID* Specify the SCCS identification string (*SID*) of the delta to be deleted.113587 **OPERANDS**

113588 The following operand shall be supported:

113589 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *rmdel*  
 113590 utility shall behave as though each file in the directory were specified as a named  
 113591 file, except that non-SCCS files (last component of the pathname does not begin  
 113592 with **s**.) and unreadable files shall be silently ignored.

113593 If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
 113594 each line of the standard input is taken to be the name of an SCCS file to be  
 113595 processed. Non-SCCS files and unreadable files shall be silently ignored.

113596 **STDIN**

113597 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
 113598 line of the text file shall be interpreted as an SCCS pathname.

113599 **INPUT FILES**

113600 The SCCS files shall be files of unspecified format.

113601 **ENVIRONMENT VARIABLES**113602 The following environment variables shall affect the execution of *rmdel*:

113603 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 113604 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 113605 variables used to determine the values of locale categories.)

113606 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 113607 internationalization variables.

113608 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 113609 characters (for example, single-byte as opposed to multi-byte characters in  
 113610 arguments and input files).

- 113611 **LC\_MESSAGES**
- 113612 Determine the locale that should be used to affect the format and contents of
- 113613 diagnostic messages written to standard error.
- 113614 **NLSPATH** Determine the location of messages objects and message catalogs.
- 113615 **ASYNCHRONOUS EVENTS**
- 113616 Default.
- 113617 **STDOUT**
- 113618 Not used.
- 113619 **STDERR**
- 113620 The standard error shall be used only for diagnostic messages.
- 113621 **OUTPUT FILES**
- 113622 The SCCS files shall be files of unspecified format. During processing of a *file*, a temporary *x-file*,
- 113623 as described in *admin*, may be created and deleted; a locking *z-file*, as described in *get*, may be
- 113624 created and deleted.
- 113625 **EXTENDED DESCRIPTION**
- 113626 None.
- 113627 **EXIT STATUS**
- 113628 The following exit values shall be returned:
- 113629 0 Successful completion.
- 113630 >0 An error occurred.
- 113631 **CONSEQUENCES OF ERRORS**
- 113632 Default.
- 113633 **APPLICATION USAGE**
- 113634 None.
- 113635 **EXAMPLES**
- 113636 None.
- 113637 **RATIONALE**
- 113638 None.
- 113639 **FUTURE DIRECTIONS**
- 113640 If this utility is directed to create a new directory entry that contains any bytes that have the
- 113641 encoded value of a <newline> character, implementations are encouraged to treat this as an
- 113642 error. A future version of this standard may require implementations to treat this as an error.
- 113643 **SEE ALSO**
- 113644 *admin*, *delta*, *get*, *prs*
- 113645 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)
- 113646 **CHANGE HISTORY**
- 113647 First released in Issue 2.
- 113648 **Issue 6**
- 113649 The normative text is reworded to avoid use of the term “must” for application requirements.

113650 **Issue 8**

113651 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
113652 filenames containing any bytes that have the encoded value of a <newline> character.

113653 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

113654 **NAME**

113655           rmdir — remove directories

113656 **SYNOPSIS**113657           rmdir [-p] *dir*...113658 **DESCRIPTION**113659           The *rmdir* utility shall remove the directory entry specified by each *dir* operand.113660           For each *dir* operand, the *rmdir* utility shall perform actions equivalent to the *rmdir()* function  
113661           called with the *dir* operand as its only argument.113662           Directories shall be processed in the order specified. If a directory and a subdirectory of that  
113663           directory are specified in a single invocation of the *rmdir* utility, the application shall specify the  
113664           subdirectory before the parent directory so that the parent directory is empty when the *rmdir*  
113665           utility tries to remove it.113666 **OPTIONS**113667           The *rmdir* utility shall conform to XBD [Section 12.2](#) (on page 215).

113668           The following option shall be supported:

113669           **-p**           Remove all directories in a pathname. For each *dir* operand:

- 113670                           1. The directory entry it names shall be removed.
- 
- 113671                           2. If the
- dir*
- operand includes more than one pathname component, effects
- 
- 113672                           equivalent to the following command shall occur:

113673                           rmdir -p \$(dirname *dir*)113674 **OPERANDS**

113675           The following operand shall be supported:

113676           *dir*           A pathname of an empty directory to be removed.113677 **STDIN**

113678           Not used.

113679 **INPUT FILES**

113680           None.

113681 **ENVIRONMENT VARIABLES**113682           The following environment variables shall affect the execution of *rmdir*:113683           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
113684                           (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
113685                           variables used to determine the values of locale categories.)113686           **LC\_ALL**          If set to a non-empty string value, override the values of all the other  
113687                           internationalization variables.113688           **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
113689                           characters (for example, single-byte as opposed to multi-byte characters in  
113690                           arguments).113691           **LC\_MESSAGES**113692                           Determine the locale that should be used to affect the format and contents of  
113693                           diagnostic messages written to standard error.

- 113694 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 113695 **ASYNCHRONOUS EVENTS**
- 113696 Default.
- 113697 **STDOUT**
- 113698 Not used.
- 113699 **STDERR**
- 113700 The standard error shall be used only for diagnostic messages.
- 113701 **OUTPUT FILES**
- 113702 None.
- 113703 **EXTENDED DESCRIPTION**
- 113704 None.
- 113705 **EXIT STATUS**
- 113706 The following exit values shall be returned:
- 113707 0 Each directory entry specified by a *dir* operand was removed successfully.
- 113708 >0 An error occurred.
- 113709 **CONSEQUENCES OF ERRORS**
- 113710 Default.
- 113711 **APPLICATION USAGE**
- 113712 The definition of an empty directory is one that contains, at most, directory entries for dot and dot-dot.
- 113713
- 113714 **EXAMPLES**
- 113715 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty
- 113716 except it contains a directory **c**:
- 113717 `rmdir -p a/b/c`
- 113718 removes all three directories.
- 113719 **RATIONALE**
- 113720 On historical System V systems, the `-p` option also caused a message to be written to the
- 113721 standard output. The message indicated whether the whole path was removed or whether part
- 113722 of the path remained for some reason. The **STDERR** section requires this diagnostic when the
- 113723 entire path specified by a *dir* operand is not removed, but does not allow the status message
- 113724 reporting success to be written as a diagnostic.
- 113725 The *rmdir* utility on System V also included a `-s` option that suppressed the informational
- 113726 message output by the `-p` option. This option has been omitted because the informational
- 113727 message is not specified by this volume of POSIX.1-2024.
- 113728 **FUTURE DIRECTIONS**
- 113729 None.
- 113730 **SEE ALSO**
- 113731 *rm*
- 113732 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)
- 113733 XSH *remove()*, *rmdir()*, *unlink()*



113734 **CHANGE HISTORY**

113735 First released in Issue 2.

113736 **Issue 6**

113737 The normative text is reworded to avoid use of the term “must” for application requirements.

113738 **Issue 8**

113739 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

113740 **NAME**113741 sact — print current SCCS file-editing activity (**DEVELOPMENT**)113742 **SYNOPSIS**113743 XSI `sact file...`113744 **DESCRIPTION**

113745 The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a  
 113746 list to standard output. This situation occurs when *get -e* has been executed previously without  
 113747 a subsequent execution of *delta*, *unget*, or *sccs unedit*.

113748 **OPTIONS**

113749 None.

113750 **OPERANDS**

113751 The following operand shall be supported:

113752 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *sact*  
 113753 utility shall behave as though each file in the directory were specified as a named  
 113754 file, except that non-SCCS files (last component of the pathname does not begin  
 113755 with **s**.) and unreadable files shall be silently ignored.

113756 If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
 113757 each line of the standard input shall be taken to be the name of an SCCS file to be  
 113758 processed. Non-SCCS files and unreadable files shall be silently ignored.

113759 **STDIN**

113760 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
 113761 line of the text file shall be interpreted as an SCCS pathname.

113762 **INPUT FILES**

113763 Any SCCS files interrogated are files of an unspecified format.

113764 **ENVIRONMENT VARIABLES**113765 The following environment variables shall affect the execution of *sact*:

113766 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 113767 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 113768 variables used to determine the values of locale categories.)

113769 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 113770 internationalization variables.

113771 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 113772 characters (for example, single-byte as opposed to multi-byte characters in  
 113773 arguments and input files).

113774 *LC\_MESSAGES*

113775 Determine the locale that should be used to affect the format and contents of  
 113776 diagnostic messages written to standard error.

113777 *NLSPATH* Determine the location of messages objects and message catalogs.

113778 **ASYNCHRONOUS EVENTS**

113779 Default.

**113780 STDOUT**

113781 The output for each named file shall consist of a line in the following format:

113782 "%sΔ%sΔ%sΔ%sΔ%s\n", <SID>, <new SID>, <login>, <date>, <time>

113783 <SID> Specifies the SID of a delta that currently exists in the SCCS file to which changes  
113784 are made to make the new delta.

113785 <new SID> Specifies the SID for the new delta to be created.

113786 <login> Contains the login name of the user who makes the delta (that is, who executed a  
113787 *get* for editing).

113788 <date> Contains the date that *get -e* was executed, in the format used by the *prs :D:* data  
113789 keyword.

113790 <time> Contains the time that *get -e* was executed, in the format used by the *prs :T:* data  
113791 keyword.

113792 If there is more than one named file or if a directory or standard input is named, each pathname  
113793 shall be written before each of the preceding lines:

113794 "\n%s:\n", <pathname>

**113795 STDERR**

113796 The standard error shall be used only for optional informative messages concerning SCCS files  
113797 with no impending deltas, and for diagnostic messages.

**113798 OUTPUT FILES**

113799 None.

**113800 EXTENDED DESCRIPTION**

113801 None.

**113802 EXIT STATUS**

113803 The following exit values shall be returned:

113804 0 Successful completion.

113805 >0 An error occurred.

**113806 CONSEQUENCES OF ERRORS**

113807 Default.

**113808 APPLICATION USAGE**

113809 None.

**113810 EXAMPLES**

113811 None.

**113812 RATIONALE**

113813 None.

**113814 FUTURE DIRECTIONS**

113815 If this utility is directed to display a pathname that contains any bytes that have the encoded  
113816 value of a <newline> character when <newline> is a terminator or separator in the output  
113817 format being used, implementations are encouraged to treat this as an error. A future version of  
113818 this standard may require implementations to treat this as an error.

113819 **SEE ALSO**113820 *delta, get, sccs, unget*

113821 XBD Chapter 8 (on page 167)

113822 **CHANGE HISTORY**

113823 First released in Issue 2.

113824 **Issue 8**

113825 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
113826 directed to display a pathname that contains any bytes that have the encoded value of a  
113827 <newline> character when <newline> is a terminator or separator in the output format being  
113828 used.

113829 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

113830 **NAME**113831 `sccs` — front end for the SCCS subsystem (**DEVELOPMENT**)113832 **SYNOPSIS**113833 XSI `sccs [-r] [-d path] [-p path] command [options...] [operands...]`113834 **DESCRIPTION**113835 The `sccs` utility is a front end to the SCCS programs. It also includes the capability to run set-  
113836 user-id to another user to provide additional protection.113837 The `sccs` utility shall invoke the specified *command* with the specified *options* and *operands*. By  
113838 default, each of the *operands* shall be modified by prefixing it with the string "SCCS/s.". 113839 The *command* can be the name of one of the SCCS utilities in this volume of POSIX.1-2024 (*admin*,  
113840 *delta*, *get*, *prs*, *rmDEL*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the  
113841 EXTENDED DESCRIPTION section.113842 **OPTIONS**113843 The `sccs` utility shall conform to XBD [Section 12.2](#) (on page 215), except that *options* operands are  
113844 actually options to be passed to the utility named by *command*. When the portion of the  
113845 command:113846 `command [options ... ] [operands ... ]`113847 is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax  
113848 Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the  
113849 Guidelines to the extent indicated by their individual OPTIONS sections.113850 The following options shall be supported preceding the *command* operand:113851 **-d path** A pathname of a directory to be used as a root directory for the SCCS files. The  
113852 default shall be the current directory. The **-d** option shall take precedence over the  
113853 *PROJECTDIR* variable. See **-p**.113854 **-p path** A pathname of a directory in which the SCCS files are located. The default shall be  
113855 the **SCCS** directory.113856 The **-p** option differs from the **-d** option in that the **-d** option-argument shall be  
113857 prefixed to the entire pathname and the **-p** option-argument shall be inserted  
113858 before the final component of the pathname. For example:113859 `sccs -d /x -p y get a/b`

113860 converts to:

113861 `get /x/a/y/s.b`

113862 This allows the creation of aliases such as:

113863 `alias syssccs="sccs -d /usr/src"`

113864 which is used as:

113865 `syssccs get cmd/who.c`113866 **-r** Invoke *command* with the real user ID of the process, not any effective user ID that  
113867 the `sccs` utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmDEL*,  
113868 and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to  
113869 change the authorizations. These commands are always run as the real user.

113870 **OPERANDS**

113871 The following operands shall be supported:

113872 *command* An SCCS utility name or the name of one of the pseudo-utilities listed in the  
113873 EXTENDED DESCRIPTION section.

113874 *options* An option or option-argument to be passed to *command*.

113875 *operands* An operand to be passed to *command*.

113876 **STDIN**

113877 See the utility description for the specified *command*.

113878 **INPUT FILES**

113879 See the utility description for the specified *command*.

113880 **ENVIRONMENT VARIABLES**

113881 The following environment variables shall affect the execution of *sccs*:

113882 *LANG* Provide a default value for the internationalization variables that are unset or null.  
113883 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
113884 variables used to determine the values of locale categories.)

113885 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
113886 internationalization variables.

113887 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
113888 characters (for example, single-byte as opposed to multi-byte characters in  
113889 arguments and input files).

113890 *LC\_MESSAGES*

113891 Determine the locale that should be used to affect the format and contents of  
113892 diagnostic messages written to standard error.

113893 *NLSPATH* Determine the location of messages objects and message catalogs.

113894 *PROJECTDIR*

113895 Provide a default value for the *-d path* option. If the value of *PROJECTDIR* begins  
113896 with a <slash>, it shall be considered an absolute pathname; otherwise, the value  
113897 of *PROJECTDIR* is treated as a user name and that user's initial working directory  
113898 shall be examined for a subdirectory *src* or *source*. If such a directory is found, it  
113899 shall be used. Otherwise, the value shall be used as a relative pathname.

113900 Additional environment variable effects may be found in the utility description for the specified  
113901 *command*.

113902 **ASYNCHRONOUS EVENTS**

113903 Default.

113904 **STDOUT**

113905 See the utility description for the specified *command*.

113906 **STDERR**

113907 See the utility description for the specified *command*.

113908 **OUTPUT FILES**

113909 See the utility description for the specified *command*.

## 113910 EXTENDED DESCRIPTION

- 113911 The following pseudo-utilities shall be supported as *command* operands. All options referred to  
113912 in the following list are values given in the *options* operands following *command*.
- 113913 **check** Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a  
113914 non-zero exit status shall be returned if anything is being edited. The intent is to have  
113915 this included in an ``install'' entry in a makefile to ensure that everything is included  
113916 into the SCCS file before a version is installed.
- 113917 **clean** Remove everything from the current directory that can be recreated from SCCS files,  
113918 but do not remove any files being edited. If the **-b** option is given, branches shall be  
113919 ignored in the determination of whether they are being edited; this is dangerous if  
113920 branches are kept in the same directory.
- 113921 **create** Create an SCCS file, taking the initial contents from the file of the same name. Any  
113922 options to *admin* are accepted. If the creation is successful, the original files shall be  
113923 renamed by prefixing the basenames with a comma. These renamed files should be  
113924 removed after it has been verified that the SCCS files have been created successfully.
- 113925 **delget** Perform a *delta* on the named files and then *get* new versions. The new versions shall  
113926 have ID keywords expanded and shall not be editable. Any **-m**, **-p**, **-r**, **-s**, and **-y**  
113927 options shall be passed to *delta*, and any **-b**, **-c**, **-e**, **-i**, **-k**, **-l**, **-s**, and **-x** options shall be  
113928 passed to *get*.
- 113929 **deledit** Equivalent to **delget**, except that the *get* phase shall include the **-e** option. This option is  
113930 useful for making a checkpoint of the current editing phase. The same options shall be  
113931 passed to *delta* as described above, and all the options listed for *get* above except **-e**  
113932 shall be passed to **edit**.
- 113933 **diffs** Write a difference listing between the current version of the files checked out for editing  
113934 and the versions in SCCS format. Any **-r**, **-c**, **-i**, **-x**, and **-t** options shall be passed to  
113935 *get*; any **-l**, **-s**, **-e**, **-f**, **-h**, and **-b** options shall be passed to *diff*. A **-C** option shall be  
113936 passed to *diff* as **-c**.
- 113937 **edit** Equivalent to *get -e*.
- 113938 **fix** Remove the named delta, but leave a copy of the delta with the changes that were in it.  
113939 It is useful for fixing small compiler bugs, and so on. The application shall ensure that it  
113940 is followed by a **-r** *SID* option. Since **fix** does not leave audit trails, it should be used  
113941 carefully.
- 113942 **info** Write a listing of all files being edited. If the **-b** option is given, branches (that is, SIDs  
113943 with two or fewer components) shall be ignored. If a **-u** *user* option is given, then only  
113944 files being edited by the named user shall be listed. A **-U** option shall be equivalent to  
113945 **-u**<*current user*>.
- 113946 **print** Write out verbose information about the named files, equivalent to *sccs prs*.
- 113947 **tell** Write a <newline>-separated list of the files being edited to standard output. Takes the  
113948 **-b**, **-u**, and **-U** options like **info** and **check**.
- 113949 **unedit** This is the opposite of an **edit** or a *get -e*. It should be used with caution, since any  
113950 changes made since the *get* are lost.

113951 **EXIT STATUS**

113952 The following exit values shall be returned:

113953 0 Successful completion.

113954 &gt;0 An error occurred.

113955 **CONSEQUENCES OF ERRORS**

113956 Default.

113957 **APPLICATION USAGE**

113958 Many of the SCCS utilities take directory names as operands as well as specific filenames. The  
 113959 pseudo-utilities supported by *sccs* are not described as having this capability, but are not  
 113960 prohibited from doing so.

113961 **EXAMPLES**

113962 1. To get a file for editing, edit it and produce a new delta:

113963 `sccs get -e file.c`113964 `ex file.c`113965 `sccs delta file.c`

113966 2. To get a file from another directory:

113967 `sccs -p /usr/src/sccs/s. get cc.c`

113968 or:

113969 `sccs get /usr/src/sccs/s.cc.c`

113970 3. To make a delta of a large number of files in the current directory:

113971 `sccs delta *.c`

113972 4. To get a list of files being edited that are not on branches:

113973 `sccs info -b`

113974 5. To delta everything being edited by the current user:

113975 `sccs delta $(sccs tell -U)`

113976 6. In a makefile, to get source files from an SCCS file if it does not already exist:

113977 `SRCS = <list of source files>`113978 `$(SRCS) :`113979 `sccs get $(REL) $@`113980 **RATIONALE**

113981 *sccs* and its associated utilities are part of the XSI Development Utilities option within the XSI  
 113982 option.

113983 SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement  
 113984 tracking tool. When a file is put under SCCS, the source code control system maintains the file  
 113985 and, when changes are made, identifies and stores them in the file with the original source code  
 113986 and/or documentation. As other changes are made, they too are identified and retained in the  
 113987 file.

113988 Retrieval of the original and any set of changes is possible. Any version of the file as it develops  
 113989 can be reconstructed for inspection or additional modification. History data can be stored with  
 113990 each version, documenting why the changes were made, who made them, and when they were  
 113991 made.



**113992 FUTURE DIRECTIONS**

113993 None.

**113994 SEE ALSO**

113995 *admin, delta, get, make, prs, rmdel, sact, unget, val, what*

113996 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

**113997 CHANGE HISTORY**

113998 First released in Issue 4.

**113999 Issue 6**

114000 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from  
114001 ``otherwise, the home directory of a user of that name is examined'' to ``otherwise, the value of  
114002 *PROJECTDIR* is treated as a user name and that user's initial working directory is examined''.

114003 The normative text is reworded to avoid use of the term ``must'' for application requirements.

**114004 Issue 7**

114005 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

**114006 Issue 8**

114007 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

114008 **NAME**

114009 sed — stream editor

114010 **SYNOPSIS**114011 sed [-En] *script* [*file...*]114012 sed [-En] -e *script* [-e *script*]... [-f *script\_file*]... [*file...*]114013 sed [-En] [-e *script*]... -f *script\_file* [-f *script\_file*]... [*file...*]114014 **DESCRIPTION**

114015 The *sed* utility is a stream editor that shall read one or more text files, make editing changes  
 114016 according to a script of editing commands, and write the results to standard output. The script  
 114017 shall be obtained from either the *script* operand string or a combination of the option-arguments  
 114018 from the -e *script* and -f *script\_file* options.

114019 **OPTIONS**

114020 The *sed* utility shall conform to XBD Section 12.2 (on page 215), except that the order of  
 114021 presentation of the -e and -f options is significant.

114022 The following options shall be supported:

114023 -E Match using extended regular expressions. Treat each pattern specified as an ERE,  
 114024 as described in XBD Section 9.4 (on page 187).

114025 -e *script* Add the editing commands specified by the *script* option-argument to the end of  
 114026 the script of editing commands.

114027 -f *script\_file* Add the editing commands in the file *script\_file* to the end of the script of editing  
 114028 commands.

114029 -n Suppress the default output (in which each line, after it is examined for editing, is  
 114030 written to standard output). Only lines explicitly selected for output are written.

114031 If any -e or -f options are specified, the script of editing commands shall initially be empty. The  
 114032 commands specified by each -e or -f option shall be added to the script in the order specified.  
 114033 When each addition is made, if the previous addition (if any) was from a -e option, a <newline>  
 114034 shall be inserted before the new addition. The resulting script shall have the same properties as  
 114035 the *script* operand, described in the OPERANDS section.

114036 **OPERANDS**

114037 The following operands shall be supported:

114038 *file* A pathname of a file whose contents are read and edited. If multiple *file* operands  
 114039 are specified, the named files shall be read in the order specified and the  
 114040 concatenation shall be edited. If no *file* operands are specified, the standard input  
 114041 shall be used.

114042 *script* A string to be used as the script of editing commands. The application shall not  
 114043 present a *script* that violates the restrictions of a text file except that the final  
 114044 character need not be a <newline>.

114045 **STDIN**

114046 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
 114047 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
 114048 the standard input shall not be used. See the INPUT FILES section.

114049 **INPUT FILES**

114050 The input files shall be text files. The *script\_files* named by the `-f` option shall consist of editing  
114051 commands.

114052 **ENVIRONMENT VARIABLES**

114053 The following environment variables shall affect the execution of *sed*:

114054 *LANG* Provide a default value for the internationalization variables that are unset or null.  
114055 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
114056 variables used to determine the values of locale categories.)

114057 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
114058 internationalization variables.

114059 *LC\_COLLATE*

114060 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
114061 character collating elements within regular expressions.

114062 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
114063 characters (for example, single-byte as opposed to multi-byte characters in  
114064 arguments and input files), and the behavior of character classes within regular  
114065 expressions.

114066 *LC\_MESSAGES*

114067 Determine the locale that should be used to affect the format and contents of  
114068 diagnostic messages written to standard error.

114069 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

114070 **ASYNCHRONOUS EVENTS**

114071 Default.

114072 **STDOUT**

114073 The input files shall be written to standard output, with the editing commands specified in the  
114074 script applied. If the `-n` option is specified, only those input lines selected by the script shall be  
114075 written to standard output.

114076 **STDERR**

114077 The standard error shall be used only for diagnostic and warning messages.

114078 **OUTPUT FILES**

114079 The output files shall be text files whose formats are dependent on the editing commands given.

114080 **EXTENDED DESCRIPTION**

114081 The *script* shall consist of editing commands of the following form:

114082 `[address[, address]] function`

114083 where *function* represents a single-character command verb from the list in [Editing Commands](#)  
114084 [in sed](#) (on page 3357), followed by any applicable arguments.

114085 The command can be preceded by <blank> characters and/or <semicolon> characters. The  
114086 function can be preceded by <blank> characters. These optional characters shall have no effect.

114087 In default operation, *sed* cyclically shall append a line of input, less its terminating <newline>  
114088 character, into the pattern space. Reading from input shall be skipped if a <newline> was in the  
114089 pattern space prior to a **D** command ending the previous cycle. The *sed* utility shall then apply in  
114090 sequence all commands whose addresses select that pattern space, until a command starts the  
114091 next cycle or quits. If no commands explicitly started a new cycle, then at the end of the script  
114092 the pattern space shall be copied to standard output (except when `-n` is specified) and the

114093 pattern space shall be deleted. Whenever the pattern space is written to standard output or a  
114094 named file, *sed* shall immediately follow it with a <newline>.

114095 Some of the editing commands use a hold space to save all or part of the pattern space for  
114096 subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

### 114097 **Addresses in sed**

114098 An address is either a decimal number that counts input lines cumulatively across files, a '\$'  
114099 character that addresses the last line of input, or a context address. A context address has either  
114100 the form "/RE/" or "\cREc", where RE is a regular expression as described in [Regular](#)  
114101 [Expressions in sed](#), and *c* is any character other than <backslash> or <newline>. In a *sed* context  
114102 address, the BRE and ERE syntax shall be extended to support escaping occurrences of the  
114103 <slash> or *c* delimiter within the RE by means of an escape sequence (see [XBD Section 9.1](#), on  
114104 page 179). For the "\cREc" form, if the character designated by *c* is not listed as a special BRE  
114105 character (if the -E option is not specified) or a special ERE character (if -E is specified) in [XBD](#)  
114106 [Section 9.3.3](#) (on page 182) or [XBD Section 9.4.3](#) (on page 188), respectively, the escape sequence  
114107 <backslash>*c* shall be treated as that literal character; otherwise, it is unspecified whether the  
114108 escape sequence <backslash>*c* is treated as the literal character or the special character. In either  
114109 case, the escape sequence <backslash>*c* shall not terminate the RE. For example, in the context  
114110 address "/abc/def/", the second <slash> stands for itself, so that the RE is "abc/def", and  
114111 in "\xabc\xdefx", the second 'x' stands for itself, so that the RE is "abcxdef".

114112 An editing command with no addresses shall select every pattern space.

114113 An editing command with one address shall select each pattern space that matches the address.

114114 An editing command with two addresses shall select the inclusive range from the first pattern  
114115 space that matches the first address through the next pattern space that matches the second. (If  
114116 the second address is a number less than or equal to the line number first selected, only one line  
114117 shall be selected.) Starting at the first line following the selected range, *sed* shall look again for  
114118 the first address. Thereafter, the process shall be repeated. Omitting either or both of the address  
114119 components in the following form produces undefined results:

114120 `[address[, address]]`

### 114121 **Regular Expressions in sed**

114122 The *sed* utility shall support the REs described in [XBD Chapter 9](#) (on page 179); by default it shall  
114123 use BREs as described in [XBD Section 9.3](#) (on page 181), but if the -E option is used, it shall use  
114124 EREs as described in [XBD Section 9.4](#) (on page 187). In *sed*, the BRE and ERE syntax shall be  
114125 extended as follows:

- 114126 • The delimiter character that precedes and follows the RE shall not terminate the RE when  
114127 it appears within a bracket expression, and shall have its normal meaning in the bracket  
114128 expression. For example, the context address "\% [%] %" is equivalent to "[ % ] /", and the  
114129 command "s-[0-9]--g" is equivalent to "s/[0-9]//g".
- 114130 • The escape sequence '\n' shall match a <newline> embedded in the pattern space. A  
114131 literal <newline> shall not be used in the RE of a context address or in the substitute  
114132 function.
- 114133 • If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in  
114134 the last command applied (either as an address or as part of a substitute command) was  
114135 specified.

114136 **Editing Commands in sed**

114137 In the following list of editing commands, the maximum number of permissible addresses for  
114138 each function is indicated by *[0addr]*, *[1addr]*, or *[2addr]*, representing zero, one, or two  
114139 addresses.

114140 The argument *text* shall consist of one or more lines. A <backslash> in the text can be escaped  
114141 with another <backslash>. The application shall ensure that each embedded <newline> (that is,  
114142 those other than the terminating <newline> of the last line) in the text is preceded by an  
114143 unescaped <backslash>. The behavior is unspecified if an unescaped <backslash> is  
114144 immediately followed by any character other than <backslash> or <newline>, or by the end of a  
114145 *script*.

114146 The **r** and **w** command verbs, and the *w* flag to the **s** command, take an *rfile* (or *wfile*) parameter,  
114147 separated from the command verb letter or flag by one or more <blank> characters;  
114148 implementations may allow zero separation as an extension.

114149 The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be  
114150 created before processing begins. Implementations shall support at least ten *wfile* arguments in  
114151 the script; the actual number (greater than or equal to 10) that is supported by the  
114152 implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially  
114153 created, if it does not exist, or shall replace the contents of an existing file.

114154 The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following  
114155 synopses indicate which arguments shall be separated from the command verbs by a single  
114156 <space>.

114157 The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and  
114158 the contents of the file specified for the **r** command, shall be written to standard output just  
114159 before the next attempt to fetch a line of input when executing the **c**, **D**, **d**, **N**, or **n** commands,  
114160 just before executing the **q** command, or when reaching the end of the script. If written when  
114161 reaching the end of the script, and the **-n** option was not specified, the text shall be written after  
114162 copying the pattern space to standard output. The contents of the file specified for the **r**  
114163 command shall be as of the time the output is written, not the time the **r** command is applied.  
114164 The text shall be output in the order in which the **a** and **r** commands were applied to the input.

114165 Editing commands other than **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a <semicolon>,  
114166 optional <blank> characters, and another editing command. However, when an **s** editing  
114167 command is used with the *w* flag, following it with another command in this manner produces  
114168 undefined results.

114169 A function can be preceded by a **!** character, in which case the function shall be applied if the  
114170 addresses do not select the pattern space. Zero or more <blank> characters shall be accepted  
114171 before the **!** character. It is unspecified whether <blank> characters can follow the **!**  
114172 character, and conforming applications shall not follow the **!** character with <blank>  
114173 characters.

114174 If a *label* argument (to a **b**, **t**, or **:** command) contains characters outside of the portable filename  
114175 character set, or if a *label* is longer than 8 bytes, the behavior is unspecified. The implementation  
114176 shall support *label* arguments recognized as unique up to at least 8 bytes; the actual length  
114177 (greater than or equal to 8) supported by the implementation is unspecified. It is unspecified  
114178 whether exceeding the maximum supported label length causes an error or a silent truncation.

114179 *[2addr] {editing command}*  
114180 *editing command*

|        |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 114181 | ...               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114182 | }                 | Execute a list of <i>sed</i> editing commands only when the pattern space is selected. The list of <i>sed</i> editing commands shall be surrounded by braces. The braces can be preceded or followed by <blank> characters. The <right-brace> shall be preceded by a <newline> or <semicolon> (before any optional <blank> characters preceding the <right-brace>).                                                                                                                    |
| 114183 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114184 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114185 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114186 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114187 |                   | Each command in the list of commands shall be terminated by a <newline> character, or by a <semicolon> character if permitted when the command is used outside the braces. The editing commands can be preceded by <blank> characters, but shall not be followed by <blank> characters.                                                                                                                                                                                                |
| 114188 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114189 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114190 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114191 | [1addr]a\<br>text | Write text to standard output as described previously.                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 114192 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114193 | [2addr]b [label]  | Branch to the : command verb bearing the <i>label</i> argument. If <i>label</i> is not specified, branch to the end of the script.                                                                                                                                                                                                                                                                                                                                                     |
| 114194 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114195 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114196 | [2addr]c\<br>text | Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range, place <i>text</i> on the output. Start the next cycle.                                                                                                                                                                                                                                                                                                                                             |
| 114197 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114198 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114199 | [2addr]d          | Delete the pattern space and start the next cycle.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114200 | [2addr]D          | If the pattern space contains no <newline>, delete the pattern space and start a normal new cycle as if the <b>d</b> command was issued. Otherwise, delete the initial segment of the pattern space through the first <newline>, and start the next cycle with the resultant pattern space and without reading any new input.                                                                                                                                                          |
| 114201 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114202 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114203 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114204 | [2addr]g          | Replace the contents of the pattern space by the contents of the hold space.                                                                                                                                                                                                                                                                                                                                                                                                           |
| 114205 | [2addr]G          | Append to the pattern space a <newline> followed by the contents of the hold space.                                                                                                                                                                                                                                                                                                                                                                                                    |
| 114206 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114207 | [2addr]h          | Replace the contents of the hold space with the contents of the pattern space.                                                                                                                                                                                                                                                                                                                                                                                                         |
| 114208 | [2addr]H          | Append to the hold space a <newline> followed by the contents of the pattern space.                                                                                                                                                                                                                                                                                                                                                                                                    |
| 114209 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114210 | [1addr]i\<br>text | Write <i>text</i> to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 114211 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114212 | [2addr]l          | (The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in XBD Table 5-1 (on page 113) ('\\', '\a', '\b', '\f', '\r', '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-printable characters not in that table shall be written as one three-digit octal number (with a preceding <backslash>) for each byte in the character (most significant byte first). |
| 114213 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114214 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114215 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114216 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114217 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114218 |                   | Long lines shall be folded, with the point of folding indicated by writing a <backslash> followed by a <newline>; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a '\$'.                                                                                                                                                                                                                |
| 114219 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114220 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114221 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114222 | [2addr]n          | Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input, less its terminating <newline>.                                                                                                                                                                                                                                                                                                   |
| 114223 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 114224 |                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

- 114225 If no next line of input is available, the **n** command verb shall branch to the end of  
114226 the script and quit without starting a new cycle.
- 114227 **[2addr]N** Append the next line of input, less its terminating <newline>, to the pattern space,  
114228 using an embedded <newline> to separate the appended material from the  
114229 original material. Note that the current line number changes.
- 114230 If no next line of input is available, the **N** command verb shall branch to the end of  
114231 the script and quit without starting a new cycle or copying the pattern space to  
114232 standard output.
- 114233 **[2addr]p** Write the pattern space to standard output.
- 114234 **[2addr]P** Write the pattern space, up to the first <newline>, to standard output.
- 114235 **[1addr]q** Branch to the end of the script and quit without starting a new cycle.
- 114236 **[1addr]r rfile** Copy the contents of *rfile* to standard output as described previously. If *rfile* does  
114237 not exist or cannot be read, it shall be treated as if it were an empty file, causing no  
114238 error condition.
- 114239 **[2addr]s/RE/replacement/flags**  
114240 Substitute the replacement string for instances of the RE in the pattern space. Any  
114241 character other than <backslash> or <newline> can be used instead of a <slash> to  
114242 delimit the RE and the replacement. Within the RE (as a *sed* extension to the BRE  
114243 and ERE syntax) and the replacement, the delimiter shall not terminate the RE or  
114244 replacement if it is the second character of an escape sequence (see XBD [Section](#)  
114245 [9.1](#), on page 179). If the delimiter character is not listed as a special BRE character  
114246 (if the **-E** option is not specified) or a special ERE character (if **-E** is specified) in  
114247 XBD [Section 9.3.3](#) (on page 182) or XBD [Section 9.4.3](#) (on page 188), respectively,  
114248 the escaped delimiter shall be treated as that literal character in the RE; otherwise,  
114249 it is unspecified whether the escaped delimiter is treated as the literal character or  
114250 the special character. Likewise, if the delimiter character is not <ampersand>  
114251 ('&'), the escaped delimiter shall be treated as that literal character in the  
114252 replacement; if it is <ampersand>, it is unspecified whether the escaped delimiter  
114253 is treated as the literal character or the special character (see below).
- 114254 The replacement string shall be scanned from beginning to end. An <ampersand>  
114255 ('&') appearing in the replacement shall be replaced by the string matching the  
114256 RE. The special meaning of '&' in this context can be suppressed by preceding it  
114257 by a <backslash>. The characters "*\n*", where *n* is a digit, shall be replaced by the  
114258 text matched by the corresponding back-reference expression. If the corresponding  
114259 back-reference expression does not match, then the characters "*\n*" shall be  
114260 replaced by the empty string. The special meaning of "*\n*" where *n* is a digit in  
114261 this context, can be suppressed by preceding it by a <backslash>. For each other  
114262 <backslash> encountered, the following character shall lose its special meaning (if  
114263 any).
- 114264 A line can be split by substituting a <newline> into it. The application shall escape  
114265 the <newline> in the replacement by preceding it by a <backslash>.
- 114266 The meaning of an unescaped <backslash> immediately followed by any character  
114267 other than '&', <backslash>, a digit, <newline>, or the delimiter character used for  
114268 this command, is unspecified.
- 114269 Any <backslash> used to alter the default meaning of a subsequent character shall  
114270 be discarded from the resulting replacement string. A substitution shall be

|        |                                                               |                                                                                                              |
|--------|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| 114271 |                                                               | considered to have been performed even if the resulting replacement string is                                |
| 114272 |                                                               | identical to the string that it replaces.                                                                    |
| 114273 |                                                               | The value of <i>flags</i> shall be zero or more of:                                                          |
| 114274 | <i>n</i>                                                      | Substitute for the <i>n</i> th occurrence only of the RE found within the                                    |
| 114275 |                                                               | pattern space.                                                                                               |
| 114276 | <b>g</b>                                                      | Globally substitute for all non-overlapping instances of the RE rather                                       |
| 114277 |                                                               | than just the first one. If both <b>g</b> and <i>n</i> are specified, the results are                        |
| 114278 |                                                               | unspecified.                                                                                                 |
| 114279 | <b>i</b>                                                      | Match the regular expression in a case-insensitive way.                                                      |
| 114280 | <b>p</b>                                                      | Write the pattern space to standard output if a replacement was                                              |
| 114281 |                                                               | made.                                                                                                        |
| 114282 | <b>w</b> <i>wfile</i>                                         | Write. Append the pattern space to <i>wfile</i> if a replacement was made.                                   |
| 114283 |                                                               | A conforming application shall precede the <i>wfile</i> argument with one                                    |
| 114284 |                                                               | or more <blank> characters. If the <b>w</b> flag is not the last flag value                                  |
| 114285 |                                                               | given in a concatenation of multiple flag values, the results are                                            |
| 114286 |                                                               | undefined.                                                                                                   |
| 114287 | [ <i>2addr</i> ] <b>t</b> [ <i>label</i> ]                    |                                                                                                              |
| 114288 |                                                               | Test. Branch to the <b>:</b> command verb bearing the <i>label</i> if any substitutions have been            |
| 114289 |                                                               | made since the most recent reading of an input line or execution of a <b>t</b> . If <i>label</i> is          |
| 114290 |                                                               | not specified, branch to the end of the script.                                                              |
| 114291 | [ <i>2addr</i> ] <b>w</b> <i>wfile</i>                        |                                                                                                              |
| 114292 |                                                               | Append (write) the pattern space to <i>wfile</i> .                                                           |
| 114293 | [ <i>2addr</i> ] <b>x</b>                                     | Exchange the contents of the pattern and hold spaces.                                                        |
| 114294 | [ <i>2addr</i> ] <b>y</b> / <i>string1</i> / <i>string2</i> / |                                                                                                              |
| 114295 |                                                               | Replace all occurrences of characters in <i>string1</i> with the corresponding characters                    |
| 114296 |                                                               | in <i>string2</i> . If a <backslash> followed by an 'n' appear in <i>string1</i> or <i>string2</i> , the two |
| 114297 |                                                               | characters shall be handled as a single <newline>. If (after resolving any escape                            |
| 114298 |                                                               | sequences) the numbers of characters in <i>string1</i> and <i>string2</i> are not equal, or if any           |
| 114299 |                                                               | of the characters in <i>string1</i> appear more than once, the results are undefined. Any                    |
| 114300 |                                                               | character other than <backslash> or <newline> can be used instead of <slash> to                              |
| 114301 |                                                               | delimit the strings. If the delimiter is not 'n', within <i>string1</i> and <i>string2</i> , the             |
| 114302 |                                                               | delimiter itself can be used as a literal character if it is preceded by an unescaped                        |
| 114303 |                                                               | <backslash>. If a <backslash> character is escaped by an immediately preceding                               |
| 114304 |                                                               | unescaped <backslash> character in <i>string1</i> or <i>string2</i> , the two <backslash>                    |
| 114305 |                                                               | characters shall be treated as a single literal <backslash> character. The meaning of                        |
| 114306 |                                                               | an unescaped <backslash> followed by any character that is not 'n', a                                        |
| 114307 |                                                               | <backslash>, or the delimiter character is undefined.                                                        |
| 114308 | [ <i>0addr</i> ]: <i>label</i>                                | Do nothing. This command bears a <i>label</i> to which the <b>b</b> and <b>t</b> commands branch.            |
| 114309 | [ <i>1addr</i> ]=                                             | Write the following to standard output:                                                                      |
| 114310 |                                                               | "%d\n", <current line number>                                                                                |
| 114311 | [ <i>0addr</i> ]                                              | Ignore this empty command.                                                                                   |
| 114312 | [ <i>0addr</i> ] <b>#</b>                                     | Ignore the '#' and the remainder of the line (treat them as a comment), with the                             |
| 114313 |                                                               | single exception that if the first two characters in the script are "#n", the default                        |
| 114314 |                                                               | output shall be suppressed; this shall be the equivalent of specifying <b>-n</b> on the                      |



114315 command line.

#### 114316 EXIT STATUS

114317 The following exit values shall be returned:

114318 0 Successful completion.

114319 >0 An error occurred.

#### 114320 CONSEQUENCES OF ERRORS

114321 Default.

#### 114322 APPLICATION USAGE

114323 Regular expressions match entire strings, not just individual lines, but a <newline> is matched  
114324 by '\n' in a *sed* RE; a <newline> is not allowed by the general definition of regular expression  
114325 in POSIX.1-2024. Also note that '\n' cannot be used to match a <newline> at the end of an  
114326 arbitrary input line; <newline> characters appear in the pattern space as a result of the **N** editing  
114327 command.

114328 Applications that use a special RE character as a delimiter (for example, '.' or '\*') and need to  
114329 use the delimiter as a literal character in the RE should put it inside a bracket expression, as  
114330 implementations differ regarding whether escaping it with a <backslash> removes its special  
114331 meaning. For example, for the context address "/\.[0-9]/" to be written with '.' as  
114332 delimiter, the form "\.[.] [0-9]." needs to be used; "\.\.[0-9]." cannot be used portably  
114333 for this purpose, as it is unspecified whether this would be equivalent to "/\.[0-9]/" or  
114334 "/.[0-9]/". Portable applications cannot use a special RE character as a delimiter if that  
114335 character needs to have its special meaning in the RE, as escaping it may remove its special  
114336 meaning.

114337 When using *sed* to process pathnames, it is recommended that LC\_ALL, or at least LC\_CTYPE  
114338 and LC\_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte  
114339 sequences that do not form valid characters in some locales, in which case the utility's behavior  
114340 would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore  
114341 this problem is avoided.

114342 Note that some implementations of *sed* also support an **I** flag for the **s** command as an alias for  
114343 the lower case **i** flag.

114344 Some implementations of *sed*, when executed in a non-conforming environment, handle  
114345 <backslash> escapes in regular expressions in a similar way to how *awk* handles them in the  
114346 lexical token **ERE** (processing "\t" as a tab character, etc.). This is a compatible extension except  
114347 that it conflicts with the requirements of this standard when <backslash> appears inside a  
114348 bracket expression. A future version of this standard may allow this behavior, and therefore  
114349 applications should use two <backslash> characters in bracket expressions instead of one in  
114350 order to ensure future portability. On implementations conforming to the current standard, the  
114351 second <backslash> is redundant. In the future (and in current non-conforming environments)  
114352 the first <backslash> may escape the second.

#### 114353 EXAMPLES

114354 This *sed* script simulates the BSD *cat -s* command, squeezing excess empty lines from standard  
114355 input.

```
114356 sed -n '
114357 # Write non-empty lines.
114358 ./ {
114359     p
114360     d
```

```

114361     }
114362     # Write a single empty line, then look for more empty lines.
114363     /^$/ p
114364     # Get next line, discard the held <newline> (empty line),
114365     # and look for more empty lines.
114366     :Empty
114367     /^$/ {
114368         N
114369         s/././
114370         b Empty
114371     }
114372     # Write the non-empty line before going back to search
114373     # for the first in a set of empty lines.
114374     p
114375     '

```

114376 The following *sed* command is a much simpler method of squeezing empty lines, although it is  
 114377 not quite the same as *cat -s* since it removes any initial empty lines:

```
114378 sed -n '/./,/^$/p'
```

#### 114379 RATIONALE

114380 This volume of POSIX.1-2024 requires implementations to support at least ten distinct *wfiles*,  
 114381 matching historical practice on many implementations. Implementations are encouraged to  
 114382 support more, but conforming applications should not exceed this limit.

114383 The exit status codes specified here are different from those in System V. System V returns 2 for  
 114384 garbled *sed* commands, but returns zero with its usage message or if the input file could not be  
 114385 opened. The standard developers considered this to be a bug.

114386 The manner in which the *l* command writes non-printable characters was changed to avoid the  
 114387 historical backspace-overstrike method, and other requirements to achieve unambiguous output  
 114388 were added. See the RATIONALE for *ed* for details of the format chosen, which is the same as  
 114389 that chosen for *sed*.

114390 This volume of POSIX.1-2024 requires implementations to provide pattern and hold spaces of at  
 114391 least 8 192 bytes, larger than the 4 000 bytes spaces used by some historical implementations, but  
 114392 less than the 20 480 bytes limit used in an early proposal. Implementations are encouraged to  
 114393 allocate dynamically larger pattern and hold spaces as needed.

114394 The requirements for acceptance of <blank> and <space> characters in command lines has been  
 114395 made more explicit than in early proposals to describe clearly the historical practice and to  
 114396 remove confusion about the phrase “protect initial blanks [*sic*] and tabs from the stripping that is  
 114397 done on every script line” that appears in much of the historical documentation of the *sed* utility  
 114398 description of text. (Not all implementations are known to have stripped <blank> characters  
 114399 from text lines, although they all have allowed leading <blank> characters preceding the address  
 114400 on a command line.)

114401 The treatment of '#' comments differs from the SVID which only allows a comment as the first  
 114402 line of the script, but matches BSD-derived implementations. The comment character is treated  
 114403 as a command, and it has the same properties in terms of being accepted with leading <blank>  
 114404 characters; the BSD implementation has historically supported this.

114405 Early proposals required that a *script\_file* have at least one non-comment line. Some historical  
 114406 implementations have behaved in unexpected ways if this were not the case. The standard  
 114407 developers considered that this was incorrect behavior and that application developers should

114408 not have to avoid this feature. A correct implementation of this volume of POSIX.1-2024 shall  
114409 permit *script\_files* that consist only of comment lines.

114410 Early proposals indicated that if `-e` and `-f` options were intermixed, all `-e` options were  
114411 processed before any `-f` options. This has been changed to process them in the order presented  
114412 because it matches historical practice and is more intuitive.

114413 The characters `<backslash>` and `<newline>` cannot be used as RE delimiter characters, as they  
114414 can never be recognized as the ending delimiter:

- 114415 • `<backslash>` does not work, because if it appears unescaped later in the RE, it either  
114416 escapes the following character, which can then never be the ending delimiter, or it is part  
114417 of a bracket expression, inside which the ending delimiter for the RE cannot be located.
- 114418 • `<newline>` does not work, because if not escaped, it terminates the command, meaning it  
114419 cannot be the ending delimiter.

114420 Some historical *sed* implementations did not support escaping `'(, ')', '{', and '}'` when  
114421 used as a BRE delimiter, as the sequences `"\ ("` and so on were still treated as special, usually  
114422 resulting in an error. This standard requires that these sequences are treated as the literal  
114423 character. This is for consistency with extensions. For example, some implementations treat  
114424 `"\s"` in a BRE as matching white-space characters, as an extension. This cannot have its special  
114425 meaning when `'s'` is used as a BRE delimiter in order to ensure portability of *sed* commands  
114426 that have `'s'` as a delimiter and escape it. If `"\s"` were allowed to keep its special meaning,  
114427 then the potential for further extensions would mean portable applications would not be able to  
114428 escape any delimiter character other than `<slash>`.

114429 The treatment of the `p` flag to the `s` command differs between System V and BSD-based systems  
114430 when the default output is suppressed. In the two examples:

```
114431 echo a | sed 's/a/A/p'
114432 echo a | sed -n 's/a/A/p'
```

114433 this volume of POSIX.1-2024, BSD, System V documentation, and the SVID indicate that the first  
114434 example should write two lines with **A**, whereas the second should write one. Some System V  
114435 systems write the **A** only once in both examples because the `p` flag is ignored if the `-n` option is  
114436 not specified.

114437 This is a case of a diametrical difference between systems that could not be reconciled through  
114438 the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V  
114439 documentation behavior was adopted for this volume of POSIX.1-2024 because:

- 114440 • No known documentation for any historic system describes the interaction between the `p`  
114441 flag and the `-n` option.
- 114442 • The selected behavior is more correct as there is no technical justification for any  
114443 interaction between the `p` flag and the `-n` option. A relationship between `-n` and the `p` flag  
114444 might imply that they are only used together, but this ignores valid scripts that interrupt  
114445 the cyclical nature of the processing through the use of the `D`, `d`, `q`, or branching  
114446 commands. Such scripts rely on the `p` suffix to write the pattern space because they do not  
114447 make use of the default output at the ```bottom``` of the script.
- 114448 • Because the `-n` option makes the `p` flag unnecessary, any interaction would only be useful  
114449 if *sed* scripts were written to run both with and without the `-n` option. This is believed to  
114450 be unlikely. It is even more unlikely that programmers have coded the `p` flag expecting it to  
114451 be unnecessary. Because the interaction was not documented, the likelihood of a  
114452 programmer discovering the interaction and depending on it is further decreased.

- 114453           • Finally, scripts that break under the specified behavior produce too much output instead of  
114454           too little, which is easier to diagnose and correct.

114455           The form of the substitute command that uses the **n** suffix was limited to the first 512 matches in  
114456           an early proposal. This limit has been removed because there is no reason an editor processing  
114457           lines of {LINE\_MAX} length should have this restriction. The command *s/a/A/2047* should be  
114458           able to substitute the 2 047th occurrence of **a** on a line.

114459           The **b**, **t**, and **:** commands are documented to ignore leading white space, but no mention is  
114460           made of trailing white space. Historical implementations of *sed* assigned different locations to  
114461           the labels '**x**' and "**x**". This is not useful, and leads to subtle programming errors, but it is  
114462           historical practice, and changing it could theoretically break working scripts. Implementors are  
114463           encouraged to provide warning messages about labels that are never referenced by a **b** or **t**  
114464           command, jumps to labels that do not exist, and label arguments that are subject to truncation.

114465           Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
114466           but this has been modified in this version.

#### 114467 **FUTURE DIRECTIONS**

114468           A future version of this standard may allow *sed* to handle <backslash> escapes in regular  
114469           expressions in a similar way to how *awk* handles them in the lexical token **ERE**. ("Similar"  
114470           rather than "the same" because *sed* can use BREs or EREs whereas *awk* uses only EREs.)

#### 114471 **SEE ALSO**

114472           *awk*, *ed*, *grep*

114473           XBD Table 5-1 (on page 113), Chapter 8 (on page 167), Section 9.3 (on page 181), Section 12.2 (on  
114474           page 215)

#### 114475 **CHANGE HISTORY**

114476           First released in Issue 2.

#### 114477 **Issue 5**

114478           The FUTURE DIRECTIONS section is added.

#### 114479 **Issue 6**

114480           The following new requirements on POSIX implementations derive from alignment with the  
114481           Single UNIX Specification:

- 114482           • Implementations are required to support at least ten *wfile* arguments in an editing  
114483           command.

114484           The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

114485           IEEE PASC Interpretation 1003.2 #190 is applied.

114486           IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the  
114487           <backslash>-escape sequences in a replacement string for a BRE.

114488           IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/28 is applied, removing text describing  
114489           behavior on systems with bytes consisting of more than eight bits.

114490           IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/29 is applied, making an editorial  
114491           correction within the Editing Commands in *sed* section.

#### 114492 **Issue 7**

114493           Austin Group Interpretations 1003.1-2001 #006, #036, and #092 are applied.

114494           SD5-XCU-ERN-97 and SD5-XCU-ERN-123 are applied, updating the SYNOPSIS.

114495           A second example is added.

- 114496 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0133 [262], XCU/TC1-2008/0134  
114497 [282,431], XCU/TC1-2008/0135 [269], and XCU/TC1-2008/0136 [282,431] are applied.
- 114498 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0166 [945], XCU/TC2-2008/0167  
114499 [944], XCU/TC2-2008/0168 [945], XCU/TC2-2008/0169 [944], XCU/TC2-2008/0170 [945],  
114500 XCU/TC2-2008/0171 [533], XCU/TC2-2008/0172 [663], XCU/TC2-2008/0173 [945], and  
114501 XCU/TC2-2008/0174 [944] are applied.
- 114502 **Issue 8**
- 114503 Austin Group Defect 528 is applied, adding support for selecting the use of EREs instead of  
114504 BREs, by specifying the `-E` option.
- 114505 Austin Group Defect 779 is applied, adding the `i` flag to the `s` command.
- 114506 Austin Group Defect 961 is applied, requiring that `{...}` can be followed by a `<semicolon>`,  
114507 optional `<blank>` characters, and another editing command.
- 114508 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.
- 114509 Austin Group Defect 1231 is applied, clarifying the handling of `<backslash>` in `text` arguments.
- 114510 Austin Group Defect 1233 is applied, changing the APPLICATION USAGE and FUTURE  
114511 DIRECTIONS sections.
- 114512 Austin Group Defect 1319 is applied, changing when the text specified for the `a` command and  
114513 the contents of the file specified for the `r` command are written.
- 114514 Austin Group Defect 1550 is applied, clarifying requirements relating to delimiters in context  
114515 addresses and in `s` and `y` commands.
- 114516 Austin Group Defect 1578 is applied, clarifying the description of the `y` command.
- 114517 Austin Group Defect 1767 is applied, clarifying that a `c` command starts the next cycle on every  
114518 line that its address range matches.

114519 **NAME**

114520 sh — shell, the standard command language interpreter

114521 **SYNOPSIS**

114522 OB sh [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...  
 114523 [command\_file [argument...]]

114524 OB sh -c [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...  
 114525 command\_string [command\_name [argument...]]

114526 OB sh -s [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...  
 114527 [argument...]

114528 **DESCRIPTION**

114529 The *sh* utility is a command language interpreter that shall execute commands read from a  
 114530 command line string, the standard input, or a specified file. The application shall ensure that the  
 114531 commands to be executed are expressed in the language described in [Chapter 2](#) (on page 2472).

114532 Pathname expansion shall not fail due to the size of a file.

114533 Shell input and output redirections have an implementation-defined offset maximum that is  
 114534 established in the open file description.

114535 **OPTIONS**

114536 The *sh* utility shall conform to XBD [Section 12.2](#) (on page 215), with an extension for support of a  
 114537 leading <plus-sign> ('+') as noted below.

114538 The *-a*, *-b*, *-C*, *-e*, *-f*, *-h*, *-m*, *-n*, *-o option*, *-u*, *-v*, and *-x* options are described as part of the  
 114539 *set* utility in [Section 2.15](#) (on page 2526). The option letters derived from the *set* special built-in  
 114540 shall also be accepted with a leading <plus-sign> ('+') instead of a leading <hyphen-minus>  
 114541 (meaning the reverse case of the option as described in this volume of POSIX.1-2024). If the *-o*  
 114542 or *+o* option is specified without an option-argument, the behavior is unspecified.

114543 The following additional options shall be supported:

114544 *-c* Read commands from the *command\_string* operand. Set the value of special  
 114545 parameter 0 (see [Section 2.5.2](#), on page 2479) from the value of the *command\_name*  
 114546 operand and the positional parameters (\$1, \$2, and so on) in sequence from the  
 114547 remaining *argument* operands. No commands shall be read from the standard  
 114548 input.

114549 *-i* Specify that the shell is *interactive*; see below. An implementation may treat  
 114550 specifying the *-i* option as an error if the real user ID of the calling process does  
 114551 not equal the effective user ID or if the real group ID does not equal the effective  
 114552 group ID.

114553 *-s* Read commands from the standard input.

114554 If there are no operands and the *-c* option is not specified, the *-s* option shall be assumed.

114555 If the *-i* option is present, or if the shell reads commands from the standard input and the shell's  
 114556 standard input and standard error are attached to a terminal, the shell is considered to be  
 114557 *interactive*.

114558 **OPERANDS**

114559 The following operands shall be supported:

114560 *-* A single <hyphen-minus> shall be treated as the first operand and then ignored. If  
 114561 both '-' and "--" are given as arguments, or if other operands precede the single  
 114562 <hyphen-minus>, the results are undefined.

- 114563        *argument*     The positional parameters (\$1, \$2, and so on) shall be set to *arguments*, if any.
- 114564        *command\_file* The pathname of a file containing commands. If the pathname contains one or  
114565                    more <slash> characters, the implementation attempts to read that file; the file  
114566                    need not be executable. If the pathname does not contain a <slash> character:
- 114567                    • The implementation shall attempt to read that file from the current working  
114568                    directory; the file need not be executable.
  - 114569                    • If the file is not in the current working directory, the implementation may  
114570                    perform a search for an executable file using the value of *PATH*, as described  
114571                    in [Section 2.9.1.4](#) (on page 2502).
- 114572                    Special parameter 0 (see [Section 2.5.2](#), on page 2479) shall be set to the value of  
114573                    *command\_file*. If *sh* is called using a synopsis form that omits *command\_file*, special  
114574                    parameter 0 shall be set to the value of the first argument passed to *sh* from its  
114575                    parent (for example, *argv*[0] for a C program), which is normally a pathname used  
114576                    to execute the *sh* utility.
- 114577        *command\_name*     A string assigned to special parameter 0 when executing the commands in  
114578                    *command\_string*. If *command\_name* is not specified, special parameter 0 shall be set  
114579                    to the value of the first argument passed to *sh* from its parent (for example, *argv*[0]  
114580                    for a C program), which is normally a pathname used to execute the *sh* utility.  
114581
- 114582        *command\_string*     A string that shall be interpreted by the shell as one or more commands, as if the  
114583                    string were the argument to the *system()* function defined in the System Interfaces  
114584                    volume of POSIX.1-2024. If the *command\_string* operand is an empty string, *sh* shall  
114585                    exit with a zero exit status.  
114586
- 114587        **STDIN**
- 114588                    The standard input shall be used only if one of the following is true:
- 114589                    • The *-s* option is specified.
  - 114590                    • The *-c* option is not specified and no operands are specified.
  - 114591                    • The script executes one or more commands that require input from standard input (such as  
114592                    a *read* command that does not redirect its input).
- 114593                    See the INPUT FILES section.
- 114594                    When the shell is using standard input and it invokes a command that also uses standard input,  
114595                    the shell shall ensure that the standard input file pointer points directly after the command it has  
114596                    read when the command begins execution. It shall not read ahead in such a manner that any  
114597                    characters intended to be read by the invoked command are consumed by the shell (whether  
114598                    interpreted by the shell or not) or that characters that are not read by the invoked command are  
114599                    not seen by the shell. When the command expecting to read standard input is started  
114600                    asynchronously by an interactive shell, it is unspecified whether characters are read by the  
114601                    command or interpreted by the shell.
- 114602                    If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh*  
114603                    shall enable blocking reads on standard input. This shall remain in effect when the command  
114604                    completes.

114605 **INPUT FILES**

114606 The input file can be of any type, but the initial portion of the file intended to be parsed  
 114607 according to the shell grammar (see [Section 2.10.2](#), on page 2513) shall consist of characters and  
 114608 shall not contain the NUL character. The shell shall not enforce any line length limits. If the  
 114609 input file consists solely of zero or more blank lines and comments, *sh* shall exit with a zero exit  
 114610 status.

114611 **ENVIRONMENT VARIABLES**

114612 The following environment variables shall affect the execution of *sh*:

114613 UP **ENV** This variable, when and only when an interactive shell is invoked, shall be  
 114614 subjected to parameter expansion (see [Section 2.6.2](#), on page 2485) by the shell, and  
 114615 the resulting value shall be used as a pathname of a file containing shell  
 114616 commands to execute in the current environment. The file need not be executable.  
 114617 If the expanded value of *ENV* is not an absolute pathname, the results are  
 114618 unspecified. *ENV* shall be ignored if the real and effective user IDs or real and  
 114619 effective group IDs of the process are different. The file specified by *ENV* need not  
 114620 be processed if the file can be written by any user other than the user identified by  
 114621 the real (and effective) user ID of the shell process.

114622 UP **FCEDIT** This variable, when expanded by the shell, shall determine the default value for  
 114623 the `-e editor` option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be  
 114624 used as the editor.

114625 UP **HISTFILE** Determine a pathname naming a command history file. If the *HISTFILE* variable is  
 114626 not set, the shell may attempt to access or create a file `.sh_history` in the directory  
 114627 referred to by the *HOME* environment variable. If the shell cannot obtain both read  
 114628 and write access to, or create, the history file, it shall use an unspecified  
 114629 mechanism that allows the history to operate properly. (References to history  
 114630 "file" in this section shall be understood to mean this unspecified mechanism in  
 114631 such cases.) An implementation may choose to access this variable only when  
 114632 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt  
 114633 to retrieve entries from, or add entries to, the file, as the result of commands issued  
 114634 by the user, the file named by the *ENV* variable, or implementation-defined system  
 114635 start-up files. Implementations may choose to disable the history list mechanism  
 114636 for users with appropriate privileges who do not set *HISTFILE*; the specific  
 114637 circumstances under which this occurs are implementation-defined. If more than  
 114638 one instance of the shell is using the same history file, it is unspecified how  
 114639 updates to the history file from those shells interact. As entries are deleted from the  
 114640 history file, they shall be deleted oldest first. It is unspecified when history file  
 114641 entries are physically removed from the history file.

114642 UP **HISTSIZ** Determine a decimal number representing the limit to the number of previous  
 114643 commands that are accessible. If this variable is unset, an unspecified default  
 114644 greater than or equal to 128 shall be used. The maximum number of commands in  
 114645 the history list is unspecified, but shall be at least 128. An implementation may  
 114646 choose to access this variable only when initializing the history file, as described  
 114647 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZ*  
 114648 after the history file has been initialized are effective.

114649 **HOME** Determine the pathname of the user's home directory. The contents of *HOME* are  
 114650 used in tilde expansion as described in [Section 2.6.1](#) (on page 2485).



|        |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------|-----|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 114651 |     | <i>LANG</i>        | Provide a default value for the internationalization variables that are unset or null. (See XBD <a href="#">Section 8.2</a> (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 114652 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114653 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114654 |     | <i>LC_ALL</i>      | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114655 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114656 |     | <i>LC_COLLATE</i>  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114657 |     |                    | Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 114658 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114659 |     | <i>LC_CTYPE</i>    | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), which characters are defined as letters (character class <b>alpha</b> ), and the behavior of character classes within pattern matching.                                                                                                                                                                                                                                                                                                                                            |
| 114660 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114661 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114662 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114663 |     | <i>LC_MESSAGES</i> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114664 |     |                    | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 114665 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114666 | UP  | <i>MAIL</i>        | Determine a pathname of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to standard error prior to the writing of the next primary prompt string. Such check shall be performed only after the completion of the interval defined by the <i>MAILCHECK</i> variable after the last such check. The user shall be informed only if <i>MAIL</i> is set and <i>MAILPATH</i> is not set.                                    |
| 114667 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114668 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114669 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114670 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114671 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114672 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114673 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114674 | UP  | <i>MAILCHECK</i>   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114675 |     |                    | Establish a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the <i>MAILPATH</i> or <i>MAIL</i> variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt.                                                                                                                                                                                                                                                                                                                                                              |
| 114676 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114677 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114678 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114679 | UP  | <i>MAILPATH</i>    | Provide a list of pathnames and optional messages separated by <colon> characters. If this variable is set, the shell shall inform the user if any of the files named by the variable are created or if any of their modification times change. (See the preceding entry for <i>MAIL</i> for descriptions of mail arrival and user informing.) Each pathname can be followed by '%' and a string that shall be subjected to parameter expansion and written to standard error when the modification time changes. If a '%' character in the pathname is preceded by a <backslash>, it shall be treated as a literal '%' in the pathname. The default message is unspecified. |
| 114680 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114681 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114682 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114683 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114684 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114685 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114686 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114687 |     |                    | The <i>MAILPATH</i> environment variable takes precedence over the <i>MAIL</i> variable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114688 | XSI | <i>NLSPATH</i>     | Determine the location of messages objects and message catalogs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 114689 |     | <i>PATH</i>        | Establish a string formatted as described in XBD <a href="#">Chapter 8</a> (on page 167), used to effect command interpretation; see <a href="#">Section 2.9.1.4</a> (on page 2502).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 114690 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 114691 |     | <i>PWD</i>         | This variable shall represent an absolute pathname of the current working directory. Assignments to this variable may be ignored.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 114692 |     |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

114693 **ASYNCHRONOUS EVENTS**

114694 The *sh* utility shall take the standard action for all signals (see [Section 1.4](#), on page 2462) with the  
 114695 following exceptions.

114696 If the shell is interactive, SIGINT signals received during command line editing shall be handled  
 114697 as described in the EXTENDED DESCRIPTION, and SIGINT signals received at other times  
 114698 shall be caught but no action performed.

114699 If the shell is interactive:

- 114700 • SIGQUIT and SIGTERM signals shall be ignored.
- 114701 • If the `-m` option is in effect, SIGTTIN, SIGTTOU, and SIGTSTP signals shall be ignored.
- 114702 • If the `-m` option is not in effect, it is unspecified whether SIGTTIN, SIGTTOU, and  
 114703 SIGTSTP signals are ignored, set to the default action, or caught. If they are caught, the  
 114704 shell shall, in the signal-catching function, set the signal to the default action and raise the  
 114705 signal (after taking any appropriate steps, such as restoring terminal settings).

114706 The standard actions, and the actions described above for interactive shells, can be overridden  
 114707 by use of the *trap* special built-in utility (see *trap* (on page 2565) and [Section 2.12](#), on page 2521).

114708 **STDOUT**

114709 See the STDERR section.

114710 **STDERR**

114711 Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode),  
 114712 standard error shall be used only for diagnostic messages.

114713 **OUTPUT FILES**

114714 None.

114715 **EXTENDED DESCRIPTION**

114716 UP See [Chapter 2](#). The functionality described in the rest of the EXTENDED DESCRIPTION section  
 114717 shall be provided on implementations that support the User Portability Utilities option (and the  
 114718 rest of this section is not further shaded for this option).

114719 **Command History List**

114720 When the *sh* utility is being used interactively, it shall maintain a list of commands previously  
 114721 entered from the terminal in the file named by the *HISTFILE* environment variable. The type,  
 114722 size, and internal format of this file are unspecified. Multiple *sh* processes can share access to the  
 114723 file for a user, if file access permissions allow this; see the description of the *HISTFILE*  
 114724 environment variable.

114725 **Command Line Editing**

114726 When *sh* is being used interactively from a terminal, the current command and the command  
 114727 history (see *fc*) can be edited using *vi*-mode command line editing. This mode uses commands,  
 114728 described below, similar to a subset of those described in the *vi* utility. Implementations may  
 114729 offer other command line editing modes corresponding to other editing utilities.

114730 The command `set -o vi` shall enable *vi*-mode editing and place *sh* into *vi* insert mode (see  
 114731 [Command Line Editing \(vi-mode\)](#), on page 3371). This command also shall disable any other  
 114732 editing mode that the implementation may provide. The command `set +o vi` disables *vi*-mode  
 114733 editing.

114734 Certain block-mode terminals may be unable to support shell command line editing. If a  
 114735 terminal is unable to provide either edit mode, it need not be possible to `set -o vi` when using the

114736 shell on this terminal.

114737 In the following sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the  
114738 *stty* utility.

### 114739 **Command Line Editing (vi-mode)**

114740 In *vi* editing mode, there shall be a distinguished line, the edit line. All the editing operations  
114741 which modify a line affect the edit line. The edit line is always the newest line in the command  
114742 history buffer.

114743 With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

114744 When in insert mode, an entered character shall be inserted into the command line, except as  
114745 noted in [vi Line Editing Insert Mode](#). Upon entering *sh* and after termination of the previous  
114746 command, *sh* shall be in insert mode.

114747 Typing an escape character shall switch *sh* into command mode (see [vi Line Editing Command](#)  
114748 [Mode](#), on page 3372). In command mode, an entered character shall either invoke a defined  
114749 operation, be used as part of a multi-character operation, or be treated as an error. A character  
114750 that is not recognized as part of an editing command shall terminate any specific editing  
114751 command and shall alert the terminal. If *sh* receives a SIGINT signal in command mode  
114752 (whether generated by typing the *interrupt* character or by other means), it shall terminate  
114753 command line editing on the current command line, reissue the prompt on the next line of the  
114754 terminal, and reset the command history (see *fc*) so that the most recently executed command is  
114755 the previous command (that is, the command that was being edited when it was interrupted is  
114756 not re-entered into the history).

114757 In the following sections, the phrase “move the cursor to the beginning of the word” shall mean  
114758 “move the cursor to the first character of the current word” and the phrase “move the cursor to  
114759 the end of the word” shall mean “move the cursor to the last character of the current word”. The  
114760 phrase “beginning of the command line” indicates the point between the end of the prompt  
114761 string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and  
114762 the first character of the command text.

### 114763 **vi Line Editing Insert Mode**

114764 While in insert mode, any character typed shall be inserted in the current command line, unless  
114765 it is from the following set.

114766 <newline> Execute the current command line. If the current command line is not empty, this  
114767 line shall be entered into the command history (see *fc*).

114768 *erase* Delete the character previous to the current cursor position and move the current  
114769 cursor position back one character. In insert mode, characters shall be erased from  
114770 both the screen and the buffer when backspacing.

114771 *interrupt* If *sh* receives a SIGINT signal in insert mode (whether generated by typing the  
114772 *interrupt* character or by other means), it shall terminate command line editing  
114773 with the same effects as described for interrupting command mode; see [Command](#)  
114774 [Line Editing \(vi-mode\)](#).

114775 *kill* Clear all the characters from the input line.

114776 <control>-V Insert the next character input, even if the character is otherwise a special insert  
114777 mode character.

- 114778 <control>-W Delete the characters from the one preceding the cursor to the preceding word  
114779 boundary. The word boundary in this case is the closer to the cursor of either the  
114780 beginning of the line or a character that is in neither the **blank** nor **punct** character  
114781 classification of the current locale.
- 114782 *end-of-file* Interpreted as the end of input in *sh*. This interpretation shall occur only at the  
114783 beginning of an input line. If *end-of-file* is entered other than at the beginning of the  
114784 line, the results are unspecified.
- 114785 <ESC> Place *sh* into command mode.

## 114786 **vi Line Editing Command Mode**

114787 In command mode for the command line editing feature, decimal digits not beginning with 0  
114788 that precede a command letter shall be remembered. Some commands use these decimal digits  
114789 as a count number that affects the operation.

114790 The term *motion command* represents one of the commands:

114791 <space> 0 b F l W ^ \$ ; E f T w | , B e h t

114792 If the current line is not the edit line, any command that modifies the current line shall cause the  
114793 content of the current line to replace the content of the edit line, and the current line shall  
114794 become the edit line. This replacement cannot be undone (see the **u** and **U** commands below).  
114795 The modification requested shall then be performed to the edit line. When the current line is the  
114796 edit line, the modification shall be done directly to the edit line.

114797 Any command that is preceded by *count* shall take a count (the numeric value of any preceding  
114798 decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat  
114799 by the number of times specified by the count. Also unless otherwise noted, a *count* that is out  
114800 of range is considered an error condition and shall alert the terminal, but neither the cursor  
114801 position, nor the command line, shall change.

114802 The terms *word* and *bigword* are used as defined in the *vi* description. The term *save buffer*  
114803 corresponds to the term *unnamed buffer* in *vi*.

114804 The following commands shall be recognized in command mode:

114805 <newline> Execute the current command line. If the current command line is not empty, this  
114806 line shall be entered into the command history (see *fc*).

114807 <control>-L Redraw the current command line. Position the cursor at the same location on the  
114808 redrawn line.

114809 # Insert the character '#' at the beginning of the current command line and treat the  
114810 resulting edit line as a comment. This line shall be entered into the command  
114811 history; see *fc*.

114812 = Display the possible shell word expansions (see [Section 2.6](#), on page 2483) of the  
114813 bigword at the current command line position.

114814 **Note:** This does not modify the content of the current line, and therefore does not cause  
114815 the current line to become the edit line.

114816 These expansions shall be displayed on subsequent terminal lines. If the bigword  
114817 contains none of the characters '?', '\*', or '[', an <asterisk> ('\*') shall be  
114818 implicitly assumed at the end. If any directories are matched, these expansions  
114819 shall have a '/' character appended. After the expansion, the line shall be  
114820 redrawn, the cursor repositioned at the current cursor position, and *sh* shall be  
114821 placed in command mode.

|        |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 114822 | \                                | Perform pathname expansion (see <a href="#">Section 2.6.6</a> , on page 2493) on the current bigword, up to the largest set of characters that can be matched uniquely. If the bigword contains none of the characters '?', '*', or '[', an <asterisk> ('*') shall be implicitly assumed at the end. This maximal expansion then shall replace the original bigword in the command line, and the cursor shall be placed after this expansion. If the resulting bigword completely and uniquely matches a directory, a '/' character shall be inserted directly after the bigword. If some other file is completely matched, a single <space> shall be inserted after the bigword. After this operation, <i>sh</i> shall be placed in insert mode. |
| 114823 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114824 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114825 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114826 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114827 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114828 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114829 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114830 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114831 | *                                | Perform pathname expansion on the current bigword and insert all expansions into the command to replace the current bigword, with each expansion separated by a single <space>. If at the end of the line, the current cursor position shall be moved to the first column position following the expansions and <i>sh</i> shall be placed in insert mode. Otherwise, the current cursor position shall be the last column position of the first character after the expansions and <i>sh</i> shall be placed in insert mode. If the current bigword contains none of the characters '?', '*', or '[', before the operation, an <asterisk> ('*') shall be implicitly assumed at the end.                                                           |
| 114832 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114833 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114834 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114835 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114836 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114837 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114838 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114839 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114840 | @letter                          | Insert the value of the alias named <i>_letter</i> . The symbol <i>letter</i> represents a single alphabetic character from the portable character set; implementations may support additional characters as an extension. If the alias <i>_letter</i> contains other editing commands, these commands shall be performed as part of the insertion. If no alias <i>_letter</i> is enabled, this command shall have no effect.                                                                                                                                                                                                                                                                                                                     |
| 114841 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114842 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114843 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114844 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114845 | [count]~                         | Convert, if the current character is a lowercase letter, to the equivalent uppercase letter and <i>vice versa</i> , as prescribed by the current locale. The current cursor position then shall be advanced by one character. If the cursor was positioned on the last character of the line, the case conversion shall occur, but the cursor shall not advance. If the '~' command is preceded by a <i>count</i> , that number of characters shall be converted, and the cursor shall be advanced to the character position after the last character converted. If the <i>count</i> is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line. |
| 114846 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114847 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114848 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114849 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114850 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114851 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114852 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114853 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114854 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114855 | [count].                         | Repeat the most recent non-motion command, even if it was executed on an earlier command line. If the previous command was preceded by a <i>count</i> , and no count is given on the '.' command, the count from the previous command shall be included as part of the repeated command. If the '.' command is preceded by a <i>count</i> , this shall override any <i>count</i> argument to the previous command. The <i>count</i> specified in the '.' command shall become the count for subsequent '.' commands issued without a count.                                                                                                                                                                                                       |
| 114856 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114857 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114858 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114859 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114860 | [number]v                        | Invoke the <i>vi</i> editor to edit the current command line in a temporary file. When the editor exits, the commands in the temporary file shall be executed and placed in the command history. If a <i>number</i> is included, it specifies the command number in the command history to be edited, rather than the current command line.                                                                                                                                                                                                                                                                                                                                                                                                       |
| 114861 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114862 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114863 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114864 | [count]l (ell)<br>[count]<space> | Move the current cursor position to the next character position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.                                                                                                                                                                                                                                                                                                                                                             |
| 114865 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114866 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114867 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114868 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114869 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114870 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 114870 |                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

|        |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 114871 | <b>[count]h</b>  | Move the current cursor position to the <i>count</i> th (default 1) previous character position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the count is larger than the number of characters before the cursor, this shall not be considered an error; the cursor shall move to the first character on the line.                                         |
| 114872 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114873 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114874 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114875 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114876 | <b>[count]w</b>  | Move to the start of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.                                                                                                 |
| 114877 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114878 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114879 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114880 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114881 | <b>[count]W</b>  | Move to the start of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.                                                                                           |
| 114882 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114883 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114884 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114885 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114886 | <b>[count]e</b>  | Move to the end of the current word. If at the end of a word, move to the end of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.                                     |
| 114887 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114888 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114889 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114890 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114891 | <b>[count]E</b>  | Move to the end of the current bigword. If at the end of a bigword, move to the end of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.                         |
| 114892 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114893 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114894 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114895 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114896 | <b>[count]b</b>  | Move to the beginning of the current word. If at the beginning of a word, move to the beginning of the previous word. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the <i>count</i> is larger than the number of words preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.             |
| 114897 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114898 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114899 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114900 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114901 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114902 | <b>[count]B</b>  | Move to the beginning of the current bigword. If at the beginning of a bigword, move to the beginning of the previous bigword. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the <i>count</i> is larger than the number of bigwords preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line. |
| 114903 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114904 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114905 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114906 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114907 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114908 | <b>^</b>         | Move the current cursor position to the first character on the input line that is not a <blank>.                                                                                                                                                                                                                                                                                                                                                    |
| 114909 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114910 | <b>\$</b>        | Move to the last character position on the current command line.                                                                                                                                                                                                                                                                                                                                                                                    |
| 114911 | <b>0</b>         | (Zero.) Move to the first character position on the current command line.                                                                                                                                                                                                                                                                                                                                                                           |
| 114912 | <b>[count]  </b> | Move to the <i>count</i> th character position on the current command line. If no number is specified, move to the first position. The first character position shall be numbered 1. If the count is larger than the number of characters on the line, this shall not be considered an error; the cursor shall be placed on the last character on the line.                                                                                         |
| 114913 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114914 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114915 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 114916 |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

|        |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 114917 | <b>[count]fc</b>      | Move to the first occurrence of the character 'c' that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character 'c' does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.                      |
| 114918 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114919 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114920 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114921 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114922 | <b>[count]Fc</b>      | Move to the first occurrence of the character 'c' that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character 'c' does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.                      |
| 114923 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114924 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114925 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114926 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114927 | <b>[count]tc</b>      | Move to the character before the first occurrence of the character 'c' that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character 'c' does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved. |
| 114928 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114929 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114930 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114931 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114932 | <b>[count]Tc</b>      | Move to the character after the first occurrence of the character 'c' that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character 'c' does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.  |
| 114933 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114934 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114935 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114936 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114937 | <b>[count];</b>       | Repeat the most recent <b>f</b> , <b>F</b> , <b>t</b> , or <b>T</b> command. Any number argument on that previous command shall be ignored. Errors are those described for the repeated command.                                                                                                                                                                                                              |
| 114938 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114939 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114940 | <b>[count],</b>       | Repeat the most recent <b>f</b> , <b>F</b> , <b>t</b> , or <b>T</b> command. Any number argument on that previous command shall be ignored. However, reverse the direction of that command.                                                                                                                                                                                                                   |
| 114941 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114942 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114943 | <b>a</b>              | Enter insert mode after the current cursor position. Characters that are entered shall be inserted before the next character.                                                                                                                                                                                                                                                                                 |
| 114944 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114945 | <b>A</b>              | Enter insert mode after the end of the current command line.                                                                                                                                                                                                                                                                                                                                                  |
| 114946 | <b>i</b>              | Enter insert mode at the current cursor position. Characters that are entered shall be inserted before the current character.                                                                                                                                                                                                                                                                                 |
| 114947 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114948 | <b>I</b>              | Enter insert mode at the beginning of the current command line.                                                                                                                                                                                                                                                                                                                                               |
| 114949 | <b>R</b>              | Enter insert mode, replacing characters from the command line beginning at the current cursor position.                                                                                                                                                                                                                                                                                                       |
| 114950 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114951 | <b>[count]cmotion</b> |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114952 |                       | Delete the characters between the current cursor position and the cursor position that would result from the specified motion command. Then enter insert mode before the first character following any deleted characters. If <i>count</i> is specified, it shall be applied to the motion command. A <i>count</i> shall be ignored for the following motion commands:                                        |
| 114953 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114954 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114955 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114956 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114957 |                       | 0    ^    \$    c                                                                                                                                                                                                                                                                                                                                                                                             |
| 114958 |                       | If the motion command is the character 'c', the current command line shall be cleared and insert mode shall be entered. If the motion command would move the current cursor position toward the beginning of the command line, the character under the current cursor position shall not be deleted. If the motion command                                                                                    |
| 114959 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114960 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 114961 |                       |                                                                                                                                                                                                                                                                                                                                                                                                               |

|        |                       |                                                                                                   |
|--------|-----------------------|---------------------------------------------------------------------------------------------------|
| 114962 |                       | would move the current cursor position toward the end of the command line, the                    |
| 114963 |                       | character under the current cursor position shall be deleted. If the <i>count</i> is larger       |
| 114964 |                       | than the number of characters between the current cursor position and the end of                  |
| 114965 |                       | the command line toward which the motion command would move the cursor, this                      |
| 114966 |                       | shall not be considered an error; all of the remaining characters in the                          |
| 114967 |                       | mentioned range shall be deleted and insert mode shall be entered. If the                         |
| 114968 |                       | motion command is invalid, the terminal shall be alerted, the cursor shall not be                 |
| 114969 |                       | moved, and no text shall be deleted.                                                              |
| 114970 | <b>C</b>              | Delete from the current character to the end of the line and enter insert mode at the             |
| 114971 |                       | new end-of-line.                                                                                  |
| 114972 | <b>S</b>              | Clear the entire edit line and enter insert mode.                                                 |
| 114973 | <b>[count]rc</b>      | Replace the current character with the character 'c'. With a number <i>count</i> ,                |
| 114974 |                       | replace the current and the following <i>count</i> -1 characters. After this command, the         |
| 114975 |                       | current cursor position shall be on the last character that was changed. If the <i>count</i>      |
| 114976 |                       | is larger than the number of characters after the cursor, this shall not be considered            |
| 114977 |                       | an error; all of the remaining characters shall be changed.                                       |
| 114978 | <b>[count]_</b>       | Append a <space> after the current character position and then append the last                    |
| 114979 |                       | bigword in the previous input line after the <space>. Then enter insert mode after                |
| 114980 |                       | the last character just appended. With a number <i>count</i> , append the <i>count</i> th bigword |
| 114981 |                       | in the previous line.                                                                             |
| 114982 | <b>[count]x</b>       | Delete the character at the current cursor position and place the deleted characters              |
| 114983 |                       | in the save buffer. If the cursor was positioned on the last character of the line, the           |
| 114984 |                       | character shall be deleted and the cursor position shall be moved to the previous                 |
| 114985 |                       | character (the new last character). If the <i>count</i> is larger than the number of              |
| 114986 |                       | characters after the cursor, this shall not be considered an error; all the characters            |
| 114987 |                       | from the cursor to the end of the line shall be deleted.                                          |
| 114988 | <b>[count]X</b>       | Delete the character before the current cursor position and place the deleted                     |
| 114989 |                       | characters in the save buffer. The character under the current cursor position shall              |
| 114990 |                       | not change. If the cursor was positioned on the first character of the line, the                  |
| 114991 |                       | terminal shall be alerted, and the X command shall have no effect. If the line                    |
| 114992 |                       | contained a single character, the X command shall have no effect. If the line                     |
| 114993 |                       | contained no characters, the terminal shall be alerted and the cursor shall not be                |
| 114994 |                       | moved. If the <i>count</i> is larger than the number of characters before the cursor, this        |
| 114995 |                       | shall not be considered an error; all the characters from before the cursor to the                |
| 114996 |                       | beginning of the line shall be deleted.                                                           |
| 114997 | <b>[count]dmotion</b> |                                                                                                   |
| 114998 |                       | Delete the characters between the current cursor position and the character                       |
| 114999 |                       | position that would result from the motion command. A number <i>count</i> repeats the             |
| 115000 |                       | motion command <i>count</i> times. If the motion command would move toward the                    |
| 115001 |                       | beginning of the command line, the character under the current cursor position                    |
| 115002 |                       | shall not be deleted. If the motion command is <b>d</b> , the entire current command line         |
| 115003 |                       | shall be cleared. If the <i>count</i> is larger than the number of characters between the         |
| 115004 |                       | current cursor position and the end of the command line toward which the motion                   |
| 115005 |                       | command would move the cursor, this shall not be considered an error; all of the                  |
| 115006 |                       | remaining characters in the aforementioned range shall be deleted. The deleted                    |
| 115007 |                       | characters shall be placed in the save buffer.                                                    |



|        |                       |                                                                                               |
|--------|-----------------------|-----------------------------------------------------------------------------------------------|
| 115008 | <b>D</b>              | Delete all characters from the current cursor position to the end of the line. The            |
| 115009 |                       | deleted characters shall be placed in the save buffer.                                        |
| 115010 | <b>[count]ymotion</b> |                                                                                               |
| 115011 |                       | Yank (that is, copy) the characters from the current cursor position to the position          |
| 115012 |                       | resulting from the motion command into the save buffer. A number <i>count</i> shall be        |
| 115013 |                       | applied to the motion command. If the motion command would move toward the                    |
| 115014 |                       | beginning of the command line, the character under the current cursor position                |
| 115015 |                       | shall not be included in the set of yanked characters. If the motion command is <b>y</b> ,    |
| 115016 |                       | the entire current command line shall be yanked into the save buffer. The current             |
| 115017 |                       | cursor position shall be unchanged. If the <i>count</i> is larger than the number of          |
| 115018 |                       | characters between the current cursor position and the end of the command line                |
| 115019 |                       | toward which the motion command would move the cursor, this shall not be                      |
| 115020 |                       | considered an error; all of the remaining characters in the aforementioned range              |
| 115021 |                       | shall be yanked.                                                                              |
| 115022 | <b>Y</b>              | Yank the characters from the current cursor position to the end of the line into the          |
| 115023 |                       | save buffer. The current character position shall be unchanged.                               |
| 115024 | <b>[count]p</b>       | Put a copy of the current contents of the save buffer after the current cursor                |
| 115025 |                       | position. The current cursor position shall be advanced to the last character put             |
| 115026 |                       | from the save buffer. A <i>count</i> shall indicate how many copies of the save buffer        |
| 115027 |                       | shall be put.                                                                                 |
| 115028 | <b>[count]P</b>       | Put a copy of the current contents of the save buffer before the current cursor               |
| 115029 |                       | position. The current cursor position shall be moved to the last character put from           |
| 115030 |                       | the save buffer. A <i>count</i> shall indicate how many copies of the save buffer shall be    |
| 115031 |                       | put.                                                                                          |
| 115032 | <b>u</b>              | Undo the last command that changed the edit line. This operation shall not undo               |
| 115033 |                       | the copy of any command line to the edit line.                                                |
| 115034 | <b>U</b>              | Undo all changes made to the edit line. This operation shall not undo the copy of             |
| 115035 |                       | any command line to the edit line.                                                            |
| 115036 | <b>[count]k</b>       |                                                                                               |
| 115037 | <b>[count]-</b>       | Set the current command line to be the <i>count</i> th previous command line in the shell     |
| 115038 |                       | command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be |
| 115039 |                       | positioned on the first character of the new command. If a <b>k</b> or <b>-</b> command would |
| 115040 |                       | retreat past the maximum number of commands in effect for this shell (affected by             |
| 115041 |                       | the <i>HISTSIZE</i> environment variable), the terminal shall be alerted, and the             |
| 115042 |                       | command shall have no effect.                                                                 |
| 115043 | <b>[count]j</b>       |                                                                                               |
| 115044 | <b>[count]+</b>       | Set the current command line to be the <i>count</i> th next command line in the shell         |
| 115045 |                       | command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be |
| 115046 |                       | positioned on the first character of the new command. If a <b>j</b> or <b>+</b> command       |
| 115047 |                       | advances past the edit line, the current command line shall be restored to the edit           |
| 115048 |                       | line and the terminal shall be alerted.                                                       |
| 115049 | <b>[number]G</b>      | Set the current command line to be the oldest command line stored in the shell                |
| 115050 |                       | command history. With a number <i>number</i> , set the current command line to be the         |
| 115051 |                       | command line <i>number</i> in the history. If command line <i>number</i> does not exist, the  |
| 115052 |                       | terminal shall be alerted and the command line shall not be changed.                          |

115053 */pattern<newline>*  
 115054 Move backwards through the command history, searching for the specified  
 115055 pattern, beginning with the previous command line. Patterns use the pattern  
 115056 matching notation described in [Section 2.14](#) (on page 2523), except that the '^'  
 115057 character shall have special meaning when it appears as the first character of  
 115058 *pattern*. In this case, the '^' is discarded and the characters after the '^' shall be  
 115059 matched only at the beginning of a line. Commands in the command history shall  
 115060 be treated as strings, not as filenames. If the pattern is not found, the current  
 115061 command line shall be unchanged and the terminal shall be alerted. If it is found in  
 115062 a previous line, the current command line shall be set to that line and the cursor  
 115063 shall be set to the first character of the new command line.

115064 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If  
 115065 there is no previous non-empty pattern, the terminal shall be alerted and the  
 115066 current command line shall remain unchanged.

115067 *?pattern<newline>*  
 115068 Move forwards through the command history, searching for the specified pattern,  
 115069 beginning with the next command line. Patterns use the pattern matching notation  
 115070 described in [Section 2.14](#) (on page 2523), except that the '^' character shall have  
 115071 special meaning when it appears as the first character of *pattern*. In this case, the  
 115072 '^' is discarded and the characters after the '^' shall be matched only at the  
 115073 beginning of a line. Commands in the command history shall be treated as strings,  
 115074 not as filenames. If the pattern is not found, the current command line shall be  
 115075 unchanged and the terminal shall be alerted. If it is found in a following line, the  
 115076 current command line shall be set to that line and the cursor shall be set to the fist  
 115077 character of the new command line.

115078 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If  
 115079 there is no previous non-empty pattern, the terminal shall be alerted and the  
 115080 current command line shall remain unchanged.

115081 **n** Repeat the most recent / or ? command. If there is no previous / or ?, the terminal  
 115082 shall be alerted and the current command line shall remain unchanged.

115083 **N** Repeat the most recent / or ? command, reversing the direction of the search. If  
 115084 there is no previous / or ?, the terminal shall be alerted and the current command  
 115085 line shall remain unchanged.

115086 **EXIT STATUS**  
 115087 The following exit values shall be returned:

115088 0 The script to be executed consisted solely of zero or more blank lines or comments, or  
 115089 both.

115090 1-125 A non-interactive shell detected an error other than *command\_file* not found,  
 115091 *command\_file* not executable, or an unrecoverable read error while reading commands  
 115092 (except from the *file* operand of the *dot* special built-in); including but not limited to  
 115093 syntax, redirection, or variable assignment errors.

115094 126 A specified *command\_file* could not be executed due to an [ENOEXEC] error (see [Section](#)  
 115095 [2.9.1.4](#) (on page 2502), item 2).

115096 127 A specified *command\_file* could not be found by a non-interactive shell.

115097           128   An unrecoverable read error was detected while reading commands, except from the  
115098                    *file* operand of the *dot* special built-in.

115099           Otherwise, the shell shall terminate in the same manner as for an *exit* command with no  
115100           operands, unless the last command the shell invoked was executed without forking, in which  
115101           case the wait status seen by the parent process of the shell shall be the wait status of the last  
115102           command the shell invoked. See the *exit* utility in [Section 2.15](#) (on page 2526).

#### 115103 CONSEQUENCES OF ERRORS

115104           See [Section 2.8.1](#) (on page 2497).

#### 115105 APPLICATION USAGE

115106           Standard input and standard error are the files that determine whether a shell is interactive  
115107           when *-i* is not specified. For example:

```
115108           sh > file
```

115109           and:

```
115110           sh 2> file
```

115111           create interactive and non-interactive shells, respectively. Although both accept terminal input,  
115112           the results of error conditions are different, as described in [Section 2.8.1](#) (on page 2497); in the  
115113           second example a redirection error encountered by a special built-in utility aborts the shell.

115114           *sh -n* can be used to check for many syntax errors without waiting for *complete\_commands* to be  
115115           executed, but may be fooled into declaring false positives or missing actual errors that would  
115116           occur when the shell actually evaluates *eval* commands present in the script, or if there are *alias*  
115117           (or *unalias*) commands in the script that would alter the syntax of commands that use the  
115118           affected aliases.

115119           A conforming application must protect its first operand, if it starts with a <plus-sign>, by  
115120           preceding it with the "--" argument that denotes the end of the options.

115121           Applications should note that the standard *PATH* to the shell cannot be assumed to be either  
115122           */bin/sh* or */usr/bin/sh*, and should be determined by interrogation of the *PATH* returned by  
115123           *getconf PATH*, ensuring that the returned pathname is an absolute pathname and not a shell  
115124           built-in.

115125           For example, to determine the location of the standard *sh* utility:

```
115126           command -v sh
```

115127           On some implementations this might return:

```
115128           /usr/xpg4/bin/sh
```

115129           Furthermore, on systems that support executable scripts (the "#!" construct), it is  
115130           recommended that applications using executable scripts install them using *getconf PATH* to  
115131           determine the shell pathname and update the "#!" script appropriately as it is being installed  
115132           (for example, with *sed*). For example:

```
115133           #  
115134           # Installation time script to install correct POSIX shell pathname  
115135           #  
115136           # Get list of paths to check  
115137           #  
115138           Sifs=$IFS  
115139           Sifs_set=${IFS+y}  
115140           IFS=:
```

```

115141     set -- $(getconf PATH)
115142     if [ "$Sifs_set" = y ]
115143     then
115144         IFS=$Sifs
115145     else
115146         unset IFS
115147     fi
115148     #
115149     # Check each path for 'sh'
115150     #
115151     for i
115152     do
115153         if [ -x "${i}/sh ]
115154         then
115155             Pshell=${i}/sh
115156         fi
115157     done
115158     #
115159     # This is the list of scripts to update. They should be of the
115160     # form '${name}.source' and will be transformed to '${name}'.
115161     # Each script should begin:
115162     #
115163     # #!INSTALLSHELLPATH
115164     #
115165     scripts="a b c"
115166     #
115167     # Transform each script
115168     #
115169     for i in ${scripts}
115170     do
115171         sed -e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
115172     done

```

### 115173 EXAMPLES

- 115174 1. Execute a shell command from a string:
 

```
115175     sh -c "cat myfile"
```
- 115176 2. Execute a shell script from a file in the current directory:
 

```
115177     sh my_shell_cmds
```

### 115178 RATIONALE

115179 The *sh* utility and the *set* special built-in utility share a common set of options.

115180 The name *IFS* was originally an abbreviation of “Input Field Separators”; however, this name is  
 115181 misleading as the *IFS* characters are actually used as field terminators. One justification for  
 115182 ignoring the contents of *IFS* upon entry to the script, beyond security considerations, is to assist  
 115183 possible future shell compilers. Allowing *IFS* to be imported from the environment prevents  
 115184 many optimizations that might otherwise be performed via dataflow analysis of the script itself.

115185 The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been  
 115186 invoked, probably by a C-language program, with standard input that has been opened using  
 115187 the *O\_NONBLOCK* flag; see *open()* in the System Interfaces volume of POSIX.1-2024. If the shell  
 115188 did not reset this flag, it would immediately terminate because no input data would be available

115189 yet and that would be considered the same as end-of-file.

115190 The options associated with a *restricted shell* (command name *rsh* and the `-r` option) were  
115191 excluded because the standard developers considered that the implied level of security could  
115192 not be achieved and they did not want to raise false expectations.

115193 On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the  
115194 name `-i`. When it is called by a sequence such as:

115195 `sh -`

115196 or by:

115197 `#! usr/bin/sh -`

115198 the historical systems have assumed that no option letters follow. Thus, this volume of  
115199 POSIX.1-2024 allows the single `<hyphen-minus>` to mark the end of the options, in addition to  
115200 the use of the regular `--` argument, because it was considered that the older practice was so  
115201 pervasive. An alternative approach is taken by the KornShell, where real and effective  
115202 user/group IDs must match for an interactive shell; this behavior is specifically allowed by this  
115203 volume of POSIX.1-2024.

115204 **Note:** There are other problems with set-user-ID scripts that the two approaches described here do not  
115205 resolve.

115206 The initialization process for the history file can be dependent on the system start-up files, in  
115207 that they may contain commands that effectively preempt the user's settings of *HISTFILE* and  
115208 *HISTSIZ*E. In some historical shells, the history file is initialized just after the *ENV* file has been  
115209 processed. Therefore, it is implementation-defined whether changes made to *HISTFILE* after the  
115210 history file has been initialized are effective.

115211 The default messages for the various *MAIL*-related messages are unspecified because they vary  
115212 across implementations. Typical messages are:

115213 `"you have mail\n"`

115214 or:

115215 `"you have new mail\n"`

115216 It is important that the descriptions of command line editing refer to the same shell as that in  
115217 POSIX.1-2024 so that interactive users can also be application programmers without having to  
115218 deal with programmatic differences in their two environments. It is also essential that the utility  
115219 name *sh* be specified because this explicit utility name is too firmly rooted in historical practice  
115220 of application programs for it to change.

115221 Consideration was given to mandating a diagnostic message when attempting to set *vi*-mode on  
115222 terminals that do not support command line editing. However, it is not historical practice for the  
115223 shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in  
115224 all cases. Implementations are encouraged to supply diagnostics in this case whenever possible,  
115225 rather than leaving the user in a state where editing commands work incorrectly.

115226 In early proposals, the KornShell-derived *emacs* mode of command line editing was included,  
115227 even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant  
115228 that the full *emacs* editor not be standardized because they were concerned that an attempt to  
115229 standardize this very powerful environment would encourage vendors to ship strictly  
115230 conforming versions lacking the extensibility required by the community. The author of the  
115231 original *emacs* program also expressed his desire to omit the program. Furthermore, there were a  
115232 number of historical systems that did not include *emacs*, or included it without supporting it, but  
115233 there were very few that did not include and support *vi*. The shell *emacs* command line editing

115234 mode was finally omitted because it became apparent that the KornShell version and the editor  
 115235 being distributed with the GNU system had diverged in some respects. The author of *emacs*  
 115236 requested that the POSIX *emacs* mode either be deleted or have a significant number of  
 115237 unspecified conditions. Although the KornShell author agreed to consider changes to bring the  
 115238 shell into alignment, the standard developers decided to defer specification at that time. At the  
 115239 time, it was assumed that convergence on an acceptable definition would occur for a subsequent  
 115240 draft, but that has not happened, and there appears to be no impetus to do so. In any case,  
 115241 implementations are free to offer additional command line editing modes based on the exact  
 115242 models of editors their users are most comfortable with.

115243 Early proposals had the following list entry in *vi* [Line Editing Insert Mode](#) (on page 3371):

115244 \ If followed by the *erase* or *kill* character, that character shall be inserted into the input line.  
 115245 Otherwise, the <backslash> itself shall be inserted into the input line.

115246 However, this is not actually a feature of *sh* command line editing insert mode, but one of some  
 115247 historical terminal line drivers. Some conforming implementations continue to do this when the  
 115248 *stty ixten* flag is set.

115249 In interactive shells, SIGTERM is ignored so that `kill 0` does not kill the shell, and SIGINT is  
 115250 caught so that *wait* is interruptible. If the shell does not ignore SIGTTIN, SIGTTOU, and  
 115251 SIGTSTP signals when it is interactive and the `-m` option is not in effect, these signals suspend  
 115252 the shell if it is not a session leader. If it is a session leader, the signals are discarded if they  
 115253 would stop the process, as required by XSH [Section 2.4.3](#) (on page 516) for orphaned process  
 115254 groups.

115255 Earlier versions of this standard required that input files to the shell be text files except that line  
 115256 lengths were unlimited. However, that was overly restrictive in relation to the fact that shells can  
 115257 parse a script without a trailing newline, and in relation to a common practice of concatenating a  
 115258 shell script ending with an `exit` or `exec $command` with a binary data payload to form a  
 115259 single-file self-extracting archive.

#### 115260 FUTURE DIRECTIONS

115261 If this utility is directed to create a new directory entry that contains any bytes that have the  
 115262 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 115263 error. A future version of this standard may require implementations to treat this as an error.

#### 115264 SEE ALSO

115265 [Section 2.9.1.4](#) (on page 2502), [Chapter 2](#) (on page 2472), *cd*, *echo*, *exit*, *fc*, *pwd*, *invalid*, *set*, *stty*,  
 115266 *test*, *trap*, *umask*, *vi*

115267 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

115268 XSH *dup()*, *exec()*, *exit()*, *fork()*, *getrlimit()*, *open()*, *pipe()*, *signal()*, *system()*, *umask()*, *wait()*

#### 115269 CHANGE HISTORY

115270 First released in Issue 2.

#### 115271 Issue 5

115272 The FUTURE DIRECTIONS section is added.

115273 Text is added to the DESCRIPTION for the Large File Summit proposal.

#### 115274 Issue 6

115275 The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.

115276 The Open Group Corrigendum U027/3 is applied, correcting a typographical error.

115277 The following new requirements on POSIX implementations derive from alignment with the

- 115278 Single UNIX Specification:
- 115279 • The option letters derived from the *set* special built-in are also accepted with a leading  
115280 <plus-sign> ('+').
  - 115281 • Large file extensions are added:
    - 115282 — Pathname expansion does not fail due to the size of a file.
    - 115283 — Shell input and output redirections have an implementation-defined offset maximum  
115284 that is established in the open file description.
- 115285 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to  
115286 “directory referred to by the *HOME* environment variable”.
- 115287 Descriptions for the *ENV* and *PWD* environment variables are included to align with the  
115288 IEEE P1003.2b draft standard.
- 115289 The normative text is reworded to avoid use of the term “must” for application requirements.
- 115290 **Issue 7**
- 115291 Austin Group Interpretation 1003.1-2001 #098 is applied, changing the definition of *IFS*.
- 115292 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 115293 Changes to the *pwd* utility and *PWD* environment variable have been made to match the  
115294 changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.
- 115295 Minor editorial changes are made to the User Portability Utilities option shading. No normative  
115296 changes are implied.
- 115297 Minor changes are made to the install script example in the APPLICATION USAGE section.
- 115298 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0137 [152], XCU/TC1-2008/0138  
115299 [347], XCU/TC1-2008/0139 [347], XCU/TC1-2008/0140 [347], XCU/TC1-2008/0141 [299], and  
115300 XCU/TC1-2008/0142 [347] are applied.
- 115301 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0175 [584], XCU/TC2-2008/0176  
115302 [584], XCU/TC2-2008/0177 [718], XCU/TC2-2008/0178 [884], XCU/TC2-2008/0179 [809],  
115303 XCU/TC2-2008/0180 [884], and XCU/TC2-2008/0181 [584] are applied.
- 115304 **Issue 8**
- 115305 Austin Group Defect 51 is applied, changing the EXIT STATUS section.
- 115306 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
115307 filenames containing any bytes that have the encoded value of a <newline> character.
- 115308 Austin Group Defect 981 is applied, removing a reference to the *set* **-o nolog** option from the  
115309 RATIONALE section.
- 115310 Austin Group Defect 1006 is applied, changing the description of the *ENV* environment variable.
- 115311 Austin Group Defect 1055 is applied, adding a paragraph about the **-n** option to the  
115312 APPLICATION USAGE section.
- 115313 Austin Group Defect 1063 is applied, adding OB shading to the **-h** option and adding it to the  
115314 list of options that are described as part of the *set* utility.
- 115315 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 115316 Austin Group Defect 1250 is applied, changing the INPUT FILES section.
- 115317 Austin Group Defect 1266 is applied, clarifying the circumstances under which the shell is

- 115318 considered to be interactive.
- 115319 Austin Group Defect 1267 is applied, changing the ENVIRONMENT VARIABLES section to  
115320 remove the UP shading from *HOME* and add it to *HISTSIZE*.
- 115321 Austin Group Defect 1519 is applied, making the behavior explicitly unspecified if the *-o* or *+o*  
115322 option is specified without an option-argument.
- 115323 Austin Group Defect 1629 is applied, changing the EXIT STATUS section.



115324 **NAME**

115325 sleep — suspend execution for an interval

115326 **SYNOPSIS**115327 sleep *time*115328 **DESCRIPTION**115329 The *sleep* utility shall suspend execution for at least the integral number of seconds specified by  
115330 the *time* operand.115331 **OPTIONS**

115332 None.

115333 **OPERANDS**

115334 The following operand shall be supported:

115335 *time* A non-negative decimal integer specifying the number of seconds for which to  
115336 suspend execution.115337 **STDIN**

115338 Not used.

115339 **INPUT FILES**

115340 None.

115341 **ENVIRONMENT VARIABLES**115342 The following environment variables shall affect the execution of *sleep*:115343 *LANG* Provide a default value for the internationalization variables that are unset or null.  
115344 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
115345 variables used to determine the values of locale categories.)115346 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
115347 internationalization variables.115348 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
115349 characters (for example, single-byte as opposed to multi-byte characters in  
115350 arguments).115351 *LC\_MESSAGES*115352 Determine the locale that should be used to affect the format and contents of  
115353 diagnostic messages written to standard error.115354 XSI *NLSPATH* Determine the location of messages objects and message catalogs.115355 **ASYNCHRONOUS EVENTS**115356 If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

- 115357 1. Terminate normally with a zero exit status.
- 
- 115358 2. Effectively ignore the signal.
- 
- 115359 3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS
- 
- 115360 section of
- [Section 1.4](#)
- (on page 2462). This could include terminating with a non-zero exit
- 
- 115361 status.

115362 The *sleep* utility shall take the standard action for all other signals.

115363 **STDOUT**

115364 Not used.

115365 **STDERR**

115366 The standard error shall be used only for diagnostic messages.

115367 **OUTPUT FILES**

115368 None.

115369 **EXTENDED DESCRIPTION**

115370 None.

115371 **EXIT STATUS**

115372 The following exit values shall be returned:

115373 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal  
115374 was received. See the ASYNCHRONOUS EVENTS section.

115375 &gt;0 An error occurred.

115376 **CONSEQUENCES OF ERRORS**

115377 Default.

115378 **APPLICATION USAGE**

115379 None.

115380 **EXAMPLES**115381 The *sleep* utility can be used to execute a command after a certain amount of time, as in:115382 `(sleep 105; command) &`

115383 or to execute a command every so often, as in:

115384 `while true`  
115385 `do`  
115386  `command`  
115387  `sleep 37`  
115388 `done`115389 **RATIONALE**115390 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because  
115391 most implementations of this utility rely on the arrival of that signal to notify them that the  
115392 requested finishing time has been successfully attained. Such implementations thus do not  
115393 distinguish this situation from the successful completion case. Other implementations are  
115394 allowed to catch the signal and go back to sleep until the requested time expires or to provide  
115395 the normal signal termination procedures.115396 As with all other utilities that take integral operands and do not specify subranges of allowed  
115397 values, *sleep* is required by this volume of POSIX.1-2024 to deal with *time* requests of up to  
115398 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls to  
115399 the delay mechanism of the underlying operating system if its argument range is less than this.115400 **FUTURE DIRECTIONS**

115401 None.

115402 **SEE ALSO**115403 [\*wait\*](#)115404 [XBD Chapter 8](#) (on page 167)115405 [XSH \*alarm\(\)\*, \*sleep\(\)\*](#)

115406 **CHANGE HISTORY**

115407 First released in Issue 2.

115408 **Issue 8**

115409 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

115410 **NAME**

115411 sort — sort, merge, or sequence check text files

115412 **SYNOPSIS**115413 sort [-m] [-o *output*] [-bdfinru] [-t *char*] [-k *keydef*]... [*file*...]115414 sort [-c|-C] [-bdfinru] [-t *char*] [-k *keydef*] [*file*]115415 **DESCRIPTION**115416 The *sort* utility shall perform one of the following functions:

- 115417 1. Sort lines of all the named files together and write the result to the specified output.
- 115418 2. Merge lines of all the named (presorted) files together and write the result to the specified  
115419 output.
- 115420 3. Check that a single input file is correctly presorted.

115421 Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no  
115422 sort keys are specified, the entire line up to, but not including, the terminating <newline>), and  
115423 shall be performed using the collating sequence of the current locale. If this collating sequence  
115424 does not have a total ordering of all characters (see XBD Section 7.3.2, on page 139), any lines of  
115425 input that collate equally shall be further compared byte-by-byte using the collating sequence  
115426 for the POSIX locale.

115427 **OPTIONS**

115428 The *sort* utility shall conform to XBD Section 12.2 (on page 215), except for Guideline 9, and the  
115429 **-k** *keydef* option should follow the **-b**, **-d**, **-f**, **-i**, **-n**, and **-r** options. In addition, '+' may be  
115430 recognized as an option delimiter as well as '-'.

115431 The following options shall be supported:

- 115432 **-c** Check that the single input file is ordered as specified by the arguments and the  
115433 collating sequence of the current locale. Output shall not be sent to standard  
115434 output. The exit code shall indicate whether or not disorder was detected or an  
115435 error occurred. If disorder (or, with **-u**, a duplicate key) is detected, a warning  
115436 message shall be sent to standard error indicating where the disorder or duplicate  
115437 key was found.
- 115438 **-C** Same as **-c**, except that a warning message shall not be sent to standard error if  
115439 disorder or, with **-u**, a duplicate key is detected.
- 115440 **-m** Merge only; the input file shall be assumed to be already sorted.
- 115441 **-o** *output* Specify the name of an output file to be used instead of the standard output. This  
115442 file can be the same as one of the input *files*.
- 115443 **-u** Unique: suppress all but one in each set of lines having equal keys. If used with  
115444 the **-c** option, check that there are no lines with duplicate keys, in addition to  
115445 checking that the input file is sorted.

115446 The following options shall override the default ordering rules. When ordering options appear  
115447 independent of any key field specifications, the requested field ordering rules shall be applied  
115448 globally to all sort keys. When attached to a specific key (see **-k**), the specified ordering options  
115449 shall override all global ordering options for that key.

- 115450 **-d** Specify that only <blank> characters and alphanumeric characters, according to  
115451 the current setting of *LC\_CTYPE*, shall be significant in comparisons. The behavior  
115452 is undefined for a sort key to which **-i** or **-n** also applies.

- 115453        **-f**            Consider all lowercase characters that have uppercase equivalents, according to  
115454                    the current setting of *LC\_CTYPE*, to be the uppercase equivalent for the purposes  
115455                    of comparison.
- 115456        **-i**            Ignore all characters that are non-printable, according to the current setting of  
115457                    *LC\_CTYPE*. The behavior is undefined for a sort key for which **-n** also applies.
- 115458        **-n**            Restrict the sort key to an initial numeric string, consisting of optional <blank>  
115459                    characters, optional <hyphen-minus> character, and zero or more digits with an  
115460                    optional radix character and thousands separators (as defined in the current  
115461                    locale), which shall be sorted by arithmetic value. An empty digit string shall be  
115462                    treated as zero. Leading zeros and signs on zeros shall not affect ordering.
- 115463        **-r**            Reverse the sense of comparisons.
- 115464        The treatment of field separators can be altered using the options:
- 115465        **-b**            Ignore leading <blank> characters when determining the starting and ending  
115466                    positions of a restricted sort key. If the **-b** option is specified before the first **-k**  
115467                    option, it shall be applied to all **-k** options. Otherwise, the **-b** option can be  
115468                    attached independently to each **-k** *field\_start* or *field\_end* option-argument (see  
115469                    below).
- 115470        **-t char**        Use *char* as the field separator character; *char* shall not be considered to be part of a  
115471                    field (although it can be included in a sort key). Each occurrence of *char* shall be  
115472                    significant (for example, <*char*><*char*> delimits an empty field). If **-t** is not  
115473                    specified, <blank> characters shall be used as default field separators; each  
115474                    maximal non-empty sequence of <blank> characters that follows a non-<blank>  
115475                    shall be a field separator.
- 115476        Sort keys can be specified using the options:
- 115477        **-k keydef**        The *keydef* argument is a restricted sort key field definition. The format of this  
115478                    definition is:
- 115479                    *field\_start*[*type*][,*field\_end*[*type*]]
- 115480                    where *field\_start* and *field\_end* define a key field restricted to a portion of the line  
115481                    (see the EXTENDED DESCRIPTION section), and *type* is one or more modifiers  
115482                    from the list of characters 'b', 'd', 'f', 'i', 'n', 'r'. The 'b' modifier shall  
115483                    behave like the **-b** option, but shall apply only to the *field\_start* or *field\_end* to  
115484                    which it is attached. The other modifiers shall behave like the corresponding  
115485                    options, but shall apply only to the key field to which they are attached; they shall  
115486                    have this effect if specified with *field\_start*, *field\_end*, or both. If any modifier is  
115487                    attached to a *field\_start* or to a *field\_end*, no option shall apply to either.  
115488                    Implementations shall support at least nine occurrences of the **-k** option, which  
115489                    shall be significant in command line order. If no **-k** option is specified, a default  
115490                    sort key of the entire line shall be used.
- 115491                    When there are multiple key fields, later keys shall be compared only after all  
115492                    earlier keys compare equal. Except when the **-u** option is specified, lines that  
115493                    otherwise compare equal shall be ordered as if none of the options **-d**, **-f**, **-i**, **-n**, or  
115494                    **-k** were present (but with **-r** still in effect, if it was specified) and with all bytes in  
115495                    the lines significant to the comparison. The order in which lines that still compare  
115496                    equal are written is unspecified.

115497 **OPERANDS**

115498 The following operand shall be supported:

115499 *file* A pathname of a file to be sorted, merged, or checked. If no *file* operands are  
 115500 specified, or if a *file* operand is '-', the standard input shall be used. If *sort*  
 115501 encounters an error when opening or reading a *file* operand, it may exit without  
 115502 writing any output to standard output or processing later operands.

115503 **STDIN**

115504 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.  
 115505 See the INPUT FILES section.

115506 **INPUT FILES**

115507 The input files shall be text files, except that the *sort* utility shall add a <newline> to the end of a  
 115508 file ending with an incomplete last line.

115509 **ENVIRONMENT VARIABLES**

115510 The following environment variables shall affect the execution of *sort*:

115511 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 115512 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 115513 variables used to determine the values of locale categories.)

115514 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 115515 internationalization variables.

115516 *LC\_COLLATE*

115517 Determine the locale for ordering rules.

115518 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 115519 characters (for example, single-byte as opposed to multi-byte characters in  
 115520 arguments and input files) and the behavior of character classification for the **-b**,  
 115521 **-d**, **-f**, **-i**, and **-n** options.

115522 *LC\_MESSAGES*

115523 Determine the locale that should be used to affect the format and contents of  
 115524 diagnostic messages written to standard error.

115525 *LC\_NUMERIC*

115526 Determine the locale for the definition of the radix character and thousands  
 115527 separator for the **-n** option.

115528 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

115529 *TMPDIR* Provide a pathname that shall override the default directory for temporary files, if  
 115530 any.

115531 **ASYNCHRONOUS EVENTS**

115532 Default.

115533 **STDOUT**

115534 Unless the **-o** or **-c** options are in effect, the standard output shall contain the sorted input.

115535 **STDERR**

115536 The standard error shall be used for diagnostic messages. When **-c** is specified, if disorder is  
 115537 detected (or if **-u** is also specified and a duplicate key is detected), a message shall be written to  
 115538 the standard error which identifies the input line at which disorder (or a duplicate key) was  
 115539 detected. A warning message about correcting an incomplete last line of an input file may be  
 115540 generated, but need not affect the final exit status.

115541 **OUTPUT FILES**

115542 If the `-o` option is in effect, the sorted input shall be written to the file *output*.

115543 **EXTENDED DESCRIPTION**

115544 The notation:

115545 `-k field_start[type] [, field_end[type]]`

115546 shall define a key field that begins at *field\_start* and ends at *field\_end* inclusive, unless *field\_start*  
115547 falls beyond the end of the line or after *field\_end*, in which case the key field is empty. A missing  
115548 *field\_end* shall mean the last character of the line.

115549 A field comprises a maximal sequence of non-separating characters and, in the absence of option  
115550 `-t`, any preceding field separator.

115551 The *field\_start* portion of the *keydef* option-argument shall have the form:

115552 `field_number[.first_character]`

115553 Fields and characters within fields shall be numbered starting with 1. The *field\_number* and  
115554 *first\_character* pieces, interpreted as positive decimal integers, shall specify the first character to  
115555 be used as part of a sort key. If *first\_character* is omitted, it shall refer to the first character of the  
115556 field.

115557 The *field\_end* portion of the *keydef* option-argument shall have the form:

115558 `field_number[.last_character]`

115559 The *field\_number* shall be as described above for *field\_start*. The *last\_character* piece, interpreted  
115560 as a non-negative decimal integer, shall specify the last character to be used as part of the sort  
115561 key. If *last\_character* evaluates to zero or *last\_character* is omitted, it shall refer to the last  
115562 character of the field specified by *field\_number*.

115563 If the `-b` option or `b` type modifier is in effect, characters within a field shall be counted from the  
115564 first non-`<blank>` in the field. (This shall apply separately to *first\_character* and *last\_character*.)

115565 **EXIT STATUS**

115566 The following exit values shall be returned:

115567 0 All input files were output successfully, or `-c` was specified and the input file was correctly  
115568 sorted.

115569 1 Under the `-c` option, the file was not ordered as specified, or if the `-c` and `-u` options were  
115570 both specified, two input lines were found with equal keys.

115571 `>1` An error occurred.

115572 **CONSEQUENCES OF ERRORS**

115573 The default requirements shall apply, except that if *sort* encounters an error when opening or  
115574 reading a *file* operand, it may exit without writing any output to standard output or processing  
115575 later operands.

## 115576 APPLICATION USAGE

115577 The default value for `-t`, `<blank>`, has different properties from, for example, `-t"<space>"`. If a  
 115578 line contains:

115579 `<space><space>foo`

115580 the following treatment would occur with default separation as opposed to specifically selecting  
 115581 a `<space>`:

| Field | Default                                    | <code>-t "&lt;space&gt;"</code> |
|-------|--------------------------------------------|---------------------------------|
| 1     | <code>&lt;space&gt;&lt;space&gt;foo</code> | <i>empty</i>                    |
| 2     | <i>empty</i>                               | <i>empty</i>                    |
| 3     | <i>empty</i>                               | foo                             |

115586 The leading field separator itself is included in a field when `-t` is not used. For example, this  
 115587 command returns an exit status of zero, meaning the input was already sorted:

```
115588 sort -c -k 2 <<eof
115589 y<tab>b
115590 x<space>a
115591 eof
```

115592 (assuming that a `<tab>` precedes the `<space>` in the current collating sequence). The field  
 115593 separator is not included in a field when it is explicitly set via `-t`. This is historical practice and  
 115594 allows usage such as:

```
115595 sort -t "|" -k 2n <<eof
115596 Atlanta|425022|Georgia
115597 Birmingham|284413|Alabama
115598 Columbia|100385|South Carolina
115599 eof
```

115600 where the second field can be correctly sorted numerically without regard to the non-numeric  
 115601 field separator.

115602 The wording in the OPTIONS section clarifies that the `-b`, `-d`, `-f`, `-i`, `-n`, and `-r` options have to  
 115603 come before the first sort key specified if they are intended to apply to all specified keys. The  
 115604 way it is described in this volume of POSIX.1-2024 matches historical practice, not historical  
 115605 documentation. The results are unspecified if these options are specified after a `-k` option.

115606 The `-f` option might not work as expected in locales where there is not a one-to-one mapping  
 115607 between an uppercase and a lowercase letter.

115608 When using `sort` to process pathnames, it is recommended that `LC_ALL`, or at least `LC_CTYPE`  
 115609 and `LC_COLLATE`, are set to `POSIX` or `C` in the environment, since pathnames can contain byte  
 115610 sequences that do not form valid characters in some locales, in which case the utility's behavior  
 115611 would be undefined. In the `POSIX` locale each byte is a valid single-byte character, and therefore  
 115612 this problem is avoided.

115613 If the collating sequence of the current locale does not have a total ordering of all characters,  
 115614 since `sort -u` suppresses lines with duplicate keys, it suppresses lines that collate equally but are  
 115615 not identical.

## 115616 EXAMPLES

115617 1. The following command sorts the contents of `infile` with the second field as the sort key:

```
115618 sort -k 2,2 infile
```



115619 2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**,  
 115620 placing the output in **outfile** and using the second character of the second field as the sort  
 115621 key (assuming that the first character of the second field is the field separator):

```
115622 sort -r -o outfile -k 2.2,2.2 infile1 infile2
```

115623 3. The following command sorts the contents of **infile1** and **infile2** using the second  
 115624 non-<blank> of the second field as the sort key:

```
115625 sort -k 2.2b,2.2b infile1 infile2
```

115626 4. The following command prints the System V password file (user database) sorted by the  
 115627 numeric user ID (the third <colon>-separated field):

```
115628 sort -t : -k 3,3n /etc/passwd
```

115629 5. The following command prints the lines of the already sorted file **infile**, suppressing all  
 115630 but one occurrence of lines having the same third field:

```
115631 sort -um -k 3.1,3.0 infile
```

### 115632 RATIONALE

115633 Examples in some historical documentation state that options **-um** with one input file keep the  
 115634 first in each set of lines with equal keys. This behavior was deemed to be an implementation  
 115635 artifact and was not standardized.

115636 The **-z** option was omitted; it is not standard practice on most systems and is inconsistent with  
 115637 using *sort* to sort several files individually and then merge them together. The text concerning **-z**  
 115638 in historical documentation appeared to require implementations to determine the proper buffer  
 115639 length during the sort phase of operation, but not during the merge.

115640 The **-y** option was omitted because of non-portability. The **-M** option, present in System V, was  
 115641 omitted because of non-portability in international usage.

115642 An undocumented **-T** option exists in some implementations. It is used to specify a directory for  
 115643 intermediate files. Implementations are encouraged to support the use of the *TMPDIR*  
 115644 environment variable instead of adding an option to support this functionality.

115645 The **-k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is  
 115646 not consistent with other utility conventions. Second, it did not meet syntax guideline  
 115647 requirements.

115648 Historical documentation indicates that “setting **-n** implies **-b**”. The description of **-n** already  
 115649 states that optional leading <blank>s are tolerated in doing the comparison. If **-b** is enabled,  
 115650 rather than implied, by **-n**, this has unusual side-effects. When a character offset is used in a  
 115651 column of numbers (for example, to sort modulo 100), that offset is measured relative to the  
 115652 most significant digit, not to the column. Based upon a recommendation from the author of the  
 115653 original *sort* utility, the **-b** implication has been omitted from this volume of POSIX.1-2024, and  
 115654 an application wishing to achieve the previously mentioned side-effects has to code the **-b** flag  
 115655 explicitly.

115656 Earlier versions of this standard allowed the **-o** option to appear after operands. Historical  
 115657 practice allowed all options to be interspersed with operands. This version of the standard  
 115658 allows implementations to accept options after operands but conforming applications should  
 115659 not use this form.

115660 Earlier versions of this standard also allowed the *-number* and *+number* options. These options  
 115661 are no longer specified by POSIX.1-2024 but may be present in some implementations.

115662 Historical implementations produced a message on standard error when **-c** was specified and

115663 disorder was detected, and when `-c` and `-u` were specified and a duplicate key was detected. An  
 115664 earlier version of this standard contained wording that did not make it clear that this message  
 115665 was allowed and some implementations removed this message to be sure that they conformed  
 115666 to the standard's requirements. Confronted with this difference in behavior, interactive users  
 115667 that wanted to be sure that they got visual feedback instead of just exit code 1 could have used a  
 115668 command like:

```
115669 sort -c file || echo disorder
```

115670 whether or not the *sort* utility provided a message in this case. But, it was not easy for a user to  
 115671 find where the disorder or duplicate key occurred on implementations that do not produce a  
 115672 message, especially when some parts of the input line were not part of the key and when one or  
 115673 more of the `-b`, `-d`, `-f`, `-i`, `-n`, or `-r` options or *keydef* type modifiers were in use. POSIX.1-2024  
 115674 requires a message to be produced in this case. POSIX.1-2024 also contains the `-C` option giving  
 115675 users the ability to choose either behavior.

115676 When a disorder or duplicate is found when the `-c` option is specified, some implementations  
 115677 print a message containing the first line that is out of order or contains a duplicate key; others  
 115678 print a message specifying the line number of the offending line. This standard allows either  
 115679 type of message.

115680 The required further byte-by-byte comparison of lines that collate equally may have an impact  
 115681 on efficiency, but this can be mitigated by only performing the additional comparison if the  
 115682 current locale's collating sequence does not have a total ordering of all characters (if the  
 115683 implementation provides a way to query this) or by only performing the additional comparison  
 115684 if the locale name associated with the *LC\_COLLATE* category has an '@' modifier in the name  
 115685 (since implementation-supplied locales without an '@' modifier have a total ordering of all  
 115686 characters — see XBD Section 7.3.2 (on page 139) — and *localedef* users are warned to follow the  
 115687 same convention). Note that if the implementation provides a *stable sort* option as an extension  
 115688 (usually `-s`), the additional comparison should not be performed when this option has been  
 115689 specified.

#### 115690 FUTURE DIRECTIONS

115691 If this utility is directed to create a new directory entry that contains any bytes that have the  
 115692 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 115693 error. A future version of this standard may require implementations to treat this as an error.

#### 115694 SEE ALSO

115695 *comm*, *join*, *uniq*

115696 XBD Section 7.3.2 (on page 139), Chapter 8 (on page 167), Section 12.2 (on page 215)

115697 XSH *toupper*()

#### 115698 CHANGE HISTORY

115699 First released in Issue 2.

#### 115700 Issue 6

115701 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

115702 IEEE PASC Interpretation 1003.2 #168 is applied.

#### 115703 Issue 7

115704 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility  
 115705 Syntax Guidelines does not apply and noting that '+' may be recognized as an option delimiter.

115706 Austin Group Interpretation 1003.1-2001 #120 is applied, clarifying the use of the `-c` option and  
 115707 introducing the `-C` option.

- 115708 XCU-ERN-81 is applied, modifying the description of the `-i` option.
- 115709 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 115710 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0182 [963], XCU/TC2-2008/0183  
115711 [584], XCU/TC2-2008/0184 [510], XCU/TC2-2008/0185 [962], XCU/TC2-2008/0186 [663], and  
115712 XCU/TC2-2008/0187 [963] are applied.
- 115713 **Issue 8**
- 115714 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
115715 filenames containing any bytes that have the encoded value of a <newline> character.
- 115716 Austin Group Defect 862 is applied, adding `TMPDIR` to the ENVIRONMENT VARIABLES  
115717 section.
- 115718 Austin Group Defect 1070 is applied, requiring that any lines of input that collate equally when  
115719 comparing them as whole lines are further compared byte-by-byte using the collating sequence  
115720 for the POSIX locale.
- 115721 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.

115722 **NAME**115723 `split` — split a file into pieces115724 **SYNOPSIS**115725 `split [-l line_count] [-a suffix_length] [file [name]]`115726 `split -b n[k|m] [-a suffix_length] [file [name]]`115727 **DESCRIPTION**

115728 The *split* utility shall read an input file and write zero or more output files. The default size of  
 115729 each output file shall be 1 000 lines. The size of the output files can be modified by specification  
 115730 of the `-b` or `-l` options. Each output file shall be created with a unique suffix. The suffix shall  
 115731 consist of exactly *suffix\_length* lowercase letters from the POSIX locale. The letters of the suffix  
 115732 shall be used as if they were a base-26 digit system, with the first suffix to be created consisting  
 115733 of all 'a' characters, the second with a 'b' replacing the last 'a', and so on, until a name of all  
 115734 'z' characters is created. By default, the names of the output files shall be 'x', followed by a  
 115735 two-character suffix from the character set as described above, starting with "aa", "ab", "ac",  
 115736 and so on, and continuing until the suffix "zz", for a maximum of 676 files.

115737 If the number of files required exceeds the maximum allowed by the suffix length provided,  
 115738 such that the last allowable file would be larger than the requested size, the *split* utility shall fail  
 115739 after creating the last file with a valid suffix; *split* shall not delete the files it created with valid  
 115740 suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the  
 115741 input file, and may be smaller than the requested size. If the input is an empty file, no output file  
 115742 shall be created and this shall not be considered to be an error.

115743 **OPTIONS**115744 The *split* utility shall conform to XBD [Section 12.2](#) (on page 215).

115745 The following options shall be supported:

115746 `-a suffix_length`

115747 Use *suffix\_length* letters to form the suffix portion of the filenames of the split file. If  
 115748 `-a` is not specified, the default suffix length shall be two. If the sum of the *name*  
 115749 operand and the *suffix\_length* option-argument would create a filename exceeding  
 115750 {NAME\_MAX} bytes, an error shall result; *split* shall exit with a diagnostic message  
 115751 and no files shall be created.

115752 `-b n` Split a file into pieces *n* bytes in size.115753 `-b nk` Split a file into pieces *n*\*1 024 bytes in size.115754 `-b nm` Split a file into pieces *n*\*1 048 576 bytes in size.

115755 `-l line_count` Specify the number of lines in each resulting file piece. The *line\_count* argument is  
 115756 an unsigned decimal integer. The default is 1 000. If the input does not end with a  
 115757 <newline>, the partial line shall be included in the last output file.

115758 **OPERANDS**

115759 The following operands shall be supported:

115760 *file* The pathname of the ordinary file to be split. If no input file is given or *file* is '-',  
 115761 the standard input shall be used.

115762 *name* The prefix to be used for each of the files resulting from the split operation. If no  
 115763 *name* argument is given, 'x' shall be used as the prefix of the output files. The  
 115764 combined length of the basename of *prefix* and *suffix\_length* cannot exceed  
 115765 {NAME\_MAX} bytes. See the OPTIONS section.

115766 **STDIN**

115767 See the INPUT FILES section.

115768 **INPUT FILES**

115769 Any file can be used as input.

115770 **ENVIRONMENT VARIABLES**115771 The following environment variables shall affect the execution of *split*:

115772 **LANG** Provide a default value for the internationalization variables that are unset or null.  
115773 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
115774 variables used to determine the values of locale categories.)

115775 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
115776 internationalization variables.

115777 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
115778 characters (for example, single-byte as opposed to multi-byte characters in  
115779 arguments and input files).

115780 **LC\_MESSAGES**

115781 Determine the locale that should be used to affect the format and contents of  
115782 diagnostic messages written to standard error.

115783 **XSI NLSPATH** Determine the location of messages objects and message catalogs.

115784 **ASYNCHRONOUS EVENTS**

115785 Default.

115786 **STDOUT**

115787 Not used.

115788 **STDERR**

115789 The standard error shall be used only for diagnostic messages.

115790 **OUTPUT FILES**

115791 The output files contain portions of the original input file; otherwise, unchanged.

115792 **EXTENDED DESCRIPTION**

115793 None.

115794 **EXIT STATUS**

115795 The following exit values shall be returned:

115796 0 Successful completion.

115797 &gt;0 An error occurred.

115798 **CONSEQUENCES OF ERRORS**

115799 Default.

115800 **APPLICATION USAGE**

115801 None.

115802 **EXAMPLES**115803 In the following examples **foo** is a text file that contains 5 000 lines.115804 1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:115805 

```
split foo
```

115806 2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**,  
115807 **xaab**, **xaac**, **xaad**, and **xaae**:115808 

```
split -a 3 foo
```

115809 3. Create three files with four-letter suffixes and a supplied prefix, **bar\_aaaa**, **bar\_aaab**, and  
115810 **bar\_aaac**:115811 

```
split -a 4 -l 2000 foo bar_
```

115812 4. Create as many files as are necessary to contain at most 20\*1 024 bytes, each with the  
115813 default prefix of **x** and a five-letter suffix:115814 

```
split -a 5 -b 20k foo
```

115815 **RATIONALE**115816 The **-b** option was added to provide a mechanism for splitting files other than by lines. While  
115817 most uses of the **-b** option are for transmitting files over networks, some believed it would have  
115818 additional uses.115819 The **-a** option was added to overcome the limitation of being able to create only 676 files.115820 Consideration was given to deleting this utility, using the rationale that the functionality  
115821 provided by this utility is available via the *csplit* utility (see *csplit*). Upon reconsideration of the  
115822 purpose of the User Portability Utilities option, it was decided to retain both this utility and the  
115823 *csplit* utility because users use both utilities and have historical expectations of their behavior.  
115824 Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical  
115825 *csplit*.115826 The text “*split* shall not delete the files it created with valid suffixes” would normally be  
115827 assumed, but since the related utility, *csplit*, does delete files under some circumstances, the  
115828 historical behavior of *split* is made explicit to avoid misinterpretation.115829 Earlier versions of this standard allowed a *-line\_count* option. This form is no longer specified by  
115830 POSIX.1-2024 but may be present in some implementations.115831 **FUTURE DIRECTIONS**115832 If this utility is directed to create a new directory entry that contains any bytes that have the  
115833 encoded value of a <newline> character, implementations are encouraged to treat this as an  
115834 error. A future version of this standard may require implementations to treat this as an error.115835 **SEE ALSO**115836 [\*csplit\*](#)115837 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)115838 **CHANGE HISTORY**

115839 First released in Issue 2.

115840 **Issue 6**

115841 This utility is marked as part of the User Portability Utilities option.

115842 The APPLICATION USAGE section is added.

115843 The obsolescent SYNOPSIS is removed.

115844 **Issue 7**

115845 Austin Group Interpretation 1003.1-2001 #027 is applied.

115846 The *split* utility is moved from the User Portability Utilities option to the Base. User Portability  
115847 Utilities is now an option for interactive utilities.

115848 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

115849 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0188 [731] is applied.

115850 **Issue 8**

115851 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
115852 filenames containing any bytes that have the encoded value of a <newline> character.

115853 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

115854 **NAME**

115855 strings — find printable strings in files

115856 **SYNOPSIS**115857 strings [-a] [-t *format*] [-n *number*] [*file...*]115858 **DESCRIPTION**

115859 The *strings* utility shall look for printable strings in regular files and shall write those strings to  
 115860 standard output. A printable string is any sequence of four (by default) or more printable  
 115861 characters terminated by a <newline> or NUL character. Additional implementation-defined  
 115862 strings may be written; see *localedef*.

115863 If any argument is '-', the results are unspecified.

115864 **OPTIONS**

115865 The *strings* utility shall conform to XBD [Section 12.2](#) (on page 215), except for the unspecified  
 115866 usage of '-'.

115867 The following options shall be supported:

115868 **-a** Scan files in their entirety. If **-a** is not specified, it is implementation-defined what  
 115869 portion of each file is scanned for strings.

115870 **-n *number*** Specify the minimum string length, where the *number* argument is a positive  
 115871 decimal integer. The default shall be 4.

115872 **-t *format*** Write each string preceded by its byte offset from the start of the file. The format  
 115873 shall be dependent on the single character used as the *format* option-argument:

115874 d The offset shall be written in decimal.

115875 o The offset shall be written in octal.

115876 x The offset shall be written in hexadecimal.

115877 **OPERANDS**

115878 The following operand shall be supported:

115879 *file* A pathname of a regular file to be used as input. If no *file* operand is specified, the  
 115880 *strings* utility shall read from the standard input.

115881 **STDIN**

115882 See the INPUT FILES section.

115883 **INPUT FILES**

115884 The input files named by the utility arguments or the standard input shall be regular files of any  
 115885 format.

115886 **ENVIRONMENT VARIABLES**115887 The following environment variables shall affect the execution of *strings*:

115888 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 115889 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 115890 variables used to determine the values of locale categories.)

115891 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 115892 internationalization variables.

115893 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 115894 characters (for example, single-byte as opposed to multi-byte characters in  
 115895 arguments and input files) and to identify printable strings.



- 115896 **LC\_MESSAGES**  
 115897 Determine the locale that should be used to affect the format and contents of  
 115898 diagnostic messages written to standard error.
- 115899 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 115900 **ASYNCHRONOUS EVENTS**  
 115901 Default.
- 115902 **STDOUT**  
 115903 Strings found shall be written to the standard output, one per line.  
 115904 When the `-t` option is not specified, the format of the output shall be:  
 115905 `"%s", <string>`  
 115906 With the `-t o` option, the format of the output shall be:  
 115907 `"%o %s", <byte offset>, <string>`  
 115908 With the `-t x` option, the format of the output shall be:  
 115909 `"%x %s", <byte offset>, <string>`  
 115910 With the `-t d` option, the format of the output shall be:  
 115911 `"%d %s", <byte offset>, <string>`
- 115912 **STDERR**  
 115913 The standard error shall be used only for diagnostic messages.
- 115914 **OUTPUT FILES**  
 115915 None.
- 115916 **EXTENDED DESCRIPTION**  
 115917 None.
- 115918 **EXIT STATUS**  
 115919 The following exit values shall be returned:  
 115920 0 Successful completion.  
 115921 >0 An error occurred.
- 115922 **CONSEQUENCES OF ERRORS**  
 115923 Default.
- 115924 **APPLICATION USAGE**  
 115925 By default the data area (as opposed to the text, ```bss```, or header areas) of a binary executable  
 115926 file is scanned. Implementations document which areas are scanned.  
 115927 Some historical implementations do not require NUL or `<newline>` terminators for strings to  
 115928 permit those languages that do not use NUL as a string terminator to have their strings written.
- 115929 **EXAMPLES**  
 115930 None.
- 115931 **RATIONALE**  
 115932 Apart from rationalizing the option syntax and slight difficulties with object and executable  
 115933 binary files, *strings* is specified to match historical practice closely. The `-a` and `-n` options were  
 115934 introduced to replace the non-conforming `-` and `-number` options. These options are no longer  
 115935 specified by POSIX.1-2024 but may be present in some implementations.

- 115936 The `-o` option historically means different things on different implementations. Some use it to mean “offset in decimal”, while others use it as “offset in octal”. Instead of trying to decide which way would be least objectionable, the `-t` option was added. It was originally named `-O` to mean “offset”, but was changed to `-t` to be consistent with *od*.
- 115937
- 115938
- 115939
- 115940 The ISO C standard function *isprint()* is restricted to a domain of **unsigned char**. This volume of
- 115941 POSIX.1-2024 requires implementations to write strings as defined by the current locale.
- 115942 **FUTURE DIRECTIONS**
- 115943 None.
- 115944 **SEE ALSO**
- 115945 *localedef, nm*
- 115946 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)
- 115947 **CHANGE HISTORY**
- 115948 First released in Issue 4.
- 115949 **Issue 6**
- 115950 This utility is marked as part of the User Portability Utilities option.
- 115951 The obsolescent SYNOPSIS is removed.
- 115952 The normative text is reworded to avoid use of the term “must” for application requirements.
- 115953 **Issue 7**
- 115954 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
- 115955 argument is `'-'`.
- 115956 The *strings* utility is moved from the User Portability Utilities option to the Base. User
- 115957 Portability Utilities is now an option for interactive utilities.
- 115958 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 115959 **Issue 8**
- 115960 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 115961 Austin Group Defect 1599 is applied, making the behavior unspecified when any argument is
- 115962 `'-'`.

115963 **NAME**115964 strip — remove unnecessary information from strippable files (**DEVELOPMENT**)115965 **SYNOPSIS**115966 SD strip *file...*115967 **DESCRIPTION**115968 XSI A strippable file is defined as a relocatable, object, or executable file. On XSI-conformant  
115969 systems, a strippable file can also be an archive of object or relocatable files.115970 The *strip* utility shall remove from strippable files named by the *file* operands any information  
115971 the implementor deems unnecessary for execution of those files. The nature of that information  
115972 is unspecified. The effect of *strip* on object and executable files shall be similar to the use of the  
115973 XSI **-s** option to *c17*. The effect of *strip* on an archive of object files shall be similar to the use of the  
115974 **-s** option to *c17* for each object file in the archive.115975 **OPTIONS**

115976 None.

115977 **OPERANDS**

115978 The following operand shall be supported:

115979 *file* A pathname referring to a strippable file.115980 **STDIN**

115981 Not used.

115982 **INPUT FILES**115983 The input files shall be in the form of strippable files successfully produced by any compiler  
115984 XSI defined by this volume of POSIX.1-2024 or produced by creating or updating an archive of such  
115985 files using the *ar* utility.115986 **ENVIRONMENT VARIABLES**115987 The following environment variables shall affect the execution of *strip*:115988 *LANG* Provide a default value for the internationalization variables that are unset or null.  
115989 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
115990 variables used to determine the values of locale categories.)115991 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
115992 internationalization variables.115993 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
115994 characters (for example, single-byte as opposed to multi-byte characters in  
115995 arguments).115996 *LC\_MESSAGES*115997 Determine the locale that should be used to affect the format and contents of  
115998 diagnostic messages written to standard error.115999 XSI *NLSPATH* Determine the location of messages objects and message catalogs.116000 **ASYNCHRONOUS EVENTS**

116001 Default.

116002 **STDOUT**

116003 Not used.

**116004 STDERR**

116005 The standard error shall be used only for diagnostic messages.

**116006 OUTPUT FILES**

116007 The *strip* utility shall produce strippable files of unspecified format.

**116008 EXTENDED DESCRIPTION**

116009 None.

**116010 EXIT STATUS**

116011 The following exit values shall be returned:

116012 0 Successful completion.

116013 >0 An error occurred.

**116014 CONSEQUENCES OF ERRORS**

116015 Default.

**116016 APPLICATION USAGE**

116017 None.

**116018 EXAMPLES**

116019 None.

**116020 RATIONALE**

116021 Historically, this utility has been used to remove the symbol table from a strippable file. It was  
116022 included since it is known that the amount of symbolic information can amount to several  
116023 megabytes; the ability to remove it in a portable manner was deemed important, especially for  
116024 smaller systems.

116025 The behavior of *strip* on object and executable files is said to be the same as the `-s` option to a  
116026 compiler. While the end result is essentially the same, it is not required to be identical.

116027 XSI-conformant systems support use of *strip* on archive files containing object files or relocatable  
116028 files.

**116029 FUTURE DIRECTIONS**

116030 None.

**116031 SEE ALSO**

116032 [ar](#), [c17](#)

116033 [XBD Chapter 8](#) (on page 167)

**116034 CHANGE HISTORY**

116035 First released in Issue 2.

**116036 Issue 6**

116037 This utility is marked as part of the Software Development Utilities option.

**116038 Issue 7**

116039 Austin Group Interpretation 1003.1-2001 #103 is applied.

**116040 Issue 8**

116041 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

116042 Austin Group Defect 1330 is applied, removing obsolescent interfaces.

116043 **NAME**116044 `stty` — set the options for a terminal116045 **SYNOPSIS**116046 `stty [-a|-g]`116047 `stty operand...`116048 **DESCRIPTION**

116049 The `stty` utility shall set or report on terminal I/O characteristics for the device that is its  
 116050 standard input. Without options or operands specified, it shall report the settings of certain  
 116051 characteristics, usually those that differ from implementation-defined defaults. Otherwise, it  
 116052 shall modify the terminal state according to the specified operands. Detailed information about  
 116053 the modes listed in the first five groups below are described in XBD [Chapter 11](#) (on page 199).  
 116054 Operands in the Combination Modes group (see [Combination Modes](#), on page 3410) are  
 116055 implemented using operands in the previous groups. Some combinations of operands are  
 116056 mutually-exclusive on some terminal types; the results of using such combinations are  
 116057 unspecified.

116058 Typical implementations of this utility require a communications line configured to use the  
 116059 **termios** interface defined in the System Interfaces volume of POSIX.1-2024. On systems where  
 116060 none of these lines are available, and on lines not currently configured to support the **termios**  
 116061 interface, some of the operands need not affect terminal characteristics.

116062 **OPTIONS**116063 The `stty` utility shall conform to XBD [Section 12.2](#) (on page 215).

116064 The following options shall be supported:

- 116065 **-a** Write to standard output all the current settings for the terminal.
- 116066 **-g** Write to standard output all the current settings, optionally excluding the terminal  
 116067 window size, in an unspecified form that, when stripped of trailing `<newline>`  
 116068 characters, and used as the one and only argument to another invocation of the `stty`  
 116069 utility on the same system, attempts to apply those settings to the terminal. The  
 116070 form used shall not contain any sequence that would form an Informational Query,  
 116071 and shall consist of one line of text consisting of only printable characters from the  
 116072 portable character set, excluding white-space characters (other than the  
 116073 terminating `<newline>`) and these characters that could be altered by pathname  
 116074 expansion performed by the shell: `'*'`, `'?'`, and `'['`.

116075 **OPERANDS**

116076 The following operands shall be supported.

116077 **Control Modes**

116078 **par**en**b** (**-par**en**b**) Enable (disable) parity generation and detection. This shall have the effect of  
 116079 setting (not setting) PARENb in the **termios** `c_flag` field, as defined in XBD  
 116080 [Chapter 11](#) (on page 199).

116081 **par**od**d** (**-par**od**d**)  
 116082 Select odd (even) parity. This shall have the effect of setting (not setting)  
 116083 PARODD in the **termios** `c_flag` field, as defined in XBD [Chapter 11](#) (on page  
 116084 199).

116085 **cs**5** cs**6** cs**7** cs**8**** Select character size, if possible. This shall have the effect of setting CS5, CS6,  
 116086 CS7, and CS8, respectively, in the **termios** `c_flag` field, as defined in XBD  
 116087 [Chapter 11](#) (on page 199).

|        |                                  |                                                                                                                                                                                                                                                                                                                               |
|--------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 116088 | <i>number</i>                    | Set terminal baud rate to the number given, if possible. If the baud rate is set to zero, the modem control lines shall no longer be asserted. This shall have the effect of setting the input and output <b>termios</b> baud rate values as defined in XBD <a href="#">Chapter 11</a> (on page 199).                         |
| 116089 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116090 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116091 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116092 | <b>ispeed</b> <i>number</i>      | Set terminal input baud rate to the number given, if possible. If the input baud rate is set to zero, the input baud rate shall be specified by the value of the output baud rate. This shall have the effect of setting the input <b>termios</b> baud rate value as defined in XBD <a href="#">Chapter 11</a> (on page 199). |
| 116093 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116094 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116095 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116096 | <b>ospeed</b> <i>number</i>      | Set terminal output baud rate to the number given, if possible. If the output baud rate is set to zero, the modem control lines shall no longer be asserted. This shall have the effect of setting the output <b>termios</b> baud rate value as defined in XBD <a href="#">Chapter 11</a> (on page 199).                      |
| 116097 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116098 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116099 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116100 | <b>hupcl</b> ( <b>-hupcl</b> )   | Stop asserting modem control lines (do not stop asserting modem control lines) on last close. This shall have the effect of setting (not setting) HUPCL in the <b>termios</b> <i>c_cflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                               |
| 116101 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116102 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116103 | <b>hup</b> ( <b>-hup</b> )       | Equivalent to <b>hupcl</b> ( <b>-hupcl</b> ).                                                                                                                                                                                                                                                                                 |
| 116104 | <b>cstopb</b> ( <b>-cstopb</b> ) | Use two (one) stop bits per character. This shall have the effect of setting (not setting) CSTOPB in the <b>termios</b> <i>c_cflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                     |
| 116105 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116106 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116107 | <b>cread</b> ( <b>-cread</b> )   | Enable (disable) the receiver. This shall have the effect of setting (not setting) CREAD in the <b>termios</b> <i>c_cflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                              |
| 116108 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116109 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116110 | <b>clocal</b> ( <b>-clocal</b> ) | Assume a line without (with) modem control. This shall have the effect of setting (not setting) CLOCAL in the <b>termios</b> <i>c_cflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                |
| 116111 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116112 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116113 |                                  | It is unspecified whether <i>stty</i> shall report an error if an attempt to set a Control Mode fails.                                                                                                                                                                                                                        |
| 116114 | <b>Input Modes</b>               |                                                                                                                                                                                                                                                                                                                               |
| 116115 | <b>ignbrk</b> ( <b>-ignbrk</b> ) | Ignore (do not ignore) break on input. This shall have the effect of setting (not setting) IGNBRK in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                     |
| 116116 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116117 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116118 | <b>brkint</b> ( <b>-brkint</b> ) | Signal (do not signal) INTR on break. This shall have the effect of setting (not setting) BRKINT in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                      |
| 116119 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116120 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116121 | <b>ignpar</b> ( <b>-ignpar</b> ) | Ignore (do not ignore) bytes with parity errors. This shall have the effect of setting (not setting) IGNPAR in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                           |
| 116122 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116123 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116124 | <b>parmrk</b> ( <b>-parmrk</b> ) | Mark (do not mark) parity errors. This shall have the effect of setting (not setting) PARMRK in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                          |
| 116125 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116126 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116127 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116128 | <b>inpck</b> ( <b>-inpck</b> )   | Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                     |
| 116129 |                                  |                                                                                                                                                                                                                                                                                                                               |
| 116130 |                                  |                                                                                                                                                                                                                                                                                                                               |

|            |                                  |                                                                                                                                                                                                                                                                                                                          |
|------------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 116131     | <b>istrip</b> ( <b>-istrip</b> ) | Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                  |
| 116132     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116133     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116134     | <b>inlcr</b> ( <b>-inlcr</b> )   | Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                    |
| 116135     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116136     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116137     | <b>igncr</b> ( <b>-igncr</b> )   | Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                    |
| 116138     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116139     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116140     | <b>icrnl</b> ( <b>-icrnl</b> )   | Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                    |
| 116141     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116142     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116143     | <b>ixon</b> ( <b>-ixon</b> )     | Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199). |
| 116144     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116145     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116146     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116147     | <b>ixany</b> ( <b>-ixany</b> )   | Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                 |
| 116148     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116149     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116150     | <b>ixoff</b> ( <b>-ixoff</b> )   | Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the <b>termios</b> <i>c_iflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).          |
| 116151     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116152     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116153     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116154     | <b>Output Modes</b>              |                                                                                                                                                                                                                                                                                                                          |
| 116155     | <b>opost</b> ( <b>-opost</b> )   | Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                       |
| 116156     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116157     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116158 XSI | <b>onlcr</b> ( <b>-onlcr</b> )   | Map (do not map) NL to CR-NL on output. This shall have the effect of setting (not setting) ONLCR in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                |
| 116159     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116160     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116161 XSI | <b>ocrnl</b> ( <b>-ocrnl</b> )   | Map (do not map) CR to NL on output. This shall have the effect of setting (not setting) OCRNL in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                   |
| 116162     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116163     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116164 XSI | <b>onocr</b> ( <b>-onocr</b> )   | Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                  |
| 116165     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116166     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116167 XSI | <b>onlret</b> ( <b>-onlret</b> ) | The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                 |
| 116168     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116169     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116170 XSI | <b>ofill</b> ( <b>-ofill</b> )   | Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                           |
| 116171     |                                  |                                                                                                                                                                                                                                                                                                                          |
| 116172     |                                  |                                                                                                                                                                                                                                                                                                                          |

|        |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
|--------|-----|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 116173 | XSI | <b>ofdel</b> ( <b>-ofdel</b> )   | Fill characters are DELs (NULs). This shall have the effect of setting (not setting) OFDEL in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                                  |
| 116174 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116175 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116176 | XSI | <b>cr0 cr1 cr2 cr3</b>           | Select the style of delay for CRs. This shall have the effect of setting CRDLY to CR0, CR1, CR2, or CR3, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                      |
| 116177 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116178 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116179 | XSI | <b>nl0 nl1</b>                   | Select the style of delay for NL. This shall have the effect of setting NLDLY to NL0 or NL1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                  |
| 116180 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116181 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116182 | XSI | <b>tab0 tab1 tab2 tab3</b>       | Select the style of delay for horizontal tabs. This shall have the effect of setting TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199). Note that TAB3 has the effect of expanding <tab> characters to <space> characters.  |
| 116183 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116184 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116185 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116186 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116187 | XSI | <b>tabs</b> ( <b>-tabs</b> )     | Synonym for <b>tab0</b> ( <b>tab3</b> ).                                                                                                                                                                                                                                                                                            |
| 116188 | XSI | <b>bs0 bs1</b>                   | Select the style of delay for <backspace> characters. This shall have the effect of setting BSDLY to BS0 or BS1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                              |
| 116189 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116190 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116191 | XSI | <b>ff0 ff1</b>                   | Select the style of delay for <form-feed> characters. This shall have the effect of setting FFDLY to FF0 or FF1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                              |
| 116192 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116193 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116194 | XSI | <b>vt0 vt1</b>                   | Select the style of delay for <vertical-tab> characters. This shall have the effect of setting VTDLY to VT0 or VT1, respectively, in the <b>termios</b> <i>c_oflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                           |
| 116195 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116196 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116197 |     | <b>Local Modes</b>               |                                                                                                                                                                                                                                                                                                                                     |
| 116198 |     | <b>isig</b> ( <b>-isig</b> )     | Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the <b>termios</b> <i>c_lflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                           |
| 116199 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116200 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116201 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116202 |     | <b>icanon</b> ( <b>-icanon</b> ) | Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the <b>termios</b> <i>c_lflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                    |
| 116203 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116204 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116205 |     | <b>iexten</b> ( <b>-iexten</b> ) | Enable (disable) any implementation-defined special control characters not currently controlled by <b>icanon</b> , <b>isig</b> , <b>ixon</b> , or <b>ixoff</b> . This shall have the effect of setting (not setting) IEXTEN in the <b>termios</b> <i>c_lflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199). |
| 116206 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116207 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116208 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116209 |     | <b>echo</b> ( <b>-echo</b> )     | Echo back (do not echo back) every character typed. This shall have the effect of setting (not setting) ECHO in the <b>termios</b> <i>c_lflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                                                                                                |
| 116210 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116211 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116212 |     | <b>echoe</b> ( <b>-echoe</b> )   | The ERASE character visually erases (does not erase) the last character in the current line from the display, if possible. This shall have the effect of setting (not setting) ECHOE in the <b>termios</b> <i>c_lflag</i> field, as defined in XBD <a href="#">Chapter 11</a> (on page 199).                                        |
| 116213 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116214 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |
| 116215 |     |                                  |                                                                                                                                                                                                                                                                                                                                     |



- 116216 **echok** (**-echok**) Echo (do not echo) NL after KILL character. This shall have the effect of setting  
 116217 (not setting) ECHOK in the **termios** *c\_lflag* field, as defined in XBD [Chapter 11](#)  
 116218 (on page 199).
- 116219 **echonl** (**-echonl**) Echo (do not echo) NL, even if **echo** is disabled. This shall have the effect of  
 116220 setting (not setting) ECHONL in the **termios** *c\_lflag* field, as defined in XBD  
 116221 [Chapter 11](#) (on page 199).
- 116222 **noflsh** (**-noflsh**) Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of  
 116223 setting (not setting) NOFLSH in the **termios** *c\_lflag* field, as defined in XBD  
 116224 [Chapter 11](#) (on page 199).
- 116225 **tostop** (**-tostop**) Send SIGTTOU for background output. This shall have the effect of setting  
 116226 (not setting) TOSTOP in the **termios** *c\_lflag* field, as defined in XBD [Chapter 11](#)  
 116227 (on page 199).

### 116228 Special Control Character Assignments

116229 *<control>-character string*

116230 Set *<control>-character* to *string*. If *<control>-character* is one of the character sequences in the  
 116231 first column of the following table, the corresponding XBD [Chapter 11](#) (on page 199) control  
 116232 character from the second column shall be recognized. This has the effect of setting the  
 116233 corresponding element of the **termios** *c\_cc* array (see XBD [Chapter 14](#) (on page 221),  
 116234 **<termios.h>**).

116235 **Table 3-20** Control Character Names in *stty*

| Control Character | <i>c_cc</i> Subscript | Description     |
|-------------------|-----------------------|-----------------|
| <b>eof</b>        | VEOF                  | EOF character   |
| <b>eol</b>        | VEOL                  | EOL character   |
| <b>erase</b>      | VERASE                | ERASE character |
| <b>intr</b>       | VINTR                 | INTR character  |
| <b>kill</b>       | VKILL                 | KILL character  |
| <b>quit</b>       | VQUIT                 | QUIT character  |
| <b>susp</b>       | VSUSP                 | SUSP character  |
| <b>start</b>      | VSTART                | START character |
| <b>stop</b>       | VSTOP                 | STOP character  |

116246 If *string* is a single character, the control character shall be set to that character. If *string* is the  
 116247 two-character sequence "**^**-" or the string *undef*, the control character shall be set to  
 116248 **\_POSIX\_VDISABLE**, if it is in effect for the device; if **\_POSIX\_VDISABLE** is not in effect for  
 116249 the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character  
 116250 sequence beginning with **<circumflex>** ('**^**'), and the second character is one of those listed  
 116251 in the "**^c**" column of the following table, the control character shall be set to the  
 116252 corresponding character value in the Value column of the table.

116253

Table 3-21 Circumflex Control Characters in *stty*

116254

116255

116256

116257

116258

116259

116260

116261

116262

116263

116264

116265

| <b>^c</b> | <b>Value</b> | <b>^c</b> | <b>Value</b> | <b>^c</b> | <b>Value</b> |
|-----------|--------------|-----------|--------------|-----------|--------------|
| a, A      | <SOH>        | l, L      | <FF>         | w, W      | <ETB>        |
| b, B      | <STX>        | m, M      | <CR>         | x, X      | <CAN>        |
| c, C      | <ETX>        | n, N      | <SO>         | y, Y      | <EM>         |
| d, D      | <EOT>        | o, O      | <SI>         | z, Z      | <SUB>        |
| e, E      | <ENQ>        | p, P      | <DLE>        | [         | <ESC>        |
| f, F      | <ACK>        | q, Q      | <DC1>        | \         | <FS>         |
| g, G      | <BEL>        | r, R      | <DC2>        | ]         | <GS>         |
| h, H      | <BS>         | s, S      | <DC3>        | ^         | <RS>         |
| i, I      | <HT>         | t, T      | <DC4>        | _         | <US>         |
| j, J      | <LF>         | u, U      | <NAK>        | ?         | <DEL>        |
| k, K      | <VT>         | v, V      | <SYN>        |           |              |

116266

**min number**

116267

Set the value of MIN to *number*. MIN is used in non-canonical mode input processing (**-icanon**).

116268

116269

**time number**

116270

Set the value of TIME to *number*. TIME is used in non-canonical mode input processing (**-icanon**).

116271

116272

**Combination Modes**

116273

**saved settings**

116274

Set the current terminal characteristics to the saved settings produced by the **-g** option.

116275

**evenp or parity**

116276

Enable **parenb** and **cs7**; disable **parodd**.

116277

**oddp**

116278

Enable **parenb**, **cs7**, and **parodd**.

116279

**-parity, -evenp, or -oddp**

116280

Disable **parenb**, and set **cs8**.

116281 XSI

**raw (-raw or cooked)**

116282

Enable (disable) raw input and output. Raw mode shall be equivalent to setting:

116283

```
stty cs8 erase ^- kill ^- intr ^- \
quit ^- eof ^- eol ^- -post -inpck
```

116284

116285

**nl (-nl)**

116286

Disable (enable) **icrnl**. In addition, **-nl** unsets **inlcr** and **igncr**.

116287

**ek** Reset ERASE and KILL characters back to system defaults.

116288

**sane**

116289

Reset all modes to some reasonable, unspecified, values.

116290 **Terminal Window Size**116291 **rows** *number*

116292 Set the number of rows in the terminal window size to the number given.

116293 **cols** *number*

116294 Set the number of columns in the terminal window size to the number given.

116295 The terminal window size shall be updated as if the *stty* utility calls *tcgetwinsize()* to populate a  
 116296 **winsize** structure, updates one or both of the *ws\_row* and *ws\_col* members according to the **rows**  
 116297 and **cols** numbers specified, and then calls *tcsetwinsize()* with the updated structure (see XSH  
 116298 *tcgetwinsize()* and *tcsetwinsize()*).

116299 **Informational Queries**116300 **size**

116301 Write the current terminal window size to standard output.

116302 **STDIN**

116303 Although no input is read from standard input, standard input shall be used to get the current  
 116304 terminal I/O characteristics and to set new terminal I/O characteristics.

116305 **INPUT FILES**

116306 None.

116307 **ENVIRONMENT VARIABLES**116308 The following environment variables shall affect the execution of *stty*:

116309 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 116310 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 116311 variables used to determine the values of locale categories.)

116312 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 116313 internationalization variables.

116314 **LC\_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of  
 116315 text data as characters (for example, single-byte as opposed to multi-byte  
 116316 characters in arguments) and which characters are in the class **print**.

116317 **LC\_MESSAGES**

116318 Determine the locale that should be used to affect the format and contents of  
 116319 diagnostic messages written to standard error.

116320 XSI **NLSPATH** Determine the location of messages objects and message catalogs.116321 **ASYNCHRONOUS EVENTS**

116322 Default.

116323 **STDOUT**

116324 If operands are specified and they do not include any Informational Queries, no output shall be  
 116325 produced.

116326 If the **size** operand is specified, *stty* shall write to standard output the terminal window size as  
 116327 follows:

116328 "%1dΔ%1d\n", *<rows>*, *<columns>*

116329 where *<rows>* and *<columns>* are the number of rows and columns in the terminal window size,  
 116330 respectively.

116331 If the **-g** option is specified, *stty* shall write to standard output the current settings in a form that

116332 can be used as arguments to another instance of *stty* on the same system.

116333 If the **-a** option is specified, all of the information as described in the OPERANDS section shall  
 116334 be written to standard output. Unless otherwise specified, this information shall be written as  
 116335 <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified  
 116336 number of tokens per line. Additional information may be written.

116337 If no options or operands are specified, an unspecified subset of the information written for the  
 116338 **-a** option shall be written.

116339 If speed information is written as part of the default output, or if the **-a** option is specified and if  
 116340 the terminal input speed and output speed are the same, the speed information shall be written  
 116341 as follows:

116342 "speed %d baud;", <speed>

116343 Otherwise, speeds shall be written as:

116344 "ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>

116345 In locales other than the POSIX locale, the word **baud** may be changed to something more  
 116346 appropriate in those locales.

116347 If control characters are written as part of the default output, or if the **-a** option is specified,  
 116348 control characters shall be written as:

116349 "%s = %s;", <control-character name>, <value>

116350 where <value> is either the character, or some visual representation of the character if it is non-  
 116351 printable, or the string "<undef>" if the character is disabled.

**116352 STDERR**

116353 The standard error shall be used only for diagnostic messages.

**116354 OUTPUT FILES**

116355 None.

**116356 EXTENDED DESCRIPTION**

116357 None.

**116358 EXIT STATUS**

116359 The following exit values shall be returned:

116360 0 Successful completion.

116361 >0 An error occurred.

**116362 CONSEQUENCES OF ERRORS**

116363 Default.

**116364 APPLICATION USAGE**

116365 The **-g** flag is designed to facilitate the saving and restoring of terminal state from the shell level.  
 116366 For example, a program may:

```

116367 saveterm=$(stty -g)                                # save terminal state
116368 restoresize=$(
116369     printf "stty rows %d cols %d" $(stty size)
116370 )  # save terminal size
116371 stty new settings                                  # set new state
116372 ...
116373 [ -n "$saveterm" ] && stty "$saveterm"             # restore terminal state
  
```

- 116374           eval "\$restoresize"                               # restore terminal size
- 116375           Since the format is unspecified, the saved value is not portable across systems.
- 116376           Since the `-a` format is so loosely specified, scripts that save and restore terminal settings should  
116377           use the `-g` option.
- 116378 **EXAMPLES**
- 116379           None.
- 116380 **RATIONALE**
- 116381           The original *stty* description was taken directly from System V and reflected the System V  
116382           terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.
- 116383           Output modes are specified only for XSI-conformant systems. All implementations are expected  
116384           to provide *stty* operands corresponding to all of the output modes they support.
- 116385           The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the  
116386           preferred ERASE and KILL characters. As an application programming utility, *stty* can be used  
116387           within shell scripts to alter the terminal settings for the duration of the script.
- 116388           The **termios** section states that individual disabling of control characters is possible through the  
116389           option `_POSIX_VDISABLE`. If enabled, two conventions currently exist for specifying this:  
116390           System V uses `^~`, and BSD uses *undef*. Both are accepted by *stty* in this volume of  
116391           POSIX.1-2024. The other BSD convention of using the letter 'u' was rejected because it conflicts  
116392           with the actual letter 'u', which is an acceptable value for a control character.
- 116393           Early proposals did not specify the mapping of `^c` to control characters because the control  
116394           characters were not specified in the POSIX locale character set description file requirements. The  
116395           control character set is now specified in XBD [Chapter 3](#) (on page 31), so the historical mapping is  
116396           specified. Note that although the mapping corresponds to control-character key assignments on  
116397           many terminals that use the ISO/IEC 646:1991 standard (or ASCII) character encodings, the  
116398           mapping specified here is to the control characters, not their keyboard encodings.
- 116399           Since **termios** supports separate speeds for input and output, two new options were added to  
116400           specify each distinctly.
- 116401           Some historical implementations use standard input to get and set terminal characteristics;  
116402           others use standard output. Since input from a login TTY is usually restricted to the owner while  
116403           output to a TTY is frequently open to anyone, using standard input provides fewer chances of  
116404           accidentally (or maliciously) altering the terminal settings of other users. Using standard input  
116405           also allows *stty -a* and *stty -g* output to be redirected for later use. Therefore, usage of standard  
116406           input is required by this volume of POSIX.1-2024.
- 116407 **FUTURE DIRECTIONS**
- 116408           None.
- 116409 **SEE ALSO**
- 116410           [Chapter 2](#) (on page 2472)
- 116411           XBD [Chapter 8](#) (on page 167), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 215), [<termios.h>](#)
- 116412 **CHANGE HISTORY**
- 116413           First released in Issue 2.
- 116414 **Issue 5**
- 116415           The description of **tabs** is clarified.
- 116416           The FUTURE DIRECTIONS section is added.

116417 **Issue 6**

116418 The LEGACY items **iuclc** (**-iuclc**), **xcase** (**-xcase**), **olcuc** (**-olcuc**), **lcase** (**-lcase**), and **LCASE**  
116419 (**-LCASE**) are removed.

116420 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/37 is applied, applying IEEE PASC  
116421 Interpretation 1003.2 #133, fixing an error in the OPERANDS section for the Combination Modes  
116422 **nl** (**-nl**).

116423 **Issue 7**

116424 Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the  
116425 IXANY symbol from the XSI option to the Base.

116426 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

116427 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0189 [908] is applied.

116428 **Issue 8**

116429 Austin Group Defects 1053, 1532, and 1687 are applied, changing the **-g** option and adding the  
116430 **rows number**, **cols number**, and **size** operands.

116431 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

116432 Austin Group Defect 1508 is applied, changing the EXIT STATUS section.

116433 Austin Group Defect 1604 is applied, changing *undef* to "`<undef>`" in the STDOUT section, and  
116434 changing **icanon** to **-icanon** in the descriptions of **min** and **time**.

116435 **NAME**

116436 tabs — set terminal tabs

116437 **SYNOPSIS**116438 XSI tabs [-n | -a | -a2 | -c | -c2 | -c3 | -f | -p | -s | -u ] [-T *type*]116439 tabs [-T *type*] n [[*sep*[+]n] . . . ]116440 **DESCRIPTION**

116441 The *tabs* utility shall display a series of characters that first clears the hardware terminal tab  
 116442 XSI settings and then initializes the tab stops at the specified positions and optionally adjusts the  
 116443 margin.

116444 The phrase “tab-stop position *N*” shall be taken to mean that, from the start of a line of output,  
 116445 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position  
 116446 on that line. The maximum number of tab stops allowed is terminal-dependent.

116447 It need not be possible to implement *tabs* on certain terminals. If the terminal type obtained from  
 116448 the *TERM* environment variable or *-T* option represents such a terminal, an appropriate  
 116449 diagnostic message shall be written to standard error and *tabs* shall exit with a status greater  
 116450 than zero.

116451 **OPTIONS**

116452 XSI The *tabs* utility shall conform to XBD Section 12.2 (on page 215), except for various extensions:  
 116453 the options *-a2*, *-c2*, and *-c3* are multi-character.

116454 The following options shall be supported:

116455 *-n* Specify repetitive tab stops separated by a uniform number of column positions, *n*,  
 116456 where *n* is a single-digit decimal number. The default usage of *tabs* with no  
 116457 arguments shall be equivalent to *tabs -8*. When *-0* is used, the tab stops shall be  
 116458 cleared and no new ones set.

116459 XSI *-a* 1,10,16,36,72  
 116460 Assembler, applicable to some mainframes.

116461 XSI *-a2* 1,10,16,40,72  
 116462 Assembler, applicable to some mainframes.

116463 XSI *-c* 1,8,12,16,20,55  
 116464 COBOL, normal format.

116465 XSI *-c2* 1,6,10,14,49  
 116466 COBOL, compact format (columns 1 to 6 omitted).

116467 XSI *-c3* 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67  
 116468 COBOL compact format (columns 1 to 6 omitted), with more tabs than *-c2*.

116469 XSI *-f* 1,7,11,15,19,23  
 116470 FORTRAN

116471 XSI *-p* 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  
 116472 PL/1

116473 XSI *-s* 1,10,55  
 116474 SNOBOL

116475 XSI *-u* 1,12,20,44  
 116476 Assembler, applicable to some mainframes.

116477            -T *type*        Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
 116478                                    is unset or null, an unspecified default terminal type shall be used. The setting of  
 116479                                    *type* shall take precedence over the value in *TERM*.

116480 **OPERANDS**

116481            The following operand shall be supported:

116482            *n*[[*sep*[+]*n*]...] A single command line argument that consists of one or more tab-stop values (*n*)  
 116483                                    separated by a separator character (*sep*) which is either a <comma> or a <blank>  
 116484                                    character. The application shall ensure that the tab-stop values are positive decimal  
 116485                                    integers in strictly ascending order. If any tab-stop value (except the first one) is  
 116486                                    preceded by a <plus-sign>, it is taken as an increment to be added to the previous  
 116487                                    value. For example, the tab lists 1,10,20,30 and "1 10 +10 +10" are considered  
 116488                                    to be identical.

116489 **STDIN**

116490            Not used.

116491 **INPUT FILES**

116492            None.

116493 **ENVIRONMENT VARIABLES**

116494            The following environment variables shall affect the execution of *tabs*:

116495            *LANG*            Provide a default value for the internationalization variables that are unset or null.  
 116496                                    (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 116497                                    variables used to determine the values of locale categories.)

116498            *LC\_ALL*         If set to a non-empty string value, override the values of all the other  
 116499                                    internationalization variables.

116500            *LC\_CTYPE*       Determine the locale for the interpretation of sequences of bytes of text data as  
 116501                                    characters (for example, single-byte as opposed to multi-byte characters in  
 116502                                    arguments).

116503            *LC\_MESSAGES*

116504                                    Determine the locale that should be used to affect the format and contents of  
 116505                                    diagnostic messages written to standard error.

116506 XSI        *NLSPATH*        Determine the location of messages objects and message catalogs.

116507            *TERM*            Determine the terminal type. If this variable is unset or null, and if the -T option is  
 116508                                    not specified, an unspecified default terminal type shall be used.

116509 **ASYNCHRONOUS EVENTS**

116510            Default.

116511 **STDOUT**

116512            If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be  
 116513                                    written to standard output in an unspecified format. If standard output is not a terminal,  
 116514                                    undefined results occur.

116515 **STDERR**

116516            The standard error shall be used only for diagnostic messages.

116517 **OUTPUT FILES**

116518            None.



116519 **EXTENDED DESCRIPTION**

116520 None.

116521 **EXIT STATUS**

116522 The following exit values shall be returned:

116523 0 Successful completion.

116524 &gt;0 An error occurred.

116525 **CONSEQUENCES OF ERRORS**

116526 Default.

116527 **APPLICATION USAGE**116528 This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

116529 This utility is not recommended for application use.

116530 Some integrated display units might not have escape sequences to set tab stops, but may be set  
116531 by internal system calls. On these terminals, *tabs* works if standard output is directed to the  
116532 terminal; if output is directed to another file, however, *tabs* fails.

116533 **EXAMPLES**

116534 None.

116535 **RATIONALE**

116536 Consideration was given to having the *tput* utility handle all of the functions described in *tabs*.  
116537 However, the separate *tabs* utility was retained because it seems more intuitive to use a  
116538 command named *tabs* than *tput* with a new option. The *tput* utility does not support setting or  
116539 clearing tabs, and no known historical version of *tabs* supports the capability of setting arbitrary  
116540 tab stops.

116541 The System V *tabs* interface is very complex; the version in this volume of POSIX.1-2024 has a  
116542 reduced feature list, but many of the features omitted were restored as part of the XSI option  
116543 even though the supported languages and coding styles are primarily historical.

116544 There was considerable sentiment for specifying only a means of resetting the tabs back to a  
116545 known state—presumably the “standard” of tabs every eight positions. The following features  
116546 were omitted:

- 116547 • Setting tab stops via the first line in a file, using *--file*. Since even the SVID has no  
116548 complete explanation of this feature, it is doubtful that it is in widespread use.

116549 In an early proposal, a *-t tablist* option was added for consistency with *expand*; this was later  
116550 removed when inconsistencies with the historical list of tabs were identified.

116551 Consideration was given to adding a *-p* option that would output the current tab settings so  
116552 that they could be saved and then later restored. This was not accepted because querying the tab  
116553 stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be  
116554 supported on a wide range of terminals.

116555 **FUTURE DIRECTIONS**

116556 None.

116557 **SEE ALSO**116558 *expand*, *stty*, *tput*, *unexpand*

116559 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

116560 **CHANGE HISTORY**

116561 First released in Issue 2.

116562 **Issue 6**

116563 This utility is marked as part of the User Portability Utilities option.

116564 The normative text is reworded to avoid use of the term ``must'' for application requirements.

116565 **Issue 7**

116566 The *tabs* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

116568 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

116569 The SYNOPSIS and OPERANDS sections are updated.

116570 **Issue 8**

116571 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

116572 **NAME**

116573 tail — copy the last part of a file

116574 **SYNOPSIS**116575 tail [-f] [-c *number*|-n *number*] [*file*]116576 tail -r [-n *number*] [*file*]116577 **DESCRIPTION**116578 The *tail* utility shall copy its input file to the standard output beginning at a designated place.116579 Copying shall begin at the point in the file indicated by the *-c number* or *-n number* options. The  
116580 option-argument *number* shall be counted in units of lines or bytes, according to the options *-n*  
116581 and *-c*. Both line and byte counts start from 1.116582 Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in  
116583 length. Such a buffer, if any, shall be no smaller than {LINE\_MAX}\*10 bytes.116584 **OPTIONS**116585 The *tail* utility shall conform to XBD [Section 12.2](#) (on page 215), except that '+' may be  
116586 recognized as an option delimiter as well as '-'.  
116587

116587 The following options shall be supported:

116588 *-c number* The application shall ensure that the *number* option-argument is a decimal integer,  
116589 optionally including a sign. The sign shall affect the location in the file, measured  
116590 in bytes, to begin the copying:

116591

116592

116593

116594

| Sign        | Copying Starts                         |
|-------------|----------------------------------------|
| +           | Relative to the beginning of the file. |
| -           | Relative to the end of the file.       |
| <i>none</i> | Relative to the end of the file.       |

116595 The application shall ensure that if the sign of the *number* option-argument is '+',  
116596 the *number* option-argument is a non-zero decimal integer.116597 The origin for counting shall be 1; that is, *-c +1* represents the first byte of the file,  
116598 *-c -1* the last.116599 *-f* If the input file is a regular file or if the *file* operand specifies a FIFO, do not  
116600 terminate after the last line of the input file has been copied, but read and copy  
116601 further bytes from the input file when they become available. If no *file* operand is  
116602 specified and standard input is a pipe or FIFO, the *-f* option shall be ignored. If the  
116603 input file is not a FIFO, pipe, or regular file, it is unspecified whether or not the *-f*  
116604 option shall be ignored.116605 *-n number* If *-r* is not specified, this option shall be equivalent to *-c number*, except the  
116606 starting location in the file shall be measured in lines instead of bytes. The origin  
116607 for counting shall be 1; that is, *-n +1* represents the first line of the file, *-n -1* the  
116608 last.116609 If *-r* is specified, *number* shall specify the number of lines to read (in reverse) from  
116610 the end of the input file. The application shall ensure that *number* does not have a  
116611 sign.116612 *-r* Copy the lines in reverse order (last line first). If *-n* is specified, that many lines of  
116613 the file, starting with the last line, shall be copied. If *-n* is not specified, every line  
116614 of the input file shall be copied.116615 If none of the *-c*, *-n* or *-r* options is specified, *-n 10* shall be assumed.

116616 **OPERANDS**

116617 The following operand shall be supported:

116618 *file* A pathname of an input file. If no *file* operand is specified, the standard input shall  
116619 be used.

116620 **STDIN**

116621 The standard input shall be used if no *file* operand is specified, and shall be used if the *file*  
116622 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
116623 the standard input shall not be used. See the INPUT FILES section.

116624 **INPUT FILES**

116625 If the -c option is specified, the input file can contain arbitrary data; otherwise, the input file  
116626 shall be a text file.

116627 **ENVIRONMENT VARIABLES**

116628 The following environment variables shall affect the execution of *tail*:

116629 *LANG* Provide a default value for the internationalization variables that are unset or null.  
116630 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
116631 variables used to determine the values of locale categories.)

116632 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
116633 internationalization variables.

116634 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
116635 characters (for example, single-byte as opposed to multi-byte characters in  
116636 arguments and input files).

116637 *LC\_MESSAGES*

116638 Determine the locale that should be used to affect the format and contents of  
116639 diagnostic messages written to standard error.

116640 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

116641 **ASYNCHRONOUS EVENTS**

116642 Default.

116643 **STDOUT**

116644 The designated portion of the input file shall be written to standard output.

116645 **STDERR**

116646 The standard error shall be used only for diagnostic messages.

116647 **OUTPUT FILES**

116648 None.

116649 **EXTENDED DESCRIPTION**

116650 None.

116651 **EXIT STATUS**

116652 The following exit values shall be returned:

116653 0 Successful completion.

116654 >0 An error occurred.

## 116655 CONSEQUENCES OF ERRORS

116656 Default.

## 116657 APPLICATION USAGE

116658 The `-c` option should be used with caution when the input is a text file containing multi-byte  
 116659 characters; it may produce output that does not start on a character boundary.

116660 Although the input file to *tail* can be any type, the results might not be what would be expected  
 116661 on some character special device files or on file types not described by the System Interfaces  
 116662 volume of POSIX.1-2024. Since this volume of POSIX.1-2024 does not specify the block size used  
 116663 when doing input, *tail* need not read all of the data from devices that only perform block  
 116664 transfers.

116665 When using *tail* to process pathnames, and the `-c` option is not specified, it is recommended that  
 116666 `LC_ALL`, or at least `LC_CTYPE` and `LC_COLLATE`, are set to `POSIX` or `C` in the environment,  
 116667 since pathnames can contain byte sequences that do not form valid characters in some locales, in  
 116668 which case the utility's behavior would be undefined. In the `POSIX` locale each byte is a valid  
 116669 single-byte character, and therefore this problem is avoided.

## 116670 EXAMPLES

116671 The `-f` option can be used to monitor the growth of a file that is being written by some other  
 116672 process. For example, the command:

```
116673 tail -f fred
```

116674 prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between  
 116675 the time *tail* is initiated and killed. As another example, the command:

```
116676 tail -f -c 15 fred
```

116677 prints the last 15 bytes of the file **fred**, followed by any bytes that are appended to **fred** between  
 116678 the time *tail* is initiated and killed.

## 116679 RATIONALE

116680 This version of *tail* was created to allow conformance to the Utility Syntax Guidelines. The  
 116681 historical `-b` option was omitted because of the general non-portability of block-sized units of  
 116682 text. The `-c` option historically meant “characters”, but this volume of POSIX.1-2024 indicates  
 116683 that it means “bytes”. This was selected to allow reasonable implementations when multi-byte  
 116684 characters are possible; it was not named `-b` to avoid confusion with the historical `-b`.

116685 The origin of counting both lines and bytes is 1, matching all widespread historical  
 116686 implementations. Hence *tail -n +0* is not conforming usage because it attempts to output line  
 116687 zero; but note that *tail -n 0* does conform, and outputs nothing.

116688 Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
116689 tail -[number] [b|c|l] [f] [file]
```

```
116690 tail +[number] [b|c|l] [f] [file]
```

116691 These forms are no longer specified by POSIX.1-2024, but may be present in some  
 116692 implementations.

116693 The restriction on the internal buffer is a compromise between the historical System V  
 116694 implementation of 4 096 bytes and the BSD 32 768 bytes.

116695 The `-f` option has been implemented as a loop that sleeps for 1 second and copies any bytes that  
 116696 are available. This is sufficient, but if more efficient methods of determining when new data are  
 116697 available are developed, implementations are encouraged to use them.

116698 Historical documentation indicates that *tail* ignores the `-f` option if the input file is a pipe (pipe

116699 and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System  
116700 V-based systems, this was true when input was taken from standard input, but it did not ignore  
116701 the `-f` flag if a FIFO was named as the *file* operand. Since the `-f` option is not useful on pipes and  
116702 all historical implementations ignore `-f` if no *file* operand is specified and standard input is a  
116703 pipe, this volume of POSIX.1-2024 requires this behavior. However, since the `-f` option is useful  
116704 on a FIFO, this volume of POSIX.1-2024 also requires that if a FIFO is named, the `-f` option shall  
116705 not be ignored. Earlier versions of this standard did not state any requirement for the case where  
116706 no *file* operand is specified and standard input is a FIFO. The standard has been updated to  
116707 reflect current practice which is to treat this case the same as a pipe on standard input. Although  
116708 historical behavior does not ignore the `-f` option for other file types, this is unspecified so that  
116709 implementations are allowed to ignore the `-f` option if it is known that the file cannot be  
116710 extended.

116711 The functionality made available by `tail -r` has been historically provided on some systems by a  
116712 separate utility (*tac*), although *tac* traditionally lacked support for `-n` to limit the output. While  
116713 both `tail -n$n | tac` and `tac | head -n$n` can be used to output a fixed length of  
116714 reversed line output, the standard developers decided that it was preferable to have a single  
116715 utility `tail -r -n$n` for the same purpose. Furthermore, in deciding whether to standardize  
116716 *tac* rather than `tail -r`, it was determined that more implementations that have achieved POSIX  
116717 certification had already implemented `tail -r` as an extension.

#### 116718 FUTURE DIRECTIONS

116719 None.

#### 116720 SEE ALSO

116721 [head](#)

116722 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 116723 CHANGE HISTORY

116724 First released in Issue 2.

#### 116725 Issue 6

116726 The obsolescent SYNOPSIS lines and associated text are removed.

116727 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 116728 Issue 7

116729 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized  
116730 as an option delimiter in the OPTIONS section.

116731 Austin Group Interpretation 1003.1-2001 #092 is applied.

116732 Austin Group Interpretation 1003.1-2001 #100 is applied, adding the requirement on applications  
116733 that if the sign of the option-argument *number* is '+', the *number* option-argument is a non-zero  
116734 decimal integer.

116735 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

116736 SD5-XCU-ERN-114 is applied, updating the OPTIONS section (the `-f` option).

116737 SD5-XCU-ERN-149 is applied.

116738 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0190 [663] is applied.

#### 116739 Issue 8

116740 Austin Group Defect 877 is applied, adding the `-r` option.

116741

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

116742 **NAME**

116743 talk — talk to another user

116744 **SYNOPSIS**116745 UP `talk address [terminal]`116746 **DESCRIPTION**116747 The *talk* utility is a two-way, screen-oriented communication program.116748 When first invoked, *talk* shall send a message similar to:

```
116749 Message from <unspecified string>
116750 talk: connection requested by your_address
116751 talk: respond with: talk your_address
```

116752 to the specified *address*. At this point, the recipient of the message can reply by typing:116753 `talk your_address`116754 Once communication is established, the two parties can type simultaneously, with their output  
116755 displayed in separate regions of the screen. Characters shall be processed as follows:

- 116756 • Typing the <alert> character shall alert the recipient's terminal.
- 116757 • Typing <control>-L shall cause the sender's screen regions to be refreshed.
- 116758 • Typing the erase and kill characters shall affect the sender's terminal in the manner  
116759 described by the **termios** interface in XBD [Chapter 11](#) (on page 199).
- 116760 • Typing the interrupt or end-of-file characters shall terminate the local *talk* utility. Once the  
116761 *talk* session has been terminated on one side, the other side of the *talk* session shall be  
116762 notified that the *talk* session has been terminated and shall be able to do nothing except  
116763 exit.
- 116764 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those  
116765 characters to be sent to the recipient's terminal.
- 116766 • When and only when the *stty iexten* local mode is enabled, the existence and processing of  
116767 additional special control characters and multi-byte or single-byte functions shall be  
116768 implementation-defined.
- 116769 • Typing other non-printable characters shall cause implementation-defined sequences of  
116770 printable characters to be sent to the recipient's terminal.

116771 Permission to be a recipient of a *talk* message can be denied or granted by use of the *mesg* utility.  
116772 However, a user's privilege may further constrain the domain of accessibility of other users'  
116773 terminals. The *talk* utility shall fail when the user lacks appropriate privileges to perform the  
116774 requested action.

116775 Certain block-mode terminals do not have all the capabilities necessary to support the  
116776 simultaneous exchange of messages required for *talk*. When this type of exchange cannot be  
116777 supported on such terminals, the implementation may support an exchange with reduced levels  
116778 of simultaneous interaction or it may report an error describing the terminal-related deficiency.

116779 **OPTIONS**

116780 None.



116781 **OPERANDS**

116782 The following operands shall be supported:

116783 *address* The recipient of the *talk* session. One form of *address* is the *<user name>*, as returned  
 116784 by the *who* utility. Other address formats and how they are handled are  
 116785 unspecified.

116786 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to  
 116787 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message  
 116788 shall be displayed on one or more accessible terminals in use by the recipient. The  
 116789 format of *terminal* shall be the same as that returned by the *who* utility.

116790 **STDIN**

116791 Characters read from standard input shall be copied to the recipient's terminal in an unspecified  
 116792 manner. If standard input is not a terminal, *talk* shall write a diagnostic message and exit with a  
 116793 non-zero status.

116794 **INPUT FILES**

116795 None.

116796 **ENVIRONMENT VARIABLES**

116797 The following environment variables shall affect the execution of *talk*:

116798 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 116799 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 116800 variables used to determine the values of locale categories.)

116801 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 116802 internationalization variables.

116803 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 116804 characters (for example, single-byte as opposed to multi-byte characters in  
 116805 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
 116806 equivalent to the sender's, the results are undefined.

116807 *LC\_MESSAGES*

116808 Determine the locale that should be used to affect the format and contents of  
 116809 diagnostic messages written to standard error and informative messages written to  
 116810 standard output.

116811 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

116812 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,  
 116813 an unspecified default terminal type shall be used.

116814 **ASYNCHRONOUS EVENTS**

116815 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero  
 116816 status. It shall take the standard action for all other signals.

116817 **STDOUT**

116818 If standard output is a terminal, characters copied from the recipient's standard input may be  
 116819 written to standard output. Standard output also may be used for diagnostic messages. If  
 116820 standard output is not a terminal, *talk* shall exit with a non-zero status.

116821 **STDERR**

116822 None.

116823 **OUTPUT FILES**

116824 None.

116825 **EXTENDED DESCRIPTION**

116826 None.

116827 **EXIT STATUS**

116828 The following exit values shall be returned:

116829 0 Successful completion.

116830 >0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.116831 **CONSEQUENCES OF ERRORS**

116832 Default.

116833 **APPLICATION USAGE**

116834 Because the handling of non-printable, non-`<space>` characters is tied to the *stty* description of  
116835 **iexten**, implementation extensions within the terminal driver can be accessed. For example,  
116836 some implementations provide line editing functions with certain control character sequences.

116837 **EXAMPLES**

116838 None.

116839 **RATIONALE**

116840 The *write* utility was included in this volume of POSIX.1-2024 since it can be implemented on all  
116841 terminal types. The *talk* utility, which cannot be implemented on certain terminals, was  
116842 considered to be a “better” communications interface. Both of these programs are in widespread  
116843 use on historical implementations. Therefore, both utilities have been specified.

116844 All references to networking abilities (*talking* to a user on another system) were removed as  
116845 being outside the scope of this volume of POSIX.1-2024.

116846 Historical BSD and System V versions of *talk* terminate both of the conversations when either  
116847 user breaks out of the session. This can lead to adverse consequences if a user unwittingly  
116848 continues to enter text that is interpreted by the shell when the other terminates the session.  
116849 Therefore, the version of *talk* specified by this volume of POSIX.1-2024 requires both users to  
116850 terminate their end of the session explicitly.

116851 Only messages sent to the terminal of the invoking user can be internationalized in any way:

- 116852 • The original “Message from `<unspecified string> ...`” message sent to the terminal of the  
116853 recipient cannot be internationalized because the environment of the recipient is as yet  
116854 inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.
- 116855 • Subsequent communication between the two parties cannot be internationalized because  
116856 the two parties may specify different languages in their environment (and non-portable  
116857 characters cannot be mapped from one language to another).
- 116858 • Neither party can be required to communicate in a language other than C and/or the one  
116859 specified by their environment because unavailable terminal hardware support (for  
116860 example, fonts) may be required.

116861 The text in the STDOUT section reflects the usage of the verb “display” in this section; some *talk*  
116862 implementations actually use standard output to write to the terminal, but this volume of  
116863 POSIX.1-2024 does not require that to be the case.

116864 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
116865 require that they all use or accept the same format.

116866 The handling of non-printable characters is partially implementation-defined because the details  
116867 of mapping them to printable sequences is not needed by the user. Historical implementations,  
116868 for security reasons, disallow the transmission of non-printable characters that may send  
116869 commands to the other terminal.

116870 **FUTURE DIRECTIONS**

116871 None.

116872 **SEE ALSO**

116873 *mesg, stty, who, write*

116874 XBD [Chapter 8](#) (on page 167), [Chapter 11](#) (on page 199)

116875 **CHANGE HISTORY**

116876 First released in Issue 4.

116877 **Issue 6**

116878 This utility is marked as part of the User Portability Utilities option.

116879 **Issue 8**

116880 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

116881 **NAME**

116882 tee — duplicate standard input

116883 **SYNOPSIS**116884 tee [-ai] [*file...*]116885 **DESCRIPTION**116886 The *tee* utility shall copy standard input to standard output, making a copy in zero or more files.116887 The *tee* utility shall not buffer output.116888 If the **-a** option is not specified, output files shall be written (see [Section 1.1.1.4](#), on page 2454).116889 **OPTIONS**116890 The *tee* utility shall conform to XBD [Section 12.2](#) (on page 215).

116891 The following options shall be supported:

116892 **-a** Append the output to the files.116893 **-i** Ignore the SIGINT signal.116894 **OPERANDS**

116895 The following operands shall be supported:

116896 *file* A pathname of an output file. If a *file* operand is '-', it shall refer to a file named  
 116897 -; implementations shall not treat it as meaning standard output. Processing of at  
 116898 least 13 *file* operands shall be supported.

116899 **STDIN**

116900 The standard input can be of any type.

116901 **INPUT FILES**

116902 None.

116903 **ENVIRONMENT VARIABLES**116904 The following environment variables shall affect the execution of *tee*:

116905 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 116906 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 116907 variables used to determine the values of locale categories.)

116908 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 116909 internationalization variables.

116910 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 116911 characters (for example, single-byte as opposed to multi-byte characters in  
 116912 arguments).

116913 *LC\_MESSAGES*

116914 Determine the locale that should be used to affect the format and contents of  
 116915 diagnostic messages written to standard error.

116916 *XSI NLSPATH* Determine the location of messages objects and message catalogs.

116917 **ASYNCHRONOUS EVENTS**116918 Default, except that if the **-i** option was specified, SIGINT shall be ignored.116919 **STDOUT**

116920 The standard output shall be a copy of the standard input.

**116921 STDERR**

116922 The standard error shall be used only for diagnostic messages.

**116923 OUTPUT FILES**

116924 If any *file* operands are specified, the standard input shall be copied to each named file.

**116925 EXTENDED DESCRIPTION**

116926 None.

**116927 EXIT STATUS**

116928 The following exit values shall be returned:

116929 0 The standard input was successfully copied to all output files.

116930 >0 An error occurred.

**116931 CONSEQUENCES OF ERRORS**

116932 If a write to any successfully opened *file* operand fails, writes to other successfully opened *file*  
116933 operands and standard output shall continue, but the exit status shall be non-zero. Otherwise,  
116934 the default actions specified in [Section 1.4](#) (on page 2462) apply.

**116935 APPLICATION USAGE**

116936 The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.

116937 The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.

**116938 EXAMPLES**

116939 Save an unsorted intermediate form of the data in a pipeline:

116940 `... | tee unsorted | sort > sorted`

**116941 RATIONALE**

116942 The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or  
116943 line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.

116944 It should be noted that early versions of BSD ignore any invalid options and accept a single '-'  
116945 as an alternative to -i. They also print a message if unable to open a file:

116946 `"tee: cannot access %s\n", <pathname>`

116947 Historical implementations ignore write errors. This is explicitly not permitted by this volume of  
116948 POSIX.1-2024.

116949 Some historical implementations use O\_APPEND when providing append mode; others use the  
116950 *lseek()* function to seek to the end-of-file after opening the file without O\_APPEND. This volume  
116951 of POSIX.1-2024 requires functionality equivalent to using O\_APPEND; see [Section 1.1.1.4](#) (on  
116952 page 2454).

**116953 FUTURE DIRECTIONS**

116954 If this utility is directed to create a new directory entry that contains any bytes that have the  
116955 encoded value of a <newline> character, implementations are encouraged to treat this as an  
116956 error. A future version of this standard may require implementations to treat this as an error.

**116957 SEE ALSO**

116958 [Chapter 1](#) (on page 2453), *cat*

116959 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

116960 XSH *lseek()*

116961 **CHANGE HISTORY**

116962 First released in Issue 2.

116963 **Issue 6**

116964 IEEE PASC Interpretation 1003.2 #168 is applied.

116965 **Issue 7**

116966 Austin Group Interpretation 1003.1-2001 #092 is applied.

116967 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

116968 **Issue 8**

116969 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a <newline> character.

116971 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

116972 Austin Group Defect 1494 is applied, inserting a missing closing parenthesis.

116973 **NAME**

116974 test — evaluate expression

116975 **SYNOPSIS**116976 test [*expression*]116977 [ [*expression*] ]116978 **DESCRIPTION**

116979 The *test* utility shall evaluate the *expression* and indicate the result of the evaluation by its exit  
 116980 status. An exit status of zero indicates that the expression evaluated as true and an exit status of  
 116981 1 indicates that the expression evaluated as false.

116982 In the second form of the utility, where the utility name used is *l* rather than *test*, the application  
 116983 shall ensure that the closing square bracket is a separate argument. The *test* and *l* utilities may be  
 116984 implemented as a single linked utility which examines the basename of the zeroth command  
 116985 line argument to determine whether to behave as the *test* or *l* variant. Applications using the *exec*  
 116986 family of functions to execute these utilities shall ensure that the argument passed in *argv* or  
 116987 *argv*[0] is '[' when executing the *l* utility and has a basename of "test" when executing the  
 116988 *test* utility.

116989 **OPTIONS**

116990 The *test* utility shall not recognize the "--" argument in the manner specified by Guideline 10 in  
 116991 XBD [Section 12.2](#) (on page 215). In addition, when the utility name used is *l* the utility does not  
 116992 conform to Guidelines 1 and 2.

116993 No options shall be supported.

116994 **OPERANDS**

116995 The application shall ensure that all operators and elements of primaries are presented as  
 116996 separate arguments to the *test* utility.

116997 The following primaries can be used to construct *expression*:

116998 **-b** *pathname* True if *pathname* resolves to an existing directory entry for a block special file. False  
 116999 if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry  
 117000 for a file that is not a block special file.

117001 **-c** *pathname* True if *pathname* resolves to an existing directory entry for a character special file.  
 117002 False if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory  
 117003 entry for a file that is not a character special file.

117004 **-d** *pathname* True if *pathname* resolves to an existing directory entry for a directory. False if  
 117005 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry  
 117006 for a file that is not a directory.

117007 **-e** *pathname* True if *pathname* resolves to an existing directory entry. False if *pathname* cannot be  
 117008 resolved.

117009 *pathname1* **-ef** *pathname2*

117010 True if *pathname1* and *pathname2* resolve to existing directory entries for the same  
 117011 file; otherwise, false.

117012 **-f** *pathname* True if *pathname* resolves to an existing directory entry for a regular file. False if  
 117013 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry  
 117014 for a file that is not a regular file.

117015 **-g** *pathname* True if *pathname* resolves to an existing directory entry for a file that has its set-  
 117016 group-ID flag set. False if *pathname* cannot be resolved, or if *pathname* resolves to an  
 117017 existing directory entry for a file that does not have its set-group-ID flag set.

- 117018        **-h** *pathname* True if *pathname* resolves to an existing directory entry for a symbolic link. False if  
 117019            *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry  
 117020            for a file that is not a symbolic link. If the final component of *pathname* is a  
 117021            symbolic link, that symbolic link is not followed.
- 117022        **-L** *pathname* True if *pathname* resolves to an existing directory entry for a symbolic link. False if  
 117023            *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry  
 117024            for a file that is not a symbolic link. If the final component of *pathname* is a  
 117025            symbolic link, that symbolic link is not followed.
- 117026        **-n** *string*     True if the length of *string* is non-zero; otherwise, false.
- 117027        *pathname1* **-nt** *pathname2*  
 117028            True if *pathname1* resolves to an existing file and *pathname2* cannot be resolved, or if  
 117029            both resolve to existing files and *pathname1* is newer than *pathname2* according to  
 117030            their last data modification timestamps; otherwise, false.
- 117031        *pathname1* **-ot** *pathname2*  
 117032            True if *pathname2* resolves to an existing file and *pathname1* cannot be resolved, or if  
 117033            both resolve to existing files and *pathname1* is older than *pathname2* according to  
 117034            their last data modification timestamps; otherwise, false.
- 117035        **-p** *pathname* True if *pathname* resolves to an existing directory entry for a FIFO. False if *pathname*  
 117036            cannot be resolved, or if *pathname* resolves to an existing directory entry for a file  
 117037            that is not a FIFO.
- 117038        **-r** *pathname* True if *pathname* resolves to an existing directory entry for a file for which  
 117039            permission to read from the file is granted, as defined in [Section 1.1.1.4](#) (on page  
 117040            2454). False if *pathname* cannot be resolved, or if *pathname* resolves to an existing  
 117041            directory entry for a file for which permission to read from the file is not granted.
- 117042        **-S** *pathname* True if *pathname* resolves to an existing directory entry for a socket. False if  
 117043            *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry  
 117044            for a file that is not a socket.
- 117045        **-s** *pathname* True if *pathname* resolves to an existing directory entry for a file that has a size  
 117046            greater than zero. False if *pathname* cannot be resolved, or if *pathname* resolves to an  
 117047            existing directory entry for a file that does not have a size greater than zero.
- 117048        **-t** *file\_descriptor*  
 117049            True if file descriptor number *file\_descriptor* is open and is associated with a  
 117050            terminal. False if *file\_descriptor* is not a valid file descriptor number, or if file  
 117051            descriptor number *file\_descriptor* is not open, or if it is open but is not associated  
 117052            with a terminal.
- 117053        **-u** *pathname* True if *pathname* resolves to an existing directory entry for a file that has its set-  
 117054            user-ID flag set. False if *pathname* cannot be resolved, or if *pathname* resolves to an  
 117055            existing directory entry for a file that does not have its set-user-ID flag set.
- 117056        **-w** *pathname* True if *pathname* resolves to an existing directory entry for a file for which  
 117057            permission to write to the file is granted, as defined in [Section 1.1.1.4](#) (on page  
 117058            2454). False if *pathname* cannot be resolved, or if *pathname* resolves to an existing  
 117059            directory entry for a file for which permission to write to the file is not granted.
- 117060        **-x** *pathname* True if *pathname* resolves to an existing directory entry for a file for which  
 117061            permission to execute the file (or search it, if it is a directory) is granted, as defined  
 117062            in [Section 1.1.1.4](#) (on page 2454). False if *pathname* cannot be resolved, or if  
 117063            *pathname* resolves to an existing directory entry for a file for which permission to



117064 execute (or search) the file is not granted.

117065 `-z string` True if the length of string *string* is zero; otherwise, false.

117066 `string` True if the string *string* is not the null string; otherwise, false.

117067 `s1 = s2` True if the strings *s1* and *s2* are identical; otherwise, false.

117068 `s1 != s2` True if the strings *s1* and *s2* are not identical; otherwise, false.

117069 `s1 > s2` True if *s1* collates after *s2* in the current locale; otherwise, false.

117070 `s1 < s2` True if *s1* collates before *s2* in the current locale; otherwise, false.

117071 `n1 -eq n2` True if the integers *n1* and *n2* are algebraically equal; otherwise, false.

117072 `n1 -ne n2` True if the integers *n1* and *n2* are not algebraically equal; otherwise, false.

117073 `n1 -gt n2` True if the integer *n1* is algebraically greater than the integer *n2*; otherwise, false.

117074 `n1 -ge n2` True if the integer *n1* is algebraically greater than or equal to the integer *n2*;  
117075 otherwise, false.

117076 `n1 -lt n2` True if the integer *n1* is algebraically less than the integer *n2*; otherwise, false.

117077 `n1 -le n2` True if the integer *n1* is algebraically less than or equal to the integer *n2*; otherwise,  
117078 false.

117079 With the exception of the `-h pathname` and `-L pathname` primaries, if a *pathname*, *pathname1*, or  
117080 *pathname2* argument is a symbolic link, *test* shall evaluate the expression by resolving the  
117081 symbolic link and using the file referenced by the link.

117082 These primaries can be combined with the following operator:

117083 `! expression` True if *expression* is false. False if *expression* is true.

117084 The primaries with two elements of the form:

117085 `-primary_operator primary_operand`

117086 are known as *unary primaries*. The primaries with three elements in either of the two forms:

117087 `primary_operand -primary_operator primary_operand`

117088 `primary_operand primary_operator primary_operand`

117089 are known as *binary primaries*. Additional implementation-defined operators and  
117090 *primary\_operators* may be provided by implementations. They shall be of the form `-operator`  
117091 where the first character of *operator* is not a digit.

117092 The algorithm for determining the precedence of the operators and the return value that shall be  
117093 generated is based on the number of arguments presented to *test*. (However, when using the  
117094 "[...]" form, the <right-square-bracket> final argument shall not be counted in this  
117095 algorithm.)

117096 In the following list, \$1, \$2, \$3, and \$4 represent the arguments presented to *test*:

117097 0 arguments: Exit false (1).

117098 1 argument: Exit true (0) if \$1 is not null; otherwise, exit false.

117099 2 arguments: • If \$1 is '!', exit true if \$2 is null, false if \$2 is not null.

- 117100                   • If \$1 is a unary primary, exit true if the unary test is true, false if the  
117101                   unary test is false.
- 117102                   • Otherwise, produce unspecified results.
- 117103           3 arguments:       • If \$2 is a binary primary, perform the binary test of \$1 and \$3.  
117104                   • If \$1 is '!', negate the two-argument test of \$2 and \$3.  
117105                   • Otherwise, produce unspecified results.
- 117106           4 arguments:       • If \$1 is '!', negate the three-argument test of \$2, \$3, and \$4.  
117107                   • Otherwise, the results are unspecified.
- 117108           >4 arguments:   The results are unspecified.
- 117109 **STDIN**
- 117110           Not used.
- 117111 **INPUT FILES**
- 117112           None.
- 117113 **ENVIRONMENT VARIABLES**
- 117114           The following environment variables shall affect the execution of *test*:
- 117115           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
117116                   (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
117117                   variables used to determine the values of locale categories.)
- 117118           *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
117119                   internationalization variables.
- 117120           *LC\_COLLATE*
- 117121                   Determine the locale for the behavior of the > and < string comparison operators.
- 117122           *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
117123                   characters (for example, single-byte as opposed to multi-byte characters in  
117124                   arguments).
- 117125           *LC\_MESSAGES*
- 117126                   Determine the locale that should be used to affect the format and contents of  
117127                   diagnostic messages written to standard error.
- 117128 *XSI*        *NLSPATH* Determine the location of messages objects and message catalogs.
- 117129 **ASYNCHRONOUS EVENTS**
- 117130           Default.
- 117131 **STDOUT**
- 117132           Not used.
- 117133 **STDERR**
- 117134           The standard error shall be used only for diagnostic messages.
- 117135 **OUTPUT FILES**
- 117136           None.
- 117137 **EXTENDED DESCRIPTION**
- 117138           None.

117139 **EXIT STATUS**

117140 The following exit values shall be returned:

- 117141 0 *expression* evaluated to true.
- 117142 1 *expression* evaluated to false or *expression* was missing.
- 117143 >1 An error occurred.

117144 **CONSEQUENCES OF ERRORS**

117145 Default.

117146 **APPLICATION USAGE**

117147 Since '>' and '<' are operators in the shell language, applications need to quote them when  
117148 passing them as arguments to *test* from a shell.

117149 The **-a** and **-o** binary primaries and the '(' and ')' operators have been removed. (Many  
117150 expressions using them were ambiguously defined by the grammar depending on the specific  
117151 expressions being evaluated.) Scripts using these expressions should be converted to the forms  
117152 given below. Even though many implementations will continue to support these forms, scripts  
117153 should be extremely careful when dealing with user-supplied input that could be confused with  
117154 these and other primaries and operators. Unless the application developer knows all the cases  
117155 that produce input to the script, invocations like:

117156 `test "$1" -a "$2"`

117157 should be written as:

117158 `test "$1" && test "$2"`

117159 to avoid problems if a user supplied values such as \$1 set to '!' and \$2 set to the null string.  
117160 That is, replace:

117161 `test expr1 -a expr2`

117162 with:

117163 `test expr1 && test expr2`

117164 and replace:

117165 `test expr1 -o expr2`

117166 with:

117167 `test expr1 || test expr2`

117168 but note that, in *test*, **-a** was specified as having higher precedence than **-o** while "&&" and  
117169 "||" have equal precedence in the shell.

117170 Parentheses or braces can be used in the shell command language to effect grouping.

117171 The two commands:

117172 `test "$1"`  
117173 `test ! "$1"`

117174 could not be used reliably on some historical systems. Unexpected results would occur if such a  
117175 *string* expression were used and \$1 expanded to '!', '(', or a known unary primary. Better  
117176 constructs are:

117177 `test -n "$1"`  
117178 `test -z "$1"`

117179 respectively.

117180 Historical systems have also been unreliable given the common construct:

```
117181 test "$response" = "expected string"
```

117182 One of the following is a more reliable form:

```
117183 test "X$response" = "Xexpected string"
```

```
117184 test "expected string" = "$response"
```

117185 Note that the second form assumes that *expected string* could not be confused with any unary  
117186 primary. If *expected string* starts with '-', '(', '!', or even '=', the first form should be used  
117187 instead. Using the preceding rules, any of the three comparison forms is reliable, given any  
117188 input. (However, note that the strings are quoted in all cases.)

117189 Historically, the string comparison binary primaries, '=' and '!=', had a higher precedence  
117190 than any unary primary in the greater than 4 argument case, and consequently unexpected  
117191 results could occur if arguments were not properly prepared. For example, in:

```
117192 test -d "$1" -o -d "$2"
```

117193 If \$1 evaluates to a possible directory name of '=', the first three arguments are considered a  
117194 string comparison, which causes a syntax error when the second -d is encountered. The  
117195 following form prevents this:

```
117196 test -d "$1" || test -d "$2"
```

117197 Also in the greater than 4 argument case:

```
117198 test "$1" = "bat" -a "$2" = "ball"
```

117199 syntax errors would occur if \$1 evaluates to ' (' or '!'. One of the following forms prevents  
117200 this; the second is preferred:

```
117201 test "$1" = "bat" && test "$2" = "ball"
```

```
117202 test "X$1" = "Xbat" && test "X$2" = "Xball"
```

117203 Note that none of the following examples are permitted by the syntax described:

```
117204 [-f file]
```

```
117205 [-f file ]
```

```
117206 [ -f file]
```

```
117207 [ -f file
```

```
117208 test -f file ]
```

117209 In the first two cases, if a utility named *[-f]* exists, that utility would be invoked, and not *test*. In  
117210 the remaining cases, the brackets are mismatched, and the behavior is unspecified. However:

```
117211 test ! ]
```

117212 does have a defined meaning, and must exit with status 1. Similarly:

```
117213 test ]
```

117214 must exit with status 0.

## 117215 EXAMPLES

117216 1. Exit if there are not two or three arguments (two variations):

```
117217 if [ $# -ne 2 ] && [ $# -ne 3 ]; then exit 1; fi
```

```
117218 if [ $# -lt 2 ] || [ $# -gt 3 ]; then exit 1; fi
```

```

117219      2. Perform a mkdir if a directory does not exist:
117220          test ! -d tempdir && mkdir tempdir
117221
117221      3. Wait for a file to become non-readable:
117222          while test -r thefile
117223          do
117224              sleep 30
117225          done
117226          echo "thefile" is no longer readable'
117227
117227      4. Perform a command if the argument is one of three strings (two variations):
117228          if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
117229          then
117230              command
117231          fi
117232
117232          case "$1" in
117233              pear|grape|apple) command ;;
117234          esac

```

#### 117235 RATIONALE

117236 The KornShell-derived conditional command (double bracket `[[ ]]`) was removed from the shell  
 117237 command language description in an early proposal. Objections were raised that the real  
 117238 problem is misuse of the *test* command (`!`), and putting it into the shell is the wrong way to fix  
 117239 the problem. Instead, proper documentation and a new shell reserved word (`!`) are sufficient. A  
 117240 later proposal to add `[[ ]]` in Issue 8 was also rejected because existing implementations of it were  
 117241 found to be error-prone in a similar way to historical versions of *test*, and there was also too  
 117242 much variation in behavior between shells that support it.

117243 Tests that require multiple *test* operations can be done at the shell level using individual  
 117244 invocations of the *test* command and shell logicals, rather than using the error-prone historical  
 117245 `-a` and `-o` operators of *test*.

117246 The BSD and System V versions of `-f` were not the same. The BSD definition was:

117247 `-f file` True if *file* exists and is not a directory.

117248 The SVID version (true if the file exists and is a regular file) was chosen for this volume of  
 117249 POSIX.1-2024 because its use is consistent with the `-b`, `-c`, `-d`, and `-p` operands (*file* exists and is  
 117250 a specific file type).

117251 The `-e` primary, possessing similar functionality to that provided by the C shell, was added  
 117252 because it provides the only way for a shell script to find out if a file exists without trying to  
 117253 open the file. Since implementations are allowed to add additional file types, a portable script  
 117254 cannot use:

```
117255 test -b foo || test -c foo || test -d foo || test -f foo || test -p foo
```

117256 to find out if **foo** is an existing file. On historical BSD systems, the existence of a file could be  
 117257 determined by:

```
117258 test -f foo || test -d foo
```

117259 but there was no easy way to determine that an existing file was a regular file. An early proposal  
 117260 used the KornShell `-a` primary (with the same meaning), but this was changed to `-e` because  
 117261 there were concerns about the high probability of humans confusing the `-a` primary with the  
 117262 historical `-a` binary operator.

117263 The following options were not included in this volume of POSIX.1-2024, although they are  
 117264 provided by some implementations. These operands should not be used by new  
 117265 implementations for other purposes:

117266 **-k file** True if *file* exists and its sticky bit is set.

117267 **-C file** True if *file* is a contiguous file.

117268 **-V file** True if *file* is a version file.

117269 The following option was not included because it was undocumented in most implementations,  
 117270 has been removed from some implementations (including System V), and the functionality is  
 117271 provided by the shell (see [Section 2.6.2](#) (on page 2485)).

117272 **-l string** The length of the string *string*.

117273 The **-b**, **-c**, **-g**, **-p**, **-u**, and **-x** operands are derived from the SVID; historical BSD does not  
 117274 provide them. The **-k** operand is derived from System V; historical BSD does not provide it.

117275 On historical BSD systems, *test -w directory* always returned false because *test* tried to open the  
 117276 directory for writing, which always fails.

117277 Some additional primaries newly invented or from the KornShell appeared in an early proposal  
 117278 as part of the conditional command (`([[ ]]: s1 > s2, s1 < s2, f1 -nt f2, f1 -ot f2, and f1 -ef f2`). They  
 117279 were not carried forward into the *test* utility when the conditional command was removed from  
 117280 the shell because they had not been included in the *test* utility built into historical  
 117281 implementations of the *sh* utility. However, they were later added to this standard once support  
 117282 for them became widespread.

117283 The **-t file\_descriptor** primary is shown with a mandatory argument because the grammar is  
 117284 ambiguous if it can be omitted. Historical implementations have allowed it to be omitted,  
 117285 providing a default of 1.

117286 It is noted that ' [ ' is not part of the portable filename character set; however, since it is required  
 117287 to be encoded by a single byte, and is part of the portable character set, the name of this utility  
 117288 forms a character string across all supported locales.

#### 117289 **FUTURE DIRECTIONS**

117290 None.

#### 117291 **SEE ALSO**

117292 [Section 1.1.1.4](#) (on page 2454), *find*

117293 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 117294 **CHANGE HISTORY**

117295 First released in Issue 2.

#### 117296 **Issue 5**

117297 The FUTURE DIRECTIONS section is added.

#### 117298 **Issue 6**

117299 The **-h** operand is added for symbolic links, and access permission requirements are clarified for  
 117300 the **-r**, **-w**, and **-x** operands to align with the IEEE P1003.2b draft standard.

117301 The normative text is reworded to avoid use of the term “must” for application requirements.

117302 The **-L** and **-S** operands are added for symbolic links and sockets.

117303 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/38 is applied, adding XSI margin  
 117304 marking and shading to a line in the OPERANDS section referring to the use of parentheses as

- 117305 arguments to the *test* utility.
- 117306 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/30 is applied, rewording the existence  
117307 primaries for the *test* utility.
- 117308 **Issue 7**
- 117309 Austin Group Interpretation 1003.1-2001 #107 is applied.
- 117310 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0143 [291] is applied.
- 117311 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0191 [898], XCU/TC2-2008/0192  
117312 [730], and XCU/TC2-2008/0193 [898] are applied.
- 117313 **Issue 8**
- 117314 Austin Group Defect 375 is applied, adding the *pathname1 -ef pathname2*,  
117315 *pathname1 -nt pathname2*, *pathname1 -ot pathname2*, *s1 > s2*, and *s1 < s2* primaries.
- 117316 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 117317 Austin Group Defect 1330 is applied, removing the obsolescent (and optional) *-a* and *-o* binary  
117318 primaries, and ' (' and ' ) ' operators.
- 117319 Austin Group Defect 1348 is applied, removing ``()'' from ``the *exec()* family of functions''.
- 117320 Austin Group Defect 1373 is applied, clarifying that when the utility name used is *[* the utility  
117321 does not conform to Guidelines 1 and 2.

117322 **NAME**

117323 time — time a simple command

117324 **SYNOPSIS**117325 time [-p] *utility* [*argument...*]117326 **DESCRIPTION**

117327 The *time* utility shall invoke the utility named by the *utility* operand with arguments supplied as  
 117328 the *argument* operands and write a message to standard error that lists timing statistics for the  
 117329 utility. The message shall include the following information:

- 117330 • The elapsed (real) time between invocation of *utility* and its termination.
- 117331 • The User CPU time, equivalent to the sum of the *tms\_utime* and *tms\_cutime* fields returned  
 117332 by the *times()* function defined in the System Interfaces volume of POSIX.1-2024 for the  
 117333 process in which *utility* is executed.
- 117334 • The System CPU time, equivalent to the sum of the *tms\_stime* and *tms\_cstime* fields  
 117335 returned by the *times()* function for the process in which *utility* is executed.

117336 The precision of the timing shall be no less than the granularity defined for the size of the clock  
 117337 tick unit on the system, but the results shall be reported in terms of standard time units (for  
 117338 example, 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

117339 When *time* is used in any of the following circumstances, via a simple command for which the  
 117340 word **time** is the command name (see [Section 2.9.1.1](#), on page 2500), and none of the characters  
 117341 in the word **time** is quoted, the results (including parsing of later words) are unspecified:

- 117342 • The simple command for which the word **time** is the command name includes one or more  
 117343 redirections (see [Section 2.7](#), on page 2493) or is (directly) part of a pipeline (see [Section](#)  
 117344 [2.9.2](#), on page 2504).
- 117345 • The next word that follows **time** would, if the word **time** were not present, be recognized  
 117346 as a reserved word (see [Section 2.4](#), on page 2478) or a control operator (see XBD [Section](#)  
 117347 [3.85](#), on page 44).

117348 Since these limitations only apply when *time* is executed via a simple command for which the  
 117349 word **time** is the command name and none of the characters in the word **time** is quoted, they  
 117350 can be avoided by quoting all or part of the word **time**, by arranging for the command name not  
 117351 to be **time** (for example, by having the command name be a word expansion), or by executing  
 117352 *time* via another utility such as *command* or *env*.

117353 The limitations on redirections and pipelines can also be overcome by embedding the simple  
 117354 command within a compound command—most commonly a grouping command (see [Section](#)  
 117355 [2.9.4.1](#), on page 2508)—and applying the redirections or piping to the compound command  
 117356 instead.

117357 Note that in no circumstances where the results are specified is it possible to apply different  
 117358 redirections to the *time* utility than are applied to the utility it invokes.

117359 The following examples (where *a* and *b* are assumed to be the names of utilities found by  
 117360 searching *PATH*) show unspecified usages:

```

117361 time a arg1 arg2 | b      # part of a pipeline
117362 a | time -p b            # part of a pipeline
117363 time a >/dev/null        # output redirection
117364 </dev/null time a        # input redirection
117365 time while anything...   # reserved word after time
117366 time ( cmd )             # control operator after time
  
```



```

117367     time;                # control operator after time
117368     time shift           # special built-in utility
117369     time -p cd /        # intrinsic utility

```

117370 The following examples have specified results and can be used as alternatives for the first four of  
 117371 the above when the *time* utility as specified here is intended to be invoked:

```

117372     { time a arg1 arg2; } | b
117373     t=time; a | $t -p b
117374     command time a >/dev/null
117375     </dev/null \time a

```

#### 117376 OPTIONS

117377 The *time* utility shall conform to XBD [Section 12.2](#) (on page 215).

117378 The following option shall be supported:

117379 **-p** Write the timing output to standard error in the format shown in the STDERR  
 117380 section.

#### 117381 OPERANDS

117382 The following operands shall be supported:

117383 *utility* The name of a utility that is to be invoked. If the *utility* operand names a special  
 117384 built-in utility (see [Section 2.15](#), on page 2526), an intrinsic utility (see [Section 1.7](#),  
 117385 on page 2470), or a function (see [Section 2.9.5](#), on page 2511), the results are  
 117386 unspecified.

117387 *argument* Any string to be supplied as an argument when invoking the utility named by the  
 117388 *utility* operand.

#### 117389 STDIN

117390 Not used.

#### 117391 INPUT FILES

117392 None.

#### 117393 ENVIRONMENT VARIABLES

117394 The following environment variables shall affect the execution of *time*:

117395 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 117396 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 117397 variables used to determine the values of locale categories.)

117398 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 117399 internationalization variables.

117400 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 117401 characters (for example, single-byte as opposed to multi-byte characters in  
 117402 arguments).

117403 *LC\_MESSAGES*

117404 Determine the locale that should be used to affect the format and contents of  
 117405 diagnostic and informative messages written to standard error.

117406 *LC\_NUMERIC*

117407 Determine the locale for numeric formatting.

- 117408 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 117409 **PATH** Determine the search path that shall be used to locate the utility to be invoked; see  
117410 XBD [Chapter 8](#) (on page 167).
- 117411 **ASYNCHRONOUS EVENTS**
- 117412 Default.
- 117413 **STDOUT**
- 117414 Not used.
- 117415 **STDERR**
- 117416 If the *utility* utility is invoked, the standard error shall be used to write the timing statistics and  
117417 may be used to write a diagnostic message if the utility terminates abnormally; otherwise, the  
117418 standard error shall be used to write diagnostic messages and may also be used to write the  
117419 timing statistics.
- 117420 If **-p** is specified, the following format shall be used for the timing statistics in the POSIX locale:
- 117421 "real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,  
117422 <system seconds>
- 117423 where each floating-point number shall be expressed in seconds. The precision used may be less  
117424 than the default six digits of %f, but shall be sufficiently precise to accommodate the size of the  
117425 clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits  
117426 shall follow the radix character). The number of digits following the radix character shall be no  
117427 less than one, even if this always results in a trailing zero. The implementation may append  
117428 white space and additional information following the format shown here. The implementation  
117429 may also prepend a single empty line before the format shown here.
- 117430 **OUTPUT FILES**
- 117431 None.
- 117432 **EXTENDED DESCRIPTION**
- 117433 None.
- 117434 **EXIT STATUS**
- 117435 If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise,  
117436 the *time* utility shall exit with one of the following values:
- 117437 1-125 An error occurred in the *time* utility.
- 117438 126 The utility specified by *utility* was found but could not be invoked.
- 117439 127 The utility specified by *utility* could not be found.
- 117440 **CONSEQUENCES OF ERRORS**
- 117441 Default.
- 117442 **APPLICATION USAGE**
- 117443 The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit  
117444 code 127 if a utility to be invoked cannot be found, so that applications can distinguish "failure  
117445 to find a utility" from "invoked utility exited with an error indication". The value 127 was  
117446 chosen because it is not commonly used for other meanings; most utilities use small values for  
117447 "normal error conditions" and the values above 128 can be confused with termination due to  
117448 receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could  
117449 be found, but not invoked. Some scripts produce meaningful error messages differentiating the  
117450 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice  
117451 that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any

117452 attempt to *exec* the utility fails for any other reason.

#### 117453 EXAMPLES

117454 It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by  
117455 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and  
117456 the *time* applies to everything in the file.

117457 Alternatively, the following command can be used to apply *time* to a complex command:

```
117458 time sh -c -- 'complex-command-line'
```

#### 117459 RATIONALE

117460 When the *time* utility was originally proposed to be included in the ISO POSIX-2:1993 standard,  
117461 questions were raised about its suitability for inclusion on the grounds that it was not useful for  
117462 conforming applications, specifically:

- 117463 • The underlying CPU definitions from the System Interfaces volume of POSIX.1-2024 are  
117464 vague, so the numeric output could not be compared accurately between systems or even  
117465 between invocations.
- 117466 • The creation of portable benchmark programs was outside the scope this volume of  
117467 POSIX.1-2024.

117468 However, *time* does fit in the scope of user portability. Human judgement can be applied to the  
117469 analysis of the output, and it could be very useful in hands-on debugging of applications or in  
117470 providing subjective measures of system performance. Hence it has been included in this  
117471 volume of POSIX.1-2024.

117472 The default output format has been left unspecified because historical implementations differ  
117473 greatly in their style of depicting this numeric output. The `-p` option was invented to provide  
117474 scripts with a common means of obtaining this information.

117475 In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather  
117476 than just a simple command. The POSIX definition has been worded to allow this  
117477 implementation. Consideration was given to invalidating this approach because of the historical  
117478 model from the C shell and System V shell. However, since the System V *time* utility historically  
117479 has not produced accurate results in pipeline timing (because the constituent processes are not  
117480 all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to  
117481 break historical KornShell usage.

117482 The term *utility* is used, rather than *command*, to highlight the fact that shell compound  
117483 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*  
117484 includes user application programs and shell scripts, not just the standard utilities.

#### 117485 FUTURE DIRECTIONS

117486 None.

#### 117487 SEE ALSO

117488 [Chapter 2](#) (on page 2472), *sh*

117489 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

117490 XSH *times*( )

#### 117491 CHANGE HISTORY

117492 First released in Issue 2.

117493 **Issue 6**

117494 This utility is marked as part of the User Portability Utilities option.

117495 **Issue 7**

117496 The *time* utility is moved from the User Portability Utilities option to the Base. User Portability  
117497 Utilities is now an option for interactive utilities.

117498 SD5-XCU-ERN-115 is applied, updating the example in the DESCRIPTION.

117499 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0144 [266] is applied.

117500 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0194 [723] is applied.

117501 **Issue 8**

117502 Austin Group Defect 267 is applied, allowing **time** to be a reserved word.

117503 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

117504 Austin Group Defect 1530 is applied, changing ``sh -c`` to ``sh -c --``.

117505 Austin Group Defect 1586 is applied, adding the *timeout* utility.

117506 Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

117507 **NAME**

117508            timeout — execute a utility with a time limit

117509 **SYNOPSIS**117510            timeout [-fp] [-k *time*] [-s *signal\_name*] *duration utility [argument...]*117511 **DESCRIPTION**

117512            The *timeout* utility shall execute the utility named by the *utility* operand, with arguments  
 117513            supplied as the *argument* operands (if any), in a child process. If the value of the *duration*  
 117514            operand is non-zero and the child process has not terminated after the specified time period,  
 117515            *timeout* shall send the signal specified by the *-s* option, or the SIGTERM signal if *-s* is not given.

117516            If the *-f* option is specified, the signal shall be sent only to the child process. Otherwise, it is  
 117517            implementation defined which one of the following methods is used to signal additional  
 117518            processes:

- 117519            • The *timeout* utility ensures it is a process group leader before creating the child process  
 117520            which executes the utility, in which case it shall send the signal to its process group.
- 117521            • The *timeout* utility arranges for any descendants of the child process that are orphaned to  
 117522            have their parent process changed to the *timeout* utility, in which case the signal shall be  
 117523            sent to the child process and all of its descendants.

117524            If the subsequent wait status of the child process shows that it was stopped by a signal, a  
 117525            SIGCONT signal shall also be sent in the same manner as the first signal; otherwise, a SIGCONT  
 117526            signal may be sent in the same manner.

117527            If the *-k* option is specified, and the child process created to execute the utility still has not  
 117528            terminated after the time period specified by the *time* option-argument has elapsed since the first  
 117529            signal was sent, *timeout* shall send a SIGKILL signal in the same manner as the first signal. If  
 117530            *timeout* receives a signal and propagates it to the child process (see ASYNCHRONOUS EVENTS  
 117531            below), this shall be treated as the first signal.

117532 **OPTIONS**117533            The *timeout* utility shall conform to XBD [Section 12.2](#) (on page 215).

117534            The following options shall be supported:

- 117535            **-f**            Only time out the utility itself, not its descendants.
- 117536            **-k *time***       Send a SIGKILL signal if the child process created to execute the utility has not  
 117537            terminated after the time period specified by *time* has elapsed since the first signal  
 117538            was sent. The value of *time* shall be interpreted as specified for the *duration*  
 117539            operand (see OPERANDS below).
- 117540            **-p**            Always preserve (mimic) the wait status of the executed utility, even if the time  
 117541            limit was reached.
- 117542            **-s *signal\_name***  
 117543            Specify the signal to send when the time limit is reached, using one of the symbolic  
 117544            names defined in the **<signal.h>** header. Values of *signal\_name* shall be recognized  
 117545            in a case-independent fashion, without the SIG prefix. By default, SIGTERM shall  
 117546            be sent.

117547 **OPERANDS**

117548            The following operands shall be supported:

- 117549            *duration*       The maximum amount of time to allow the utility to run, specified as a decimal  
 117550            number with an optional decimal fraction and an optional suffix, which can be:

|        |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 117551 |                              | <b>s</b> seconds                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117552 |                              | <b>m</b> minutes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117553 |                              | <b>h</b> hours                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 117554 |                              | <b>d</b> days                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 117555 |                              | If a decimal fraction is present, the application shall ensure that it is separated from the units by a <period>. If no suffix is present, the value shall specify seconds.                                                                                                                                                                                                                                                                                                                                                                                    |
| 117556 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117557 |                              | If the value is zero, <i>timeout</i> shall not enforce a time limit.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 117558 | <i>utility</i>               | The name of a utility that is to be executed. If the <i>utility</i> operand names any of the special built-in utilities in <a href="#">Section 2.15</a> (on page 2526), the results are undefined.                                                                                                                                                                                                                                                                                                                                                             |
| 117559 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117560 | <i>argument</i>              | Any string to be supplied as an argument when executing the utility named by the <i>utility</i> operand.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 117561 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117562 | <b>STDIN</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117563 |                              | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 117564 | <b>INPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117565 |                              | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 117566 | <b>ENVIRONMENT VARIABLES</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117567 |                              | The following environment variables shall affect the execution of <i>timeout</i> :                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 117568 | <i>LANG</i>                  | Provide a default value for the internationalization variables that are unset or null. (See XBD <a href="#">Section 8.2</a> (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)                                                                                                                                                                                                                                                                                                             |
| 117569 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117570 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117571 | <i>LC_ALL</i>                | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 117572 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117573 | <i>LC_CTYPE</i>              | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                                                                                                                                                                                                                                                                                      |
| 117574 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117575 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117576 | <i>LC_MESSAGES</i>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117577 |                              | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 117578 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117579 | XSI <i>NLSPATH</i>           | Determine the location of messages objects and message catalogs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117580 | <i>PATH</i>                  | Determine the search path that is used to locate the utility to be executed. See XBD <a href="#">Section 8.3</a> (on page 174).                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117581 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117582 | <b>ASYNCHRONOUS EVENTS</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117583 |                              | The default behavior specified in <a href="#">Section 1.4</a> (on page 2462) shall apply, except that:                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 117584 |                              | <ul style="list-style-type: none"> <li>• The <i>timeout</i> utility shall ignore SIGTTIN and SIGTTOU signals.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 117585 |                              | <ul style="list-style-type: none"> <li>• The <i>timeout</i> utility may alter the disposition of SIGALRM if the inherited disposition was for it to be ignored.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                     |
| 117586 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117587 |                              | <ul style="list-style-type: none"> <li>• If the signal specified with the <i>-s</i> option, or any signal whose default action is to terminate the process, is delivered to the <i>timeout</i> utility, then unless the signal is SIGKILL or SIGSTOP, the <i>timeout</i> utility shall immediately send the same signal to the process or processes to which it would send a signal when the time limit is reached. If the delivered signal is SIGALRM, <i>timeout</i> may behave as if the time limit had been reached instead of sending SIGALRM.</li> </ul> |
| 117588 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117589 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117590 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117591 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 117592 |                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

117593           • If the `-f` option is not specified, then if *timeout* sends a signal to its process group, it shall  
 117594           briefly change the disposition of that signal to ignored while it sends the signal, so that it  
 117595           does not receive the signal itself.

117596           With the single exception of the signal specified with the `-s` option, or SIGTERM if `-s` is not  
 117597           used, all signal dispositions inherited by the utility specified by the *utility* operand shall be the  
 117598           same as the disposition that *timeout* inherited.

#### 117599 **STDOUT**

117600           Not used.

#### 117601 **STDERR**

117602           The standard error shall be used only for diagnostic messages.

#### 117603 **OUTPUT FILES**

117604           None.

#### 117605 **EXTENDED DESCRIPTION**

117606           None.

#### 117607 **EXIT STATUS**

117608           If the `-p` option is not specified and the time limit was reached:

117609           • If the `-k` option was not specified or the utility terminated before the time period specified  
 117610           by the *time* option-argument elapsed since the first signal was sent, the exit status shall be  
 117611           124.

117612           • If the `-k` option was specified and the SIGKILL signal was sent, it is unspecified whether  
 117613           the exit status is 124 or the behavior is as if the `-p` option was specified.

117614           Otherwise, if the executed utility terminated by exiting, the exit status of *timeout* shall be that of  
 117615           the utility; if the utility was terminated by a signal, *timeout* shall terminate itself with the same  
 117616           signal while ensuring that a core image is not created.

117617           If an error occurs, the following exit values shall be returned:

117618           125 An error other than the two described below occurred.

117619           126 The utility specified by *utility* was found but could not be executed.

117620           127 The utility specified by *utility* could not be found.

#### 117621 **CONSEQUENCES OF ERRORS**

117622           Default.

#### 117623 **APPLICATION USAGE**

117624           Unlike the *kill* utility, the `-s` option of *timeout* is not required to accept the symbolic name 0 to  
 117625           represent signal value zero.

117626           When the value of *duration* is zero, *timeout* does not time out the utility, but it does still perform  
 117627           signal propagation (including to descendants of the utility if `-f` is not specified).

117628           Regardless of locale, the `<period>` character (the decimal-point character of the POSIX locale) is  
 117629           the decimal-point character recognized in the *duration* operand and the *time* option-argument.

117630           The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit  
 117631           code 127 if a utility to be invoked cannot be found, so that applications can distinguish “failure  
 117632           to find a utility” from “invoked utility exited with an error indication”. The value 127 was  
 117633           chosen because it is not commonly used for other meanings; most utilities use small values for  
 117634           “normal error conditions” and the values above 128 can be confused with termination due to  
 117635           receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could

117636 be found, but not invoked. Some scripts produce meaningful error messages differentiating the  
117637 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice  
117638 that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any  
117639 attempt to *exec* the utility fails for any other reason. The *timeout* utility extends these special exit  
117640 codes to 125 and 124, with the meanings described in EXIT STATUS. A *timeout* exit status below  
117641 124 can only result from passing through the exit status of the executed utility.

#### 117642 EXAMPLES

117643 None.

#### 117644 RATIONALE

117645 Some *timeout* implementations make themselves a process group leader (when *-f* is not used) in  
117646 order to be able to send signals to descendants of the child process. However, using this method  
117647 means that any descendants which change their process group do not receive the signal. To  
117648 ensure all descendants receive the signal, some implementations instead make use of a feature  
117649 whereby descendants that are orphaned have their parent process changed to the *timeout*  
117650 utility—that is, *timeout* becomes their “reaper”—together with the ability of a reaper to send a  
117651 signal to all of its descendants.

117652 Some historical *timeout* implementations exited with status *128+signal\_number* when the child  
117653 process was terminated by a signal before the time limit was reached (or when *-p* was used).  
117654 This is reasonable when *timeout* is invoked from a shell which sets  *\$?*  to *128+signal\_number*, but  
117655 not all shells do that. In particular, the KornShell sets  *\$?*  to *256+signal\_number* and so an exit  
117656 status of *128+signal\_number* from *timeout* would be misleading. In order to avoid any possible  
117657 ambiguity, this standard requires that *timeout* mimics the wait status of the child process by  
117658 terminating itself with the same signal. When it does this it needs to ensure that it does not  
117659 create a core image, otherwise it could overwrite one created by the invoked utility.

117660 The *timeout* utility ignores SIGTTIN and SIGTTOU so that if the utility it executes reads from or  
117661 writes to the controlling terminal and this generates a SIGTTIN or SIGTTOU for the process  
117662 group, *timeout* will not be stopped by the signal and can still time out the utility.

117663 Some historical *timeout* implementations always set the disposition for SIGTTIN and SIGTTOU  
117664 in the child process to default, even if these signals were inherited as ignored. This could result  
117665 in processes being stopped unexpectedly. Likewise, they did not ensure that for signals they  
117666 caught, the disposition inherited by the executed utility was the same as the disposition that was  
117667 inherited by *timeout*. This meant that, for example, if *timeout* was used in a script that was run  
117668 with *nohup*, the utility executed by *timeout* would unexpectedly not be protected from SIGHUP.  
117669 This standard requires that all signal dispositions inherited by the utility specified by the *utility*  
117670 operand are the same as the disposition that *timeout* inherited, with the single exception of the  
117671 signal that *timeout* sends when the time limit is reached, which needs to be inherited as default  
117672 in order for the timeout to take effect (without resorting to SIGKILL if *-k* is specified).

117673 Some historical *timeout* implementations only propagated a subset of the signals whose default  
117674 action is to terminate the process to the child process if one was delivered to the *timeout* utility.  
117675 Propagating these signals is beneficial, as otherwise termination of the *timeout* utility by a signal  
117676 results in the utility it executed being left running indefinitely (unless it also received the signal,  
117677 for example a terminal-generated SIGINT). There is no reason to select a subset of these signals  
117678 to be propagated, therefore this standard requires them all to be propagated (except SIGKILL,  
117679 which cannot). In the event that a user wants to prevent the utility being timed out, sending  
117680 *timeout* a SIGKILL can be used for this purpose.



117681 **FUTURE DIRECTIONS**

117682 None.

117683 **SEE ALSO**117684 *kill*117685 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215), [<signal.h>](#) (on page 346)117686 **CHANGE HISTORY**

117687 First released in Issue 8.

117688 **NAME**

117689 touch — change file access and modification times

117690 **SYNOPSIS**117691 touch [-acm] [-r *ref\_file*|-t *time*|-d *date\_time*] *file*...117692 **DESCRIPTION**117693 The *touch* utility shall change the last data modification timestamps, the last data access  
117694 timestamps, or both.117695 The time used can be specified by the **-t** *time* option-argument, the corresponding *time* fields of  
117696 the file referenced by the **-r** *ref\_file* option-argument, or the **-d** *date\_time* option-argument, as  
117697 specified in the following sections. If none of these are specified, *touch* shall use the current time.117698 For each *file* operand, *touch* shall perform actions equivalent to the following functions defined  
117699 in the System Interfaces volume of POSIX.1-2024:

- 117700 1. If *file* does not exist:
  - 117701 a. The *creat()* function is called with the following arguments:
    - 117702 — The *file* operand is used as the *path* argument.
    - 117703 — The value of the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, S\_IRGRP,  
117704 S\_IWGRP, S\_IROTH, and S\_IWOTH is used as the *mode* argument.
  - 117705 b. The *futimens()* function is called with the following arguments:
    - 117706 — The file descriptor opened in step 1a.
    - 117707 — The access time and the modification time, set as described in the OPTIONS  
117708 section, are used as the first and second elements of the *times* array argument,  
117709 respectively.
- 117710 2. If *file* exists, the *utimensat()* function is called with the following arguments:
  - 117711 a. The AT\_FDCWD special value is used as the *fd* argument.
  - 117712 b. The *file* operand is used as the *path* argument.
  - 117713 c. The access time and the modification time, set as described in the OPTIONS  
117714 section, are used as the first and second elements of the *times* array argument,  
117715 respectively.
  - 117716 d. The *flag* argument is set to zero.

117717 **OPTIONS**117718 The *touch* utility shall conform to XBD [Section 12.2](#) (on page 215).

117719 The following options shall be supported:

- 117720 **-a** Change the access time of *file*. Do not change the modification time unless **-m** is  
117721 also specified.
- 117722 **-c** Do not create a specified *file* if it does not exist. Do not write any diagnostic  
117723 messages concerning this condition.
- 117724 **-d** *date\_time* Use the specified *date\_time* instead of the current time. The option-argument shall  
117725 be a string of the form:  
117726 `YYYY-MM-DDThh:mm:ss[.frac][tz]`  
117727 or:

117728 `YYYY-MM-DDThh:mm:SS[,frac][tz]`

117729 where:

- 117730 • YYYY are at least four decimal digits giving the year.
- 117731 • MM, DD, hh, mm, and SS are as with `-t time`.
- 117732 • T is the time designator, and can be replaced by a single <space>.
- 117733 • [.frac] and [,frac] are either empty, or a <period> ('.') or a <comma>
- 117734 (' , ') respectively, followed by one or more decimal digits, specifying a
- 117735 fractional second.
- 117736 • [tz] is either empty, signifying local time, or the letter 'Z', signifying UTC.
- 117737 If [tz] is empty, the resulting time shall be affected by the value of the TZ
- 117738 environment variable.

117739 If the resulting time precedes the Epoch, the behavior is implementation-defined. If

117740 the time cannot be represented as the file's timestamp, *touch* shall exit immediately

117741 with an error status.

117742 **-m** Change the modification time of *file*. Do not change the access time unless **-a** is

117743 also specified.

117744 **-r ref\_file** Use the corresponding time of the file named by the pathname *ref\_file* instead of

117745 the current time.

117746 **-t time** Use the specified *time* instead of the current time. The option-argument shall be a

117747 decimal number of the form:

117748 `[[CC]YY]MMDDhhmm[.SS]`

117749 where each two digits represents the following:

117750 *MM* The month of the year [01,12].

117751 *DD* The day of the month [01,31].

117752 *hh* The hour of the day [00,23].

117753 *mm* The minute of the hour [00,59].

117754 *CC* The first two digits of the year (the century).

117755 *YY* The second two digits of the year.

117756 *SS* The second of the minute [00,60].

117757 Both *CC* and *YY* shall be optional. If neither is given, the current year shall be

117758 assumed. If *YY* is specified, but *CC* is not, *CC* shall be derived as follows:

| If YY is: | CC becomes: |
|-----------|-------------|
| [69,99]   | 19          |
| [00,68]   | 20          |

117762 **Note:** It is expected that in a future version of this standard the default century inferred

117763 from a 2-digit year will change. (This would apply to all commands accepting a

117764 2-digit year as input.)

117765 The resulting time shall be affected by the value of the *TZ* environment variable. If

117766 the resulting time value precedes the Epoch, the behavior is implementation-

117767 defined. If the time is out of range for the file's timestamp, *touch* shall exit

- 117768 immediately with an error status. The range of valid times past the Epoch is  
 117769 implementation-defined, but it shall extend to at least the time 0 hours, 0 minutes,  
 117770 0 seconds, January 1, 2038, Coordinated Universal Time. Some implementations  
 117771 may not be able to represent dates beyond January 18, 2038, because they use  
 117772 **signed int** as a time holder.
- 117773 The range for *SS* is [00,60] rather than [00,59] because of leap seconds. If *SS* is 60,  
 117774 and the resulting time, as affected by the *TZ* environment variable, does not refer  
 117775 to a leap second, the resulting time shall be one second after a time where *SS* is 59.  
 117776 If *SS* is not given a value, it is assumed to be zero.
- 117777 If neither the **-a** nor **-m** options were specified, *touch* shall behave as if both the **-a** and **-m**  
 117778 options were specified.
- 117779 **OPERANDS**
- 117780 The following operands shall be supported:
- 117781 *file* A pathname of a file whose times shall be modified.
- 117782 **STDIN**
- 117783 Not used.
- 117784 **INPUT FILES**
- 117785 None.
- 117786 **ENVIRONMENT VARIABLES**
- 117787 The following environment variables shall affect the execution of *touch*:
- 117788 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 117789 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 117790 variables used to determine the values of locale categories.)
- 117791 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 117792 internationalization variables.
- 117793 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 117794 characters (for example, single-byte as opposed to multi-byte characters in  
 117795 arguments).
- 117796 *LC\_MESSAGES*
- 117797 Determine the locale that should be used to affect the format and contents of  
 117798 diagnostic messages written to standard error.
- 117799 *XSI* *NLSPATH* Determine the location of messages objects and message catalogs.
- 117800 *TZ* Determine the timezone to be used for interpreting the *time* option-argument. If *TZ*  
 117801 is unset or null, an unspecified default timezone shall be used.
- 117802 **ASYNCHRONOUS EVENTS**
- 117803 Default.
- 117804 **STDOUT**
- 117805 Not used.
- 117806 **STDERR**
- 117807 The standard error shall be used only for diagnostic messages.

117808 **OUTPUT FILES**

117809 None.

117810 **EXTENDED DESCRIPTION**

117811 None.

117812 **EXIT STATUS**

117813 The following exit values shall be returned:

117814 0 The utility executed successfully and all requested changes were made.

117815 &gt;0 An error occurred.

117816 **CONSEQUENCES OF ERRORS**

117817 Default.

117818 **APPLICATION USAGE**

117819 The interpretation of time is taken to be *seconds since the Epoch* (see XBD Section 4.19, on page  
 117820 107). It should be noted that implementations conforming to the System Interfaces volume of  
 117821 POSIX.1-2024 do not take leap seconds into account when computing seconds since the Epoch.  
 117822 When *SS=60* is used, the resulting time always refers to 1 plus *seconds since the Epoch* for a time  
 117823 when *SS=59*.

117824 Although the *-t time* option-argument specifies values in 1969, the access time and modification  
 117825 time fields are defined in terms of seconds since the Epoch (00:00:00 on 1 January 1970 UTC).  
 117826 Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid  
 117827 hours in 1969 and there need not be any valid times in 1969.

117828 If the *T* time designator is replaced by a <space> for the *-d date\_time* option-argument, the  
 117829 <space> must be quoted to prevent the shell from splitting the argument.

117830 **EXAMPLES**

117831 Create or update a file called **dwc**; the resulting file has both the last data modification and last  
 117832 data access timestamps set to November 12, 2007 at 10:15:30 local time:

117833 `touch -d 2007-11-12T10:15:30 dwc`

117834 Create or update a file called **nick**; the resulting file has both the last data modification and last  
 117835 data access timestamps set to November 12, 2007 at 10:15:30 UTC:

117836 `touch -d 2007-11-12T10:15:30Z nick`

117837 Create or update a file called **gwc**; the resulting file has both the last data modification and last  
 117838 data access timestamps set to November 12, 2007 at 10:15:30 local time with a fractional second  
 117839 timestamp of .002 seconds:

117840 `touch -d 2007-11-12T10:15:30,002 gwc`

117841 Create or update a file called **ajosey**; the resulting file has both the last data modification and  
 117842 last data access timestamps set to November 12, 2007 at 10:15:30 UTC with a fractional second  
 117843 timestamp of .002 seconds:

117844 `touch -d "2007-11-12 10:15:30.002Z" ajosey`

117845 Create or update a file called **cathy**; the resulting file has both the last data modification and last  
 117846 data access timestamps set to November 12, 2007 at 10:15:00 local time:

117847 `touch -t 200711121015 cathy`

117848 Create or update a file called **drepper**; the resulting file has both the last data modification and  
 117849 last data access timestamps set to November 12, 2007 at 10:15:30 local time:

117850 touch -t 200711121015.30 drepper

117851 Create or update a file called **ebb9**; the resulting file has both the last data modification and last  
117852 data access timestamps set to November 12, 2007 at 10:15:30 local time:

117853 touch -t 0711121015.30 ebb9

117854 Create or update a file called **eggert**; the resulting file has the last data access timestamp set to  
117855 the corresponding time of the file named **mark** instead of the current time. If the file exists, the  
117856 last data modification time is not changed:

117857 touch -a -r mark eggert

#### 117858 RATIONALE

117859 The functionality of *touch* is described almost entirely through references to functions in the  
117860 System Interfaces volume of POSIX.1-2024. In this way, there is no duplication of effort required  
117861 for describing such side-effects as the relationship of user IDs to the user database, permissions,  
117862 and so on.

117863 There are some significant differences between the *touch* utility in this volume of POSIX.1-2024  
117864 and those in System V and BSD systems. They are upwards-compatible for historical  
117865 applications from both implementations:

- 117866 1. In System V, an ambiguity exists when a pathname that is a decimal number leads the  
117867 operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be  
117868 *touched* to the current time. The **-t** *time* construct solves these problems for future  
117869 conforming applications (note that the **-t** option is not historical practice).
- 117870 2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is  
117871 treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates  
117872 following the Epoch was included as recognition that some implementations are not able  
117873 to represent dates beyond 18 January 2038 because they use **signed int** as a time holder.

117874 The **-r** option was added because several comments requested this capability. This option was  
117875 named **-f** in an early proposal, but was changed because the **-f** option is used in the BSD  
117876 version of *touch* with a different meaning.

117877 At least one historical implementation of *touch* incremented the exit code if **-c** was specified and  
117878 the file did not exist. This volume of POSIX.1-2024 requires exit status zero if no errors occur.

117879 In previous version of the standard, if at least two operands are specified, and the first operand  
117880 is an eight or ten-digit decimal integer, the first operand was assumed to be a *date\_time* operand.  
117881 This usage was removed in this version of the standard since it had been marked obsolescent  
117882 previously.

117883 The **-d** *date\_time* format is an ISO 8601-1:2019 standard complete representation of date and time  
117884 extended format with an optional decimal point or <comma> followed by a string of digits  
117885 following the seconds portion to specify fractions of a second. It is not necessary to recognize  
117886 "[+/-]hh:mm" and "[+/-]hh" to specify timezones other than local time and UTC. The *T*  
117887 time designator in the ISO 8601-1:2019 standard extended format may be replaced by <space>.

#### 117888 FUTURE DIRECTIONS

117889 If this utility is directed to create a new directory entry that contains any bytes that have the  
117890 encoded value of a <newline> character, implementations are encouraged to treat this as an  
117891 error. A future version of this standard may require implementations to treat this as an error.

117892 **SEE ALSO**117893 *date*117894 XBD [Section 4.19](#) (on page 107), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215), [<sys/stat.h>](#)117895 XSH *creat()*, *futimens()*, *time()*117896 **CHANGE HISTORY**

117897 First released in Issue 2.

117898 **Issue 6**117899 The obsolescent *date\_time* operand is removed.

117900 The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal Time. This is a new requirement on POSIX implementations.

117903 The range for seconds is changed from [00,61] to [00,60] to align with the ISO/IEC 9899:1999 standard, and to allow for positive leap seconds.

117905 **Issue 7**

117906 Austin Group Interpretation 1003.1-2001 #118 is applied.

117907 Austin Group Interpretation 1003.1-2001 #193 is applied, adding support for subsecond timestamps.

117909 SD5-XCU-ERN-45 is applied, adding a new paragraph to the RATIONALE.

117910 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

117911 SD5-XCU-ERN-110 is applied, updating the OPTIONS section.

117912 Changes are made related to support for finegrained timestamps.

117913 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0195 [474] is applied.

117914 **Issue 8**

117915 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of filenames containing any bytes that have the encoded value of a &lt;newline&gt; character.

117917 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

117918 **NAME**

117919 tput — change terminal characteristics

117920 **SYNOPSIS**117921 tput [-T *type*] *operand*...117922 **DESCRIPTION**

117923 The *tput* utility shall display terminal-dependent information. The manner in which this  
 117924 information is retrieved is unspecified. The information displayed shall clear the terminal  
 117925 screen, initialize the user's terminal, or reset the user's terminal, depending on the operand  
 117926 given. The exact consequences of displaying this information are unspecified.

117927 **OPTIONS**117928 The *tput* utility shall conform to XBD [Section 12.2](#) (on page 215).

117929 The following option shall be supported:

117930 **-T *type*** Indicate the type of terminal. If this option is not supplied and the *TERM* variable  
 117931 is unset or null, an unspecified default terminal type shall be used. The setting of  
 117932 *type* shall take precedence over the value in *TERM*.

117933 **OPERANDS**

117934 The following strings shall be supported as operands by the implementation in the POSIX locale:

117935 **clear** Display the clear-screen sequence.

117936 **init** Display the sequence that initializes the user's terminal in an implementation-  
 117937 defined manner.

117938 **reset** Display the sequence that resets the user's terminal in an implementation-defined  
 117939 manner.

117940 If a terminal does not support any of the operations described by these operands, this shall not  
 117941 be considered an error condition.

117942 **STDIN**

117943 Not used.

117944 **INPUT FILES**

117945 None.

117946 **ENVIRONMENT VARIABLES**117947 The following environment variables shall affect the execution of *tput*:

117948 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 117949 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 117950 variables used to determine the values of locale categories.)

117951 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 117952 internationalization variables.

117953 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 117954 characters (for example, single-byte as opposed to multi-byte characters in  
 117955 arguments).

117956 **LC\_MESSAGES**

117957 Determine the locale that should be used to affect the format and contents of  
 117958 diagnostic messages written to standard error.



|        |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------|-----|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 117959 | XSI | <b>NLSPATH</b>                | Determine the location of messages objects and message catalogs.                                                                                                                                                                                                                                                                                                                                              |
| 117960 |     | <b>TERM</b>                   | Determine the terminal type. If this variable is unset or null, and if the <code>-T</code> option is not specified, an unspecified default terminal type shall be used.                                                                                                                                                                                                                                       |
| 117961 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117962 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117963 |     |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                      |
| 117964 |     | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117965 |     |                               | If standard output is a terminal device, it may be used for writing the appropriate sequence to clear the screen or reset or initialize the terminal. If standard output is not a terminal device, undefined results occur.                                                                                                                                                                                   |
| 117966 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117967 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117968 |     | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117969 |     |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                                                |
| 117970 |     | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117971 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                         |
| 117972 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117973 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                         |
| 117974 |     | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117975 |     |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                  |
| 117976 |     | 0                             | The requested string was written successfully.                                                                                                                                                                                                                                                                                                                                                                |
| 117977 |     | 1                             | Unspecified.                                                                                                                                                                                                                                                                                                                                                                                                  |
| 117978 |     | 2                             | Usage error.                                                                                                                                                                                                                                                                                                                                                                                                  |
| 117979 |     | 3                             | No information is available about the specified terminal type.                                                                                                                                                                                                                                                                                                                                                |
| 117980 |     | 4                             | The specified operand is invalid.                                                                                                                                                                                                                                                                                                                                                                             |
| 117981 |     | >4                            | An error occurred.                                                                                                                                                                                                                                                                                                                                                                                            |
| 117982 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117983 |     |                               | If one of the operands is not available for the terminal, <i>tput</i> continues processing the remaining operands.                                                                                                                                                                                                                                                                                            |
| 117984 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117985 |     | <b>APPLICATION USAGE</b>      |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117986 |     |                               | The difference between resetting and initializing a terminal is left unspecified, as they vary greatly based on hardware types. In general, resetting is a more severe action.                                                                                                                                                                                                                                |
| 117987 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117988 |     |                               | Some terminals use control characters to perform the stated functions, and on such terminals it might make sense to use <i>tput</i> to store the initialization strings in a file or environment variable for later use. However, because other terminals might rely on system calls to do this work, the standard output cannot be used in a portable manner, such as the following non-portable constructs: |
| 117989 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117990 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117991 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117992 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117993 |     |                               | <code>ClearVar=`tput clear`</code>                                                                                                                                                                                                                                                                                                                                                                            |
| 117994 |     |                               | <code>tput reset   mailx -s "Wake Up" ddg</code>                                                                                                                                                                                                                                                                                                                                                              |
| 117995 |     | <b>EXAMPLES</b>               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117996 |     | 1.                            | Initialize the terminal according to the type of terminal in the environmental variable <i>TERM</i> . This command can be included in a <b>profile</b> file.                                                                                                                                                                                                                                                  |
| 117997 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                               |
| 117998 |     |                               | <code>tput init</code>                                                                                                                                                                                                                                                                                                                                                                                        |

117999           2. Reset a 450 terminal.  
118000           tput -T 450 reset

#### 118001 RATIONALE

118002           The list of operands was reduced to a minimum for the following reasons:

- 118003           • The only features chosen were those that were likely to be used by human users interacting  
118004           with a terminal.
- 118005           • Specifying the full *terminfo* set was not considered desirable, but the standard developers  
118006           did not want to select among operands.
- 118007           • This volume of POSIX.1-2024 does not attempt to provide applications with sophisticated  
118008           terminal handling capabilities, as that falls outside of its assigned scope and intersects with  
118009           the responsibilities of other standards bodies.

118010           The difference between resetting and initializing a terminal is left unspecified as this varies  
118011           greatly based on hardware types. In general, resetting is a more severe action.

118012           The exit status of 1 is historically reserved for finding out if a Boolean operand is not set.  
118013           Although the operands were reduced to a minimum, the exit status of 1 should still be reserved  
118014           for the Boolean operands, for those sites that wish to support them.

#### 118015 FUTURE DIRECTIONS

118016           None.

#### 118017 SEE ALSO

118018           [stty](#), [tabs](#)

118019           XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 118020 CHANGE HISTORY

118021           First released in Issue 4.

#### 118022 Issue 6

118023           This utility is marked as part of the User Portability Utilities option.

#### 118024 Issue 7

118025           The *tput* utility is moved from the User Portability Utilities option to the Base. User Portability  
118026           Utilities is now an option for interactive utilities.

#### 118027 Issue 8

118028           Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

118029 **NAME**

118030 tr — translate characters

118031 **SYNOPSIS**118032 tr [-c|-C] [-s] *string1 string2*118033 tr -s [-c|-C] *string1*118034 tr -d [-c|-C] *string1*118035 tr -ds [-c|-C] *string1 string2*118036 **DESCRIPTION**

118037 The *tr* utility shall copy the standard input to the standard output with substitution or deletion  
 118038 of selected characters. The options specified and the *string1* and *string2* operands shall control  
 118039 translations that occur while copying characters and single-character collating elements.

118040 **OPTIONS**118041 The *tr* utility shall conform to XBD [Section 12.2](#) (on page 215).

118042 The following options shall be supported:

118043 **-c** Complement the set of values specified by *string1*. See the EXTENDED  
 118044 DESCRIPTION section.

118045 **-C** Complement the set of characters specified by *string1*. See the EXTENDED  
 118046 DESCRIPTION section.

118047 **-d** Delete all occurrences of input characters that are specified by *string1*.

118048 **-s** Replace instances of repeated characters with a single character, as described in the  
 118049 EXTENDED DESCRIPTION section.

118050 **OPERANDS**

118051 The following operands shall be supported:

118052 *string1, string2*

118053 Translation control strings. Each string shall represent a set of characters to be  
 118054 converted into an array of characters used for the translation. For a detailed  
 118055 description of how the strings are interpreted, see the EXTENDED DESCRIPTION  
 118056 section.

118057 **STDIN**

118058 The standard input can be any type of file.

118059 **INPUT FILES**

118060 None.

118061 **ENVIRONMENT VARIABLES**118062 The following environment variables shall affect the execution of *tr*:

118063 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 118064 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 118065 variables used to determine the values of locale categories.)

118066 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 118067 internationalization variables.

118068 **LC\_COLLATE**

118069 Determine the locale for the behavior of range expressions and equivalence classes.

- 118070 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behavior of character classes.
- 118071
- 118072
- 118073 **LC\_MESSAGES**
- 118074 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 118075
- 118076 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 118077 **ASYNCHRONOUS EVENTS**
- 118078 Default.
- 118079 **STDOUT**
- 118080 The *tr* output shall be identical to the input, with the exception of the specified transformations.
- 118081 **STDERR**
- 118082 The standard error shall be used only for diagnostic messages.
- 118083 **OUTPUT FILES**
- 118084 None.
- 118085 **EXTENDED DESCRIPTION**
- 118086 The operands *string1* and *string2* (if specified) define two arrays of characters. The constructs in the following list can be used to specify characters or single-character collating elements. If any of the constructs result in multi-character collating elements, *tr* shall exclude, without a diagnostic, those multi-character elements from the resulting array.
- 118087
- 118088
- 118089
- 118090 *character* Any character not described by one of the conventions below shall represent itself.
- 118091 *\octal* Octal sequences can be used to represent characters with specific coded values. An octal sequence shall consist of a <backslash> followed by the longest sequence of one, two, or three-octal-digit characters (01234567). The sequence shall cause the value whose encoding is represented by the one, two, or three-digit octal integer to be placed into the array. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
- 118092
- 118093
- 118094
- 118095
- 118096
- 118097 *\character* The <backslash>-escape sequences in XBD [Table 5-1](#) (on page 113) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be supported. The results of using any other character, other than an octal digit, following the <backslash> are unspecified. Also, if there is no character following the <backslash>, the results are unspecified.
- 118098
- 118099
- 118100
- 118101
- 118102 *c-c* In the POSIX locale, this construct shall represent the range of collating elements between the range endpoints (as long as neither endpoint is an octal sequence of the form *\octal*), inclusive, as defined by the collation sequence. The characters or collating elements in the range shall be placed in the array in ascending collation sequence. If the second endpoint precedes the starting endpoint in the collation sequence, it is unspecified whether the range of collating elements is empty, or this construct is treated as invalid. In locales other than the POSIX locale, this construct has unspecified behavior.
- 118103
- 118104
- 118105
- 118106
- 118107
- 118108
- 118109
- 118110 If either or both of the range endpoints are octal sequences of the form *\octal*, this shall represent the range of specific coded values between the two range endpoints, inclusive.
- 118111
- 118112

|        |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 118113 | <code>[:class:]</code> | Represents all characters belonging to the defined character class, as defined by the current setting of the <code>LC_CTYPE</code> locale category. The following character class names shall be accepted when specified in <code>string1</code> :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 118114 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118115 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118116 |                        | <b>alnum</b> <b>blank</b> <b>digit</b> <b>lower</b> <b>punct</b> <b>upper</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 118117 |                        | <b>alpha</b> <b>cntrl</b> <b>graph</b> <b>print</b> <b>space</b> <b>xdigit</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 118118 | XSI                    | In addition, character class expressions of the form <code>[:name:]</code> shall be recognized in those locales where the <code>name</code> keyword has been given a <b>charclass</b> definition in the <code>LC_CTYPE</code> category.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 118119 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118120 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118121 |                        | When both the <code>-d</code> and <code>-s</code> options are specified, any of the character class names shall be accepted in <code>string2</code> . Otherwise, only character class names <b>lower</b> or <b>upper</b> are valid in <code>string2</code> and then only if the corresponding character class ( <b>upper</b> and <b>lower</b> , respectively) is specified in the same relative position in <code>string1</code> . Such a specification shall be interpreted as a request for case conversion. When <code>[:lower:]</code> appears in <code>string1</code> and <code>[:upper:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the <b>toupper</b> mapping in the <code>LC_CTYPE</code> category of the current locale. When <code>[:upper:]</code> appears in <code>string1</code> and <code>[:lower:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the <b>tolower</b> mapping in the <code>LC_CTYPE</code> category of the current locale. The first character from each mapping pair shall be in the array for <code>string1</code> and the second character from each mapping pair shall be in the array for <code>string2</code> in the same relative position. |
| 118122 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118123 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118124 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118125 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118126 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118127 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118128 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118129 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118130 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118131 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118132 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118133 |                        | Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 118134 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118135 |                        | If the name specified for <code>class</code> does not define a valid character class in the current locale, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 118136 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118137 | <code>[=equiv=]</code> | Represents all characters or collating elements belonging to the same equivalence class as <code>equiv</code> , as defined by the current setting of the <code>LC_COLLATE</code> locale category. An equivalence class expression shall be allowed only in <code>string1</code> , or in <code>string2</code> when it is being used by the combined <code>-d</code> and <code>-s</code> options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 118138 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118139 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118140 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118141 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118142 | <code>[x*n]</code>     | Represents <code>n</code> repeated occurrences of the character <code>x</code> . Because this expression is used to map multiple characters to one, it is only valid when it occurs in <code>string2</code> . If <code>n</code> is omitted or is zero, it shall be interpreted as large enough to extend the <code>string2</code> -based sequence to the length of the <code>string1</code> -based sequence. If <code>n</code> has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 118143 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118144 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118145 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118146 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118147 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118148 |                        | When the <code>-d</code> option is not specified:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 118149 |                        | • If <code>string2</code> is present, each input character found in the array specified by <code>string1</code> shall be replaced by the character in the same relative position in the array specified by <code>string2</code> . If the array specified by <code>string2</code> is shorter than the one specified by <code>string1</code> , or if a character occurs more than once in <code>string1</code> , the results are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 118150 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118151 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118152 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118153 |                        | • If the <code>-C</code> option is specified, the complements of the characters specified by <code>string1</code> (the set of all characters in the current character set, as defined by the current setting of <code>LC_CTYPE</code> , except for those actually specified in the <code>string1</code> operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of <code>LC_COLLATE</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 118154 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118155 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 118156 |                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

118157 • If the `-c` option is specified, the complement of the values specified by *string1* shall be  
118158 placed in the array in ascending order by binary value.

118159 • Because the order in which characters specified by character class expressions or  
118160 equivalence class expressions is undefined, such expressions should only be used if the  
118161 intent is to map several characters into one. An exception is case conversion, as described  
118162 previously.

118163 When the `-d` option is specified:

118164 • Input characters found in the array specified by *string1* shall be deleted.

118165 • When the `-C` option is specified with `-d`, all characters except those specified by *string1*  
118166 shall be deleted. The contents of *string2* are ignored, unless the `-s` option is also specified.

118167 • When the `-c` option is specified with `-d`, all values except those specified by *string1* shall  
118168 be deleted. The contents of *string2* shall be ignored, unless the `-s` option is also specified.

118169 • The same string cannot be used for both the `-d` and the `-s` option; when both options are  
118170 specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be  
118171 required.

118172 When the `-s` option is specified, after any deletions or translations have taken place, repeated  
118173 sequences of the same character shall be replaced by one occurrence of the same character, if the  
118174 character is found in the array specified by the last operand. If the last operand contains a  
118175 character class, such as the following example:

```
118176 tr -s '[:space:]'
```

118177 the last operand's array shall contain all of the characters in that character class. However, in a  
118178 case conversion, as described previously, such as:

```
118179 tr -s '[:upper:]' '[:lower:]'
```

118180 the last operand's array shall contain only those characters defined as the second characters in  
118181 each of the **toupper** or **tolower** character pairs, as appropriate.

118182 An empty string used for *string1* or *string2* produces undefined results.

### 118183 EXIT STATUS

118184 The following exit values shall be returned:

118185 0 All input was processed successfully.

118186 >0 An error occurred.

### 118187 CONSEQUENCES OF ERRORS

118188 Default.

### 118189 APPLICATION USAGE

118190 If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

118191 If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must  
118192 use the full three digits to avoid ambiguity.

118193 When *string2* is shorter than *string1*, a difference results between historical System V and BSD  
118194 systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible  
118195 to do the following:

```
118196 tr 0123456789 d
```

118197 which would translate all digits to the letter 'd'. Since this area is specifically unspecified in

118198 this volume of POSIX.1-2024, both the BSD and System V behaviors are allowed, but a  
 118199 conforming application cannot rely on the BSD behavior. It would have to code the example in  
 118200 the following way:

```
118201 tr 0123456789 '[d*]'
```

118202 It should be noted that, despite similarities in appearance, the string operands used by *tr* are not  
 118203 regular expressions.

118204 Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL  
 118205 characters in its input stream. NUL characters can be stripped by using:

```
118206 tr -d '\000'
```

#### 118207 EXAMPLES

118208 1. The following example creates a list of all words in **file1** one per line in **file2**, where a  
 118209 word is taken to be a maximal string of letters.

```
118210 tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

118211 2. The next example translates all lowercase characters in **file1** to uppercase and writes the  
 118212 results to standard output.

```
118213 tr "[:lower:]" "[:upper:]" <file1
```

118214 3. This example uses an equivalence class to identify accented variants of the base character  
 118215 'e' in **file1**, which are stripped of diacritical marks and written to **file2**.

```
118216 tr "[=e]" "[e*]" <file1 >file2
```

#### 118217 RATIONALE

118218 In some early proposals, an explicit option `-n` was added to disable the historical behavior of  
 118219 stripping NUL characters from the input. It was considered that automatically stripping NUL  
 118220 characters from the input was not correct functionality. However, the removal of `-n` in a later  
 118221 proposal does not remove the requirement that *tr* correctly process NUL characters in its input  
 118222 stream. NUL characters can be stripped by using `tr -d '\000'`.

118223 Historical implementations of *tr* differ widely in syntax and behavior. For example, the BSD  
 118224 version has not needed the bracket characters for the repetition sequence. The *tr* utility syntax is  
 118225 based more closely on the System V and XPG3 model while attempting to accommodate  
 118226 historical BSD implementations. In the case of the short *string2* padding, the decision was to  
 118227 unspecified the behavior and preserve System V and XPG3 scripts, which might find difficulty  
 118228 with the BSD method. The assumption was made that BSD users of *tr* have to make  
 118229 accommodations to meet the syntax defined here. Since it is possible to use the repetition  
 118230 sequence to duplicate the desired behavior, whereas there is no simple way to achieve the  
 118231 System V method, this was the correct, if not desirable, approach.

118232 The use of octal values to specify control characters, while having historical precedents, is not  
 118233 portable. The introduction of escape sequences for control characters should provide the  
 118234 necessary portability. It is recognized that this may cause some historical scripts to break.

118235 An early proposal included support for multi-character collating elements. It was pointed out  
 118236 that, while *tr* does employ some syntactical elements from REs, the aim of *tr* is quite different;  
 118237 ranges, for example, do not have a similar meaning (`any of the chars in the range matches`,  
 118238 *versus* `translate each character in the range to the output counterpart`). As a result, the  
 118239 previously included support for multi-character collating elements has been removed. What  
 118240 remains are ranges in current collation order (to support, for example, accented characters),  
 118241 character classes, and equivalence classes.

118242 In XPG3 the `[class:]` and `[equiv=]` conventions are shown with double brackets, as in RE syntax.  
 118243 However, *tr* does not implement RE principles; it just borrows part of the syntax. Consequently,  
 118244 `[class:]` and `[equiv=]` should be regarded as syntactical elements on a par with `[x*n]`, which is  
 118245 not an RE bracket expression.

118246 The standard developers will consider changes to *tr* that allow it to translate characters between  
 118247 different character encodings, or they will consider providing a new utility to accomplish this.

118248 On historical System V systems, a range expression requires enclosing square-brackets, such as:

```
118249 tr '[a-z]' '[A-Z]'
```

118250 However, BSD-based systems did not require the brackets, and this convention is used here to  
 118251 avoid breaking large numbers of BSD scripts:

```
118252 tr a-z A-Z
```

118253 The preceding System V script will continue to work because the brackets, treated as regular  
 118254 characters, are translated to themselves. However, any System V script that relied on "a-z"  
 118255 representing the three characters 'a', '-', and 'z' have to be rewritten as "az-".

118256 The ISO POSIX-2:1993 standard had a `-c` option that behaved similarly to the `-C` option, but did  
 118257 not supply functionality equivalent to the `-c` option specified in POSIX.1-2024.

118258 The earlier version also said that octal sequences referred to collating elements and could be  
 118259 placed adjacent to each other to specify multi-byte characters. However, it was noted that this  
 118260 caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were  
 118261 intending to specify multi-byte characters or multiple single byte characters. POSIX.1-2024  
 118262 specifies that octal sequences always refer to single byte binary values when used to specify an  
 118263 endpoint of a range of collating elements.

118264 Earlier versions of this standard allowed for implementations with bytes other than eight bits,  
 118265 but this has been modified in this version.

#### 118266 FUTURE DIRECTIONS

118267 None.

#### 118268 SEE ALSO

118269 [\*sed\*](#)

118270 [XBD Table 5-1](#) (on page 113), [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 118271 CHANGE HISTORY

118272 First released in Issue 2.

#### 118273 Issue 6

118274 The `-C` operand is added, and the description of the `-c` operand is changed to align with the  
 118275 IEEE P1003.2b draft standard.

118276 The normative text is reworded to avoid use of the term “must” for application requirements.

118277 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/31 is applied, removing text describing  
 118278 behavior on systems with bytes consisting of more than eight bits.

118279 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/32 is applied, updating an example in the  
 118280 EXAMPLES section to avoid using unspecified behavior.

118281 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/33 is applied, making a correction to the  
 118282 RATIONALE.



118283 **Issue 7**

118284 SD5-XCU-ERN-30 is applied.

118285 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

118286 Austin Group Interpretation 1003.1-2001 #132 is applied, adding rationale to the `\character`  
118287 construct.

118288 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0145 [325] is applied.

118289 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0196 [663] is applied.

118290 **Issue 8**

118291 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.

118292 **NAME**

118293 true — return true value

118294 **SYNOPSIS**

118295 true

118296 **DESCRIPTION**

118297 The *true* utility shall return with exit code zero.

118298 **OPTIONS**

118299 None.

118300 **OPERANDS**

118301 None.

118302 **STDIN**

118303 Not used.

118304 **INPUT FILES**

118305 None.

118306 **ENVIRONMENT VARIABLES**

118307 None.

118308 **ASYNCHRONOUS EVENTS**

118309 Default.

118310 **STDOUT**

118311 Not used.

118312 **STDERR**

118313 Not used.

118314 **OUTPUT FILES**

118315 None.

118316 **EXTENDED DESCRIPTION**

118317 None.

118318 **EXIT STATUS**

118319 Zero.

118320 **CONSEQUENCES OF ERRORS**

118321 None.

118322 **APPLICATION USAGE**

118323 This utility is typically used in shell scripts, as shown in the **EXAMPLES** section.

118324 Although the special built-in utility `:` (*colon*) is similar to *true*, there are some notable differences,  
118325 including:

- 118326 • Whereas *colon* is required to accept, and do nothing with, any number of arguments, *true* is  
118327 only required to accept, and discard, a first argument of "--". Passing any other  
118328 argument(s) to *true* may cause its behavior to differ from that described in this standard.
- 118329 • A non-interactive shell exits when a redirection error occurs with *colon* (unless executed via  
118330 *command*), whereas with *true* it does not.
- 118331 • Variable assignments preceding the command name persist after executing *colon* (unless  
118332 executed via *command*), but not after executing *true*.

- 118333                   • In shell implementations where *true* is not provided as a built-in, using *colon* avoids the  
118334                   overheads associated with executing an external utility.

**118335 EXAMPLES**

118336                   This command is executed forever:

```
118337                   while true  
118338                   do  
118339                    command  
118340                   done
```

**118341 RATIONALE**

118342                   None.

**118343 FUTURE DIRECTIONS**

118344                   None.

**118345 SEE ALSO**

118346                   [Section 2.9](#) (on page 2499), *colon*, *command*, *false*

**118347 CHANGE HISTORY**

118348                   First released in Issue 2.

**118349 Issue 6**

118350                   IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/39 is applied, replacing the terms “None”  
118351                   and “Default” from the STDERR and EXIT STATUS sections, respectively, with terms as defined  
118352                   in [Section 1.4](#) (on page 2462).

**118353 Issue 8**

118354                   Austin Group Defect 1640 is applied, clarifying the differences between *true* and *:* (*colon*).

118355 **NAME**

118356 tsort — topological sort

118357 **SYNOPSIS**118358 tsort [-w] [*file*]118359 **DESCRIPTION**118360 The *tsort* utility shall write to standard output a totally ordered list of items consistent with a  
118361 partial ordering of items contained in the input.118362 The application shall ensure that the input consists of pairs of items (non-empty strings)  
118363 separated by one or more <blank> or <newline> characters. It is unspecified whether other  
118364 white-space characters can also be used as separators. Pairs of different items shall indicate  
118365 ordering. Pairs of identical items shall indicate presence, but not ordering.118366 If a cycle is found in the input, diagnostic or warning messages shall be written to standard error  
118367 reporting that there is a cycle and indicating which nodes are in the cycle(s). If the *-w* option is  
118368 specified, these messages shall be diagnostic messages. If a diagnostic message is written, the  
118369 final exit status shall be non-zero.118370 **OPTIONS**118371 The *tsort* utility shall conform to XBD [Section 12.2](#) (on page 215).

118372 The following option shall be supported:

118373 *-w* Set the exit status to the number of cycles found in the input, or to an  
118374 implementation-defined maximum if there are more cycles than that maximum. If  
118375 no cycles are found, the exit status shall be zero unless another error occurs.118376 **OPERANDS**

118377 The following operand shall be supported:

118378 *file* A pathname of a text file to order. If no *file* operand is given, the standard input  
118379 shall be used.118380 **STDIN**118381 The standard input shall be used if no *file* operand is specified, and shall be used if the *file*  
118382 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
118383 the standard input shall not be used. See the INPUT FILES section.118384 **INPUT FILES**

118385 The input file shall be a text file.

118386 **ENVIRONMENT VARIABLES**118387 The following environment variables shall affect the execution of *tsort*:118388 *LANG* Provide a default value for the internationalization variables that are unset or null.  
118389 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
118390 variables used to determine the values of locale categories.)118391 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
118392 internationalization variables.118393 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
118394 characters (for example, single-byte as opposed to multi-byte characters in  
118395 arguments and input files).118396 *LC\_MESSAGES*118397 Determine the locale that should be used to affect the format and contents of  
118398 diagnostic messages written to standard error.

118399 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

118400 **ASYNCHRONOUS EVENTS**

118401 Default.

118402 **STDOUT**

118403 The standard output shall be a text file consisting of the ordered list of items, with one item per  
118404 line, produced from the partially ordered input.

118405 **STDERR**

118406 The standard error shall be used only for diagnostic and warning messages.

118407 **OUTPUT FILES**

118408 None.

118409 **EXTENDED DESCRIPTION**

118410 None.

118411 **EXIT STATUS**

118412 The following exit values shall be returned:

118413 0 Successful completion.

118414 >0 An error occurred. If the *-w* option is specified and one or more cycles were found in the  
118415 input, the exit status shall be the number of cycles found, or an implementation-defined  
118416 maximum if more cycles than that maximum were found.

118417 **CONSEQUENCES OF ERRORS**

118418 Default.

118419 **APPLICATION USAGE**

118420 The *LC\_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not  
118421 lexicographic, but depends on the pairs of items given as input.

118422 **EXAMPLES**

118423 The command:

118424 `tsort <<EOF`  
118425 `a b c c d e`  
118426 `g g`  
118427 `f g e f`  
118428 `h h`  
118429 `EOF`

118430 produces the output:

118431 **a**  
118432 **b**  
118433 **c**  
118434 **d**  
118435 **e**  
118436 **f**  
118437 **g**  
118438 **h**

**118439 RATIONALE**

118440 At the time that the `-w` option was added to this standard, the only known implementation  
118441 reported a maximum of 255 cycles via the exit status. This has the drawback that applications  
118442 cannot distinguish, from the exit status, errors caused by cycles from other errors or (when *tsort*  
118443 is executed from a shell) termination by a signal. Implementations are urged to set the  
118444 implementation-defined maximum number of cycles reported via the exit status to at most 124,  
118445 leaving values above that maximum through 125 for other errors, and leaving values 126 and  
118446 greater to have the special meanings that the shell assigns to them.

**118447 FUTURE DIRECTIONS**

118448 A future version of this standard may require that when the `-w` option is specified, the  
118449 maximum number of cycles reported through the exit status of *tsort* is at most 124 and that exit  
118450 status values greater than 126 are not used by *tsort*.

**118451 SEE ALSO**

118452 XBD [Chapter 8](#) (on page 167)

**118453 CHANGE HISTORY**

118454 First released in Issue 2.

**118455 Issue 6**

118456 The normative text is reworded to avoid use of the term “must” for application requirements.

**118457 Issue 7**

118458 Austin Group Interpretation 1003.1-2001 #092 is applied.

118459 The *tsort* utility is moved from the XSI option to the Base.

118460 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0146 [241] is applied.

**118461 Issue 8**

118462 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

118463 Austin Group Defects 1617 and 1629 are applied, clarifying how *tsort* handles cycles found in the  
118464 input.

118465 Austin Group Defect 1745 is applied, clarifying the input separator characters and the output  
118466 format.

118467 **NAME**118468 `tty` — return user's terminal name118469 **SYNOPSIS**118470 `tty`118471 **DESCRIPTION**

118472 The `tty` utility shall write to the standard output the name of the terminal that is open as  
 118473 standard input. The name that is used shall be equivalent to the string that would be returned by  
 118474 the `ttyname()` function defined in the System Interfaces volume of POSIX.1-2024.

118475 **OPTIONS**118476 The `tty` utility shall conform to XBD [Section 12.2](#) (on page 215).118477 **OPERANDS**

118478 None.

118479 **STDIN**

118480 While no input is read from standard input, standard input shall be examined to determine  
 118481 whether or not it is a terminal, and, if so, to determine the name of the terminal.

118482 **INPUT FILES**

118483 None.

118484 **ENVIRONMENT VARIABLES**118485 The following environment variables shall affect the execution of `tty`:

118486 `LANG` Provide a default value for the internationalization variables that are unset or null.  
 118487 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 118488 variables used to determine the values of locale categories.)

118489 `LC_ALL` If set to a non-empty string value, override the values of all the other  
 118490 internationalization variables.

118491 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
 118492 characters (for example, single-byte as opposed to multi-byte characters in  
 118493 arguments).

118494 `LC_MESSAGES`

118495 Determine the locale that should be used to affect the format and contents of  
 118496 diagnostic messages written to standard error and informative messages written to  
 118497 standard output.

118498 `XSI` `NLSPATH` Determine the location of messages objects and message catalogs.

118499 **ASYNCHRONOUS EVENTS**

118500 Default.

118501 **STDOUT**

118502 If standard input is a terminal device, a pathname of the terminal as specified by the `ttyname()`  
 118503 function defined in the System Interfaces volume of POSIX.1-2024 shall be written in the  
 118504 following format:

118505 `"%s\n", <terminal name>`

118506 Otherwise, a message shall be written indicating that standard input is not connected to a  
 118507 terminal. In the POSIX locale, the `tty` utility shall use the format:

118508 `"not a tty\n"`

**118509 STDERR**

118510 The standard error shall be used only for diagnostic messages.

**118511 OUTPUT FILES**

118512 None.

**118513 EXTENDED DESCRIPTION**

118514 None.

**118515 EXIT STATUS**

118516 The following exit values shall be returned:

118517 0 Standard input is a terminal, and the output specified in STDOUT was successfully written  
118518 to standard output.

118519 1 Standard input is not a terminal, and the output specified in STDOUT was successfully  
118520 written to standard output.

118521 >1 An error occurred.

**118522 CONSEQUENCES OF ERRORS**

118523 Default.

**118524 APPLICATION USAGE**

118525 This utility checks the status of the file open as standard input against that of an  
118526 implementation-defined set of files. It is possible that no match can be found, or that the match  
118527 found need not be the same file as that which was opened for standard input (although they are  
118528 the same device).

**118529 EXAMPLES**

118530 None.

**118531 RATIONALE**

118532 None.

**118533 FUTURE DIRECTIONS**

118534 None.

**118535 SEE ALSO**

118536 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

118537 XSH [isatty\(\)](#), [ttyname\(\)](#)

**118538 CHANGE HISTORY**

118539 First released in Issue 2.

**118540 Issue 5**

118541 The SYNOPSIS is changed to indicate two forms of the command, with the second form marked  
118542 as obsolete. This is a clarification and does not change the functionality published in previous  
118543 issues.

**118544 Issue 6**

118545 The obsolescent `-s` option is removed.

**118546 Issue 8**

118547 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.

118548 Austin Group Defect 1509 is applied, changing the EXIT STATUS section.



118549 **NAME**

118550            type — write a description of command type

118551 **SYNOPSIS**118552 XSI        type *name*...118553 **DESCRIPTION**118554            The *type* utility shall indicate how each argument would be interpreted if used as a command  
118555            name.118556 **OPTIONS**

118557            None.

118558 **OPERANDS**

118559            The following operand shall be supported:

118560            *name*            A name to be interpreted.118561 **STDIN**

118562            Not used.

118563 **INPUT FILES**

118564            None.

118565 **ENVIRONMENT VARIABLES**118566            The following environment variables shall affect the execution of *type*:118567            *LANG*            Provide a default value for the internationalization variables that are unset or null.  
118568                                (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
118569                                variables used to determine the values of locale categories.)118570            *LC\_ALL*        If set to a non-empty string value, override the values of all the other  
118571                                internationalization variables.118572            *LC\_CTYPE*    Determine the locale for the interpretation of sequences of bytes of text data as  
118573                                characters (for example, single-byte as opposed to multi-byte characters in  
118574                                arguments).118575            *LC\_MESSAGES*118576                                Determine the locale that should be used to affect the format and contents of  
118577                                diagnostic messages written to standard error.118578            *NLSPATH*     Determine the location of messages objects and message catalogs.118579            *PATH*         Determine the location of *name*, as described in XBD [Chapter 8](#) (on page 167).118580 **ASYNCHRONOUS EVENTS**

118581            Default.

118582 **STDOUT**118583            The standard output of *type* contains information about each operand in an unspecified format.118584            The information provided typically identifies the operand as a shell built-in, function, alias, or  
118585            keyword, and where applicable, may display the operand's pathname.118586 **STDERR**

118587            The standard error shall be used only for diagnostic messages.

118588 **OUTPUT FILES**

118589 None.

118590 **EXTENDED DESCRIPTION**

118591 None.

118592 **EXIT STATUS**

118593 The following exit values shall be returned:

118594 0 Successful completion.

118595 &gt;0 An error occurred.

118596 **CONSEQUENCES OF ERRORS**

118597 Default.

118598 **APPLICATION USAGE**118599 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

118600 Since *type* must be aware of the contents of the current shell execution environment (such as the  
118601 lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell  
118602 regular built-in. If it is called in a separate utility execution environment, such as one of the  
118603 following:

118604 `nohup type writer`118605 `find . -type f -exec type {} +`

118606 it might not produce accurate results.

118607 **EXAMPLES**

118608 None.

118609 **RATIONALE**

118610 None.

118611 **FUTURE DIRECTIONS**

118612 If this utility is directed to display a pathname that contains any bytes that have the encoded  
118613 value of a <newline> character when <newline> is a terminator or separator in the output  
118614 format being used, implementations are encouraged to treat this as an error. A future version of  
118615 this standard may require implementations to treat this as an error.

118616 **SEE ALSO**118617 [command](#), [hash](#)118618 [XBD Chapter 8](#) (on page 167)118619 **CHANGE HISTORY**

118620 First released in Issue 2.

118621 **Issue 8**118622 Austin Group Defect 248 is applied, changing a command line in the APPLICATION USAGE  
118623 section.

118624 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
118625 directed to display a pathname that contains any bytes that have the encoded value of a  
118626 <newline> character when <newline> is a terminator or separator in the output format being  
118627 used.

118628 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
118629 this utility is required to be intrinsic.

118630

Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

118631 **NAME**

118632 ulimit — report or set resource limits

118633 **SYNOPSIS**

118634 ulimit [-H|-S] -a

118635 XSI ulimit [-H|-S] [-c|-d|-f|-n|-s|-t|-v] [*newlimit*]118636 **DESCRIPTION**118637 The *ulimit* utility shall report or set the resource limits in effect in the process in which it is  
118638 executed.118639 Soft limits can be changed by a process to any value that is less than or equal to the hard limit. A  
118640 process can (irreversibly) lower its hard limit to any value that is greater than or equal to the soft  
118641 limit. Only a process with appropriate privileges can raise a hard limit.118642 The value **unlimited** for a resource shall be considered to be larger than any other limit value.  
118643 When a resource has this limit value, the implementation shall not enforce limits on that  
118644 resource. In locales other than the POSIX locale, *ulimit* may support additional non-numeric  
118645 values with the same meaning as **unlimited**.118646 The behavior when resource limits are exceeded shall be as described in the System Interfaces  
118647 volume of POSIX.1-2024 for the *setrlimit()* function.118648 **OPTIONS**118649 The *ulimit* utility shall conform to XBD [Section 12.2](#) (on page 215), except that:

- 118650 • The order in which options other than **-H**, **-S**, and **-a** are specified may be significant.
- 118651 • Conforming applications shall specify each option separately; that is, grouping option  
118652 letters (for example, **-fH**) need not be recognized by all implementations.

118653 The following options shall be supported:

- |        |           |                                                                                                                                                                                                                                                                |
|--------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 118654 | <b>-H</b> | Report hard limit(s) or set only a hard limit.                                                                                                                                                                                                                 |
| 118655 | <b>-S</b> | Report soft limit(s) or set only a soft limit.                                                                                                                                                                                                                 |
| 118656 | <b>-a</b> | Report the limit value for all of the resources named below and for any<br>118657 implementation-specific additional resources.                                                                                                                                |
| 118658 | <b>-c</b> | Report, or set if the <i>newlimit</i> operand is present, the core image size limit(s) in units<br>118659 of 512 bytes. [RLIMIT_CORE]                                                                                                                          |
| 118660 | <b>-d</b> | Report, or set if the <i>newlimit</i> operand is present, the data segment size limit(s) in<br>118661 units of 1 024 bytes. [RLIMIT_DATA]                                                                                                                      |
| 118662 | <b>-f</b> | Report, or set if the <i>newlimit</i> operand is present, the file size limit(s) in units of 512<br>118663 bytes. [RLIMIT_FSIZE]                                                                                                                               |
| 118664 | <b>-n</b> | Report, or set if the <i>newlimit</i> operand is present, the limit(s) on the number of open<br>118665 file descriptors, given as a number one greater than the maximum value that the<br>118666 system assigns to a newly-created descriptor. [RLIMIT_NOFILE] |
| 118667 | <b>-s</b> | Report, or set if the <i>newlimit</i> operand is present, the stack size limit(s) in units of<br>118668 1 024 bytes. [RLIMIT_STACK]                                                                                                                            |
| 118669 | <b>-t</b> | Report, or set if the <i>newlimit</i> operand is present, the per-process CPU time limit(s)<br>118670 in units of seconds. [RLIMIT_CPU]                                                                                                                        |

118671        **-v**            Report, or set if the *newlimit* operand is present, the address space size limit(s) in  
118672                    units of 1 024 bytes. [RLIMIT\_AS]

118673        Where an option description is followed by [RLIMIT\_name] it indicates which resource for the  
118674        *getrlimit()* and *setrlimit()* functions, defined in the System Interfaces volume of POSIX.1-2024,  
118675        the option corresponds to.

118676        If neither the **-H** nor **-S** option is specified:

- 118677            • If the *newlimit* operand is present, it shall be used as the new value for both the hard and  
118678            soft limits.
- 118679            • If the *newlimit* operand is not present, **-S** shall be the default.

118680        If no options other than **-H** or **-S** are specified, the behavior shall be as if the **-f** option was  
118681        (also) specified.

118682        If any option other than **-H** or **-S** is repeated, the behavior is unspecified.

#### 118683 OPERANDS

118684        The following operand shall be supported:

118685        *newlimit*    Either an integer value to use as the new limit(s) for the specified resource, in the  
118686                    units specified in OPTIONS, or a non-numeric string indicating no limit, as  
118687                    described in the DESCRIPTION section. Numerals in the range 0 to the maximum  
118688                    limit value supported by the implementation for any resource shall be syntactically  
118689                    recognized as numeric values.

#### 118690 STDIN

118691        Not used.

#### 118692 INPUT FILES

118693        None.

#### 118694 ENVIRONMENT VARIABLES

118695        The following environment variables shall affect the execution of *ulimit*:

118696        *LANG*        Provide a default value for the internationalization variables that are unset or null.  
118697                    (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
118698                    variables used to determine the values of locale categories.)

118699        *LC\_ALL*     If set to a non-empty string value, override the values of all the other  
118700                    internationalization variables.

118701        *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
118702                    characters (for example, single-byte as opposed to multi-byte characters in  
118703                    arguments).

118704        *LC\_MESSAGES*

118705                    Determine the locale that should be used to affect the format and contents of  
118706                    diagnostic messages written to standard error.

118707        *NLSPATH*    Determine the location of messages objects and message catalogs.

#### 118708 ASYNCHRONOUS EVENTS

118709        Default.

118710 **STDOUT**

118711 The standard output shall be used when no *newlimit* operand is present.

118712 If the **-a** option is specified, the output written for each resource shall consist of one line that  
118713 includes:

- 118714 • A short phrase identifying the resource (for example `file size`).
- 118715 • An indication of the units used for the resource, if the corresponding option description in  
118716 **OPTIONS** specifies the units to be used.
- 118717 • The *ulimit* option used to specify the resource.
- 118718 • The limit value.

118719 The format used within each line is unspecified, except that the format used for the limit value  
118720 shall be as described below for the case where a single limit value is written.

118721 If a single limit value is to be written; that is, the **-a** option is not specified and at most one  
118722 option other than **-H** or **-S** is specified:

- 118723 • If the resource being reported has a numeric limit, the limit value shall be written in the  
118724 following format:

118725 `"%ld\n", <limit value>`

118726 where *<limit value>* is the value of the limit in the units specified in **OPTIONS**.

- 118727 • If the resource being reported does not have a numeric limit, in the POSIX locale the  
118728 following format shall be used:

118729 `"unlimited\n"`

118730 **STDERR**

118731 The standard error shall be used only for diagnostic messages.

118732 **OUTPUT FILES**

118733 None.

118734 **EXTENDED DESCRIPTION**

118735 None.

118736 **EXIT STATUS**

118737 The following exit values shall be returned:

118738 0 Successful completion.

118739 >0 A request for a higher limit was rejected or an error occurred.

118740 **CONSEQUENCES OF ERRORS**

118741 Default.

118742 **APPLICATION USAGE**

118743 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

118744 Since *ulimit* affects the current shell execution environment, it is always provided as a shell  
118745 regular built-in. If it is called with an operand in a separate utility execution environment, such  
118746 as one of the following:

118747 `nohup ulimit -f 10000`

118748 `env ulimit -S -c 10000`

118749 it does not affect the limit(s) in the caller's environment.

118750 See also the APPLICATION USAGE for *getrlimit()*.

#### 118751 EXAMPLES

118752 Set the hard and soft file size limits to 51 200 bytes:

```
118753 ulimit -f 100
```

118754 Save and restore a soft resource limit (where *X* is an option letter specifying a resource):

```
118755 saved=$(ulimit -X)
```

```
118756 ...
```

```
118757 ulimit -X -S "$saved"
```

118758 Execute a utility with a CPU limit of 5 minutes (using an asynchronous subshell to ensure the  
118759 limit is set in a child process):

```
118760 (ulimit -t 300; exec utility_name </dev/null) &  
118761 wait $!
```

#### 118762 RATIONALE

118763 The *ulimit* utility has no equivalent of the special values RLIM\_SAVED\_MAX and  
118764 RLIM\_SAVED\_CUR returned by *getrlimit()*, as *ulimit* is required to be able to output, and accept  
118765 as input, all numeric limit values supported by the system.

118766 Implementations differ in their behavior when the *-a* option is not specified and more than one  
118767 option other than *-H* or *-S* is specified. Some write output for all of the specified resources in  
118768 the same format as for *-a*; others write only the value for the last specified option. Both  
118769 behaviors are allowed by the standard, since the SYNOPSIS lists the options as mutually  
118770 exclusive.

#### 118771 FUTURE DIRECTIONS

118772 None.

#### 118773 SEE ALSO

118774 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

118775 XSH *getrlimit()*

#### 118776 CHANGE HISTORY

118777 First released in Issue 2.

#### 118778 Issue 7

118779 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

#### 118780 Issue 8

118781 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
118782 this utility is required to be intrinsic.

118783 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

118784 Austin Group Defect 1418 is applied, adding the *-H*, *-S*, *-a*, *-c*, *-d*, *-n*, *-s*, *-t*, and *-v* options,  
118785 and relating the *-f* option to the RLIMIT\_FSIZE resource for *setrlimit()*.

118786 Austin Group Defect 1669 is applied, moving the *ulimit* utility, excluding the *-t* option, from the  
118787 XSI option to the Base.

118788 **NAME**

118789 umask — get or set the file mode creation mask

118790 **SYNOPSIS**118791 umask [-S] [*mask*]118792 **DESCRIPTION**

118793 The *umask* utility shall set the file mode creation mask of the current shell execution environment  
 118794 (see [Section 2.13](#), on page 2522) to the value specified by the *mask* operand. This mask shall affect  
 118795 the initial value of the file permission bits of subsequently created files. If *umask* is called in a  
 118796 subshell or separate utility execution environment, such as one of the following:

```
118797 (umask 002)
118798 nohup umask ...
118799 find . -exec umask ... \;
```

118800 it shall not affect the file mode creation mask of the caller's environment.

118801 If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of  
 118802 the file mode creation mask of the invoking process.

118803 **OPTIONS**118804 The *umask* utility shall conform to XBD [Section 12.2](#) (on page 215).

118805 The following option shall be supported:

118806 **-S** Produce symbolic output.

118807 The default output style is unspecified, but shall be recognized on a subsequent invocation of  
 118808 *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

118809 **OPERANDS**

118810 The following operand shall be supported:

118811 *mask* A string specifying the new file mode creation mask. The string is treated in the  
 118812 same way as the *mode* operand described in the EXTENDED DESCRIPTION  
 118813 section for *chmod*.

118814 For a *symbolic\_mode* value, the new value of the file mode creation mask shall be  
 118815 the logical complement of the file permission bits portion of the file mode specified  
 118816 by the *symbolic\_mode* string.

118817 In a *symbolic\_mode* value, the permissions *op* characters '+' and '-' shall be  
 118818 interpreted relative to the current file mode creation mask; '+' shall cause the bits  
 118819 for the indicated permissions to be cleared in the mask; '-' shall cause the bits for  
 118820 the indicated permissions to be set in the mask.

118821 The interpretation of *mode* values that specify file mode bits other than the file  
 118822 permission bits is unspecified.

118823 In the octal integer form of *mode*, the specified bits are set in the file mode creation  
 118824 mask.

118825 The file mode creation mask shall be set to the resulting numeric value.

118826 The default output of a prior invocation of *umask* on the same system with no  
 118827 operand also shall be recognized as a *mask* operand.



118828 **STDIN**

118829 Not used.

118830 **INPUT FILES**

118831 None.

118832 **ENVIRONMENT VARIABLES**118833 The following environment variables shall affect the execution of *umask*:

118834 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 118835 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 118836 variables used to determine the values of locale categories.)

118837 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 118838 internationalization variables.

118839 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 118840 characters (for example, single-byte as opposed to multi-byte characters in  
 118841 arguments).

118842 *LC\_MESSAGES*

118843 Determine the locale that should be used to affect the format and contents of  
 118844 diagnostic messages written to standard error.

118845 *XSHELL* *NLSPATH* Determine the location of messages objects and message catalogs.

118846 **ASYNCHRONOUS EVENTS**

118847 Default.

118848 **STDOUT**

118849 When the *mask* operand is not specified, the *umask* utility shall write a message to standard  
 118850 output that can later be used as a *umask mask* operand.

118851 If *-S* is specified, the message shall be in the following format:

118852 "u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,  
 118853 <other permissions>

118854 where the three values shall be combinations of letters from the set {*r, w, x*}; the presence of a  
 118855 letter shall indicate that the corresponding bit is clear in the file mode creation mask.

118856 If a *mask* operand is specified, there shall be no output written to standard output.

118857 **STDERR**

118858 The standard error shall be used only for diagnostic messages.

118859 **OUTPUT FILES**

118860 None.

118861 **EXTENDED DESCRIPTION**

118862 None.

118863 **EXIT STATUS**

118864 The following exit values shall be returned:

118865 0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

118866 >0 An error occurred.

## 118867 CONSEQUENCES OF ERRORS

118868 Default.

## 118869 APPLICATION USAGE

118870 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.118871 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
118872 regular built-in.118873 In contrast to the negative permission logic provided by the file mode creation mask and the  
118874 octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those  
118875 permissions that are left alone.

## 118876 EXAMPLES

118877 Either of the commands:

118878 `umask a=rX,ug+w`118879 `umask 002`

118880 sets the mode mask so that subsequently created files have their S\_IWOTH bit cleared.

118881 After setting the mode mask with either of the above commands, the *umask* command can be  
118882 used to write out the current value of the mode mask:118883 `$ umask`118884 **0002**118885 (The output format is unspecified, but historical implementations use the octal integer mode  
118886 format.)118887 `$ umask -S`118888 **u=rwx,g=rwx,o=rX**118889 Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask*  
118890 utility.

118891 Assuming the mode mask is set as above, the command:

118892 `umask g-w`118893 sets the mode mask so that subsequently created files have their S\_IWGRP and S\_IWOTH bits  
118894 cleared.

118895 The command:

118896 `umask -- -w`118897 sets the mode mask so that subsequently created files have all their write bits cleared. Note that  
118898 *mask* operands `-r`, `-w`, `-x` or anything beginning with a <hyphen-minus>, must be preceded by  
118899 `"--"` to keep it from being interpreted as an option.

## 118900 RATIONALE

118901 Since *umask* affects the current shell execution environment, it is generally provided as a shell  
118902 regular built-in. If it is called in a subshell or separate utility execution environment, such as one  
118903 of the following:118904 `(umask 002)`118905 `nohup umask ...`118906 `find . -exec umask ... \;`

118907 it does not affect the file mode creation mask of the environment of the caller.

118908 The description of the historical utility was modified to allow it to use the symbolic modes of  
118909 *chmod*. The `-s` option used in early proposals was changed to `-S` because `-s` could be confused  
118910 with a *symbolic\_mode* form of mask referring to the `S_ISUID` and `S_ISGID` bits.

118911 The default output style is unspecified to permit implementors to provide migration to the new  
118912 symbolic style at the time most appropriate to their users. A `-o` flag to force octal mode output  
118913 was omitted because the octal mode may not be sufficient to specify all of the information that  
118914 may be present in the file mode creation mask when more secure file access permission checks  
118915 are implemented.

118916 It has been suggested that trusted systems developers might appreciate ameliorating the  
118917 requirement that the mode mask “affects” the file access permissions, since it seems access  
118918 control lists might replace the mode mask to some degree. The wording has been changed to say  
118919 that it affects the file permission bits, and it leaves the details of the behavior of how they affect  
118920 the file access permissions to the description in the System Interfaces volume of POSIX.1-2024.

#### 118921 FUTURE DIRECTIONS

118922 None.

#### 118923 SEE ALSO

118924 [Chapter 2](#) (on page 2472), *chmod*

118925 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

118926 XSH *umask()*

#### 118927 CHANGE HISTORY

118928 First released in Issue 2.

#### 118929 Issue 6

118930 The following new requirements on POSIX implementations derive from alignment with the  
118931 Single UNIX Specification:

- 118932 • The octal mode is supported.

118933 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/34 is applied, making a correction to the  
118934 RATIONALE.

#### 118935 Issue 7

118936 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

118937 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0197 [584] is applied.

#### 118938 Issue 8

118939 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
118940 this utility is required to be intrinsic.

118941 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

118942 **NAME**

118943 unalias — remove alias definitions

118944 **SYNOPSIS**118945 unalias *alias-name*...

118946 unalias -a

118947 **DESCRIPTION**

118948 The *unalias* utility shall remove the definition for each alias name specified. See [Section 2.3.1](#) (on  
 118949 page 2477). The aliases shall be removed from the current shell execution environment; see  
 118950 [Section 2.13](#) (on page 2522).

118951 **OPTIONS**118952 The *unalias* utility shall conform to XBD [Section 12.2](#) (on page 215).

118953 The following option shall be supported:

118954 **-a** Remove all alias definitions from the current shell execution environment.118955 **OPERANDS**

118956 The following operand shall be supported:

118957 *alias-name* The name of an alias to be removed.118958 **STDIN**

118959 Not used.

118960 **INPUT FILES**

118961 None.

118962 **ENVIRONMENT VARIABLES**118963 The following environment variables shall affect the execution of *unalias*:

118964 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 118965 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 118966 variables used to determine the values of locale categories.)

118967 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 118968 internationalization variables.

118969 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 118970 characters (for example, single-byte as opposed to multi-byte characters in  
 118971 arguments).

118972 **LC\_MESSAGES**

118973 Determine the locale that should be used to affect the format and contents of  
 118974 diagnostic messages written to standard error.

118975 **XSI** **NLSPATH** Determine the location of messages objects and message catalogs.118976 **ASYNCHRONOUS EVENTS**

118977 Default.

118978 **STDOUT**

118979 Not used.

118980 **STDERR**

118981 The standard error shall be used only for diagnostic messages.

**118982 OUTPUT FILES**

118983 None.

**118984 EXTENDED DESCRIPTION**

118985 None.

**118986 EXIT STATUS**

118987 The following exit values shall be returned:

118988 0 Successful completion.

118989 >0 One of the *alias-name* operands specified did not represent a valid alias definition, or an  
118990 error occurred.

**118991 CONSEQUENCES OF ERRORS**

118992 Default.

**118993 APPLICATION USAGE**

118994 This utility is required to be intrinsic. See [Section 1.7](#) (on page 2470) for details.

118995 Since *unalias* affects the current shell execution environment, it is generally provided as a shell  
118996 regular built-in.

**118997 EXAMPLES**

118998 None.

**118999 RATIONALE**

119000 The *unalias* description is based on that from historical KornShell implementations. Known  
119001 differences exist between that and the C shell. The KornShell version was adopted to be  
119002 consistent with all the other KornShell features in this volume of POSIX.1-2024, such as  
119003 command line editing.

119004 The `-a` option is the equivalent of the *unalias \** form of the C shell and is provided to address  
119005 security concerns about unknown aliases entering the environment of a user (or application)  
119006 through the allowable implementation-defined predefined alias route or as a result of an *ENV*  
119007 file. (Although *unalias* could be used to simplify the “secure” shell script shown in the *command*  
119008 rationale, it does not obviate the need to quote all command names. An initial call to *unalias -a*  
119009 would have to be quoted in case there was an alias for *unalias*.)

**119010 FUTURE DIRECTIONS**

119011 None.

**119012 SEE ALSO**

119013 [Chapter 2](#) (on page 2472), *alias*

119014 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

**119015 CHANGE HISTORY**

119016 First released in Issue 4.

**119017 Issue 6**

119018 This utility is marked as part of the User Portability Utilities option.

**119019 Issue 7**

119020 The *unalias* utility is moved from the User Portability Utilities option to the Base. User  
119021 Portability Utilities is now an option for interactive utilities.

119022 **Issue 8**

119023 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
119024 this utility is required to be intrinsic.

119025 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

119026 **NAME**

119027            uname — return system name

119028 **SYNOPSIS**

119029            uname [-amnrsv]

119030 **DESCRIPTION**

119031            By default, the *uname* utility shall write the operating system name to standard output. When  
 119032            options are specified, symbols representing one or more system characteristics shall be written to  
 119033            the standard output. The format and contents of the symbols are implementation-defined. On  
 119034            systems conforming to the System Interfaces volume of POSIX.1-2024, the symbols written shall  
 119035            be those supported by the *uname()* function as defined in the System Interfaces volume of  
 119036            POSIX.1-2024.

119037 **OPTIONS**119038            The *uname* utility shall conform to XBD [Section 12.2](#) (on page 215).

119039            The following options shall be supported:

- 119040            **-a**            Behave as though all of the options **-mnrsv** were specified.
- 119041            **-m**            Write the name of the hardware type on which the system is running to standard  
 119042            output.
- 119043            **-n**            Write the name of this node within an implementation-defined communications  
 119044            network.
- 119045            **-r**            Write the current release level of the operating system implementation.
- 119046            **-s**            Write the name of the implementation of the operating system.
- 119047            **-v**            Write the current version level of this release of the operating system  
 119048            implementation.

119049            If no options are specified, the *uname* utility shall write the operating system name, as if the **-s**  
 119050            option had been specified.

119051 **OPERANDS**

119052            None.

119053 **STDIN**

119054            Not used.

119055 **INPUT FILES**

119056            None.

119057 **ENVIRONMENT VARIABLES**119058            The following environment variables shall affect the execution of *uname*:

- 119059            **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 119060            (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 119061            variables used to determine the values of locale categories.)
- 119062            **LC\_ALL**            If set to a non-empty string value, override the values of all the other  
 119063            internationalization variables.
- 119064            **LC\_CTYPE**        Determine the locale for the interpretation of sequences of bytes of text data as  
 119065            characters (for example, single-byte as opposed to multi-byte characters in  
 119066            arguments).

119067 **LC\_MESSAGES**  
119068 Determine the locale that should be used to affect the format and contents of  
119069 diagnostic messages written to standard error.

119070 XSI **NLSPATH** Determine the location of messages objects and message catalogs.

119071 **ASYNCHRONOUS EVENTS**  
119072 Default.

119073 **STDOUT**  
119074 By default, the output shall be a single line of the following form:  
119075 "%s\n", <sysname>  
119076 If the **-a** option is specified, the output shall be a single line of the following form:  
119077 "%s %s %s %s %s\n", <sysname>, <nodename>, <release>,  
119078 <version>, <machine>  
119079 Additional implementation-defined symbols may be written; all such symbols shall be written at  
119080 the end of the line of output before the <newline>.  
119081 If options are specified to select different combinations of the symbols, only those symbols shall  
119082 be written, in the order shown above for the **-a** option. If a symbol is not selected for writing, its  
119083 corresponding trailing <blank> characters also shall not be written.

119084 **STDERR**  
119085 The standard error shall be used only for diagnostic messages.

119086 **OUTPUT FILES**  
119087 None.

119088 **EXTENDED DESCRIPTION**  
119089 None.

119090 **EXIT STATUS**  
119091 The following exit values shall be returned:  
119092 0 The requested information was successfully written.  
119093 >0 An error occurred.

119094 **CONSEQUENCES OF ERRORS**  
119095 Default.

119096 **APPLICATION USAGE**  
119097 Note that any of the symbols could include embedded <space> characters, which may affect  
119098 parsing algorithms if multiple options are selected for output.  
119099 The node name is typically a name that the system uses to identify itself for inter-system  
119100 communication addressing.

119101 **EXAMPLES**  
119102 The following command:  
119103 `uname -sr`  
119104 writes the operating system name and release level, separated by one or more <blank>  
119105 characters.



119106 **RATIONALE**

119107 It was suggested that this utility cannot be used portably since the format of the symbols is  
119108 implementation-defined. The POSIX.1 working group could not achieve consensus on defining  
119109 these formats in the underlying *uname()* function, and there was no expectation that this volume  
119110 of POSIX.1-2024 would be any more successful. Some applications may still find this historical  
119111 utility of value. For example, the symbols could be used for system log entries or for comparison  
119112 with operator or user input.

119113 **FUTURE DIRECTIONS**

119114 None.

119115 **SEE ALSO**

119116 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

119117 XSH *uname()*

119118 **CHANGE HISTORY**

119119 First released in Issue 2.

119120 **Issue 8**

119121 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

119122 **NAME**

119123 uncompress — expand compressed data

119124 **SYNOPSIS**

119125 XSI `uncompress [-cfv] [file...]`

119126 **DESCRIPTION**

119127 Refer to *compress*.

119128 **NAME**

119129 unexpand — convert spaces to tabs

119130 **SYNOPSIS**119131 unexpand [-a|-t *tablist*] [*file...*]119132 **DESCRIPTION**

119133 The *unexpand* utility shall copy files or standard input to standard output, converting <blank>  
 119134 characters at the beginning of each line into the maximum number of <tab> characters followed  
 119135 by the minimum number of <space> characters needed to fill the same column positions  
 119136 originally filled by the translated <blank> characters. By default, tabstops shall be set at every  
 119137 eighth column position. Each <backspace> shall be copied to the output, and shall cause the  
 119138 column position count for tab calculations to be decremented; the count shall never be  
 119139 decremented to a value less than one.

119140 **OPTIONS**119141 The *unexpand* utility shall conform to XBD [Section 12.2](#) (on page 215).

119142 The following options shall be supported:

119143 **-a** In addition to translating <blank> characters at the beginning of each line,  
 119144 translate all sequences of two or more <blank> characters immediately preceding a  
 119145 tab stop to the maximum number of <tab> characters followed by the minimum  
 119146 number of <space> characters needed to fill the same column positions originally  
 119147 filled by the translated <blank> characters.

119148 **-t *tablist*** Specify the tab stops. The application shall ensure that the *tablist* option-argument  
 119149 is a single argument consisting of a single positive decimal integer or multiple  
 119150 positive decimal integers, separated by <blank> or <comma> characters, in  
 119151 ascending order. If a single number is given, tabs shall be set *tablist* column  
 119152 positions apart instead of the default 8. If multiple numbers are given, the tabs  
 119153 shall be set at those specific column positions.

119154 The application shall ensure that each tab-stop position *N* is an integer value  
 119155 greater than zero, and the list shall be in strictly ascending order. This is taken to  
 119156 mean that, from the start of a line of output, tabbing to position *N* shall cause the  
 119157 next character output to be in the (*N*+1)th column position on that line. When the  
 119158 **-t** option is not specified, the default shall be the equivalent of specifying **-t 8**  
 119159 (except for the interaction with **-a**, described below).

119160 No <space>-to-<tab> conversions shall occur for characters at positions beyond  
 119161 the last of those specified in a multiple tab-stop list.

119162 When **-t** is specified, the presence or absence of the **-a** option shall be ignored;  
 119163 conversion shall not be limited to the processing of leading <blank> characters.

119164 **OPERANDS**

119165 The following operand shall be supported:

119166 *file* A pathname of a text file to be used as input.119167 **STDIN**

119168 See the INPUT FILES section.

119169 **INPUT FILES**

119170 The input files shall be text files.

119171 **ENVIRONMENT VARIABLES**

119172 The following environment variables shall affect the execution of *unexpand*:

119173 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 119174 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 119175 variables used to determine the values of locale categories.)

119176 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 119177 internationalization variables.

119178 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 119179 characters (for example, single-byte as opposed to multi-byte characters in  
 119180 arguments and input files), the processing of <tab> and <space> characters, and  
 119181 for the determination of the width in column positions each character would  
 119182 occupy on an output device.

119183 *LC\_MESSAGES*

119184 Determine the locale that should be used to affect the format and contents of  
 119185 diagnostic messages written to standard error.

119186 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

119187 **ASYNCHRONOUS EVENTS**

119188 Default.

119189 **STDOUT**

119190 The standard output shall be equivalent to the input files with the specified <space>-to-<tab>  
 119191 conversions.

119192 **STDERR**

119193 The standard error shall be used only for diagnostic messages.

119194 **OUTPUT FILES**

119195 None.

119196 **EXTENDED DESCRIPTION**

119197 None.

119198 **EXIT STATUS**

119199 The following exit values shall be returned:

119200 0 Successful completion.

119201 >0 An error occurred.

119202 **CONSEQUENCES OF ERRORS**

119203 Default.

119204 **APPLICATION USAGE**

119205 One non-intuitive aspect of *unexpand* is its restriction to leading <space> characters when neither  
 119206 *-a* nor *-t* is specified. Users who always want to convert all <space> characters in a file can  
 119207 easily alias *unexpand* to use the *-a* or *-t 8* option.

119208 **EXAMPLES**

119209 None.

119210 **RATIONALE**

119211 On several occasions, consideration was given to adding a *-t* option to the *unexpand* utility to  
 119212 complement the *-t* in *expand* (see *expand*). The historical intent of *unexpand* was to translate  
 119213 multiple <blank> characters into tab stops, where tab stops were a multiple of eight column

119214 positions on most UNIX systems. An early proposal omitted `-t` because it seemed outside the  
119215 scope of the User Portability Utilities option; it was not described in any of the base documents  
119216 for IEEE Std 1003.2-1992. However, hard-coding tab stops every eight columns was not suitable  
119217 for the international community and broke historical precedents for some vendors in the  
119218 FORTRAN community, so `-t` was restored in conjunction with the list of valid extension  
119219 categories considered by the standard developers. Thus, *unexpand* is now the logical converse of  
119220 *expand*.

#### 119221 **FUTURE DIRECTIONS**

119222 None.

#### 119223 **SEE ALSO**

119224 *expand*, *tabs*

119225 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 119226 **CHANGE HISTORY**

119227 First released in Issue 4.

#### 119228 **Issue 6**

119229 This utility is marked as part of the User Portability Utilities option.

119230 The definition of the `LC_CTYPE` environment variable is changed to align with the  
119231 IEEE P1003.2b draft standard.

119232 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 119233 **Issue 7**

119234 The *unexpand* utility is moved from the User Portability Utilities option to the Base. User  
119235 Portability Utilities is now an option for interactive utilities.

119236 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

119237 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0198 [885] is applied.

#### 119238 **Issue 8**

119239 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.

119240 **NAME**119241 unget — undo a previous get of an SCCS file (**DEVELOPMENT**)119242 **SYNOPSIS**119243 XSI unget [-ns] [-r *SID*] *file...*119244 **DESCRIPTION**119245 The *unget* utility shall reverse the effect of a *get -e* done prior to creating the intended new delta.119246 **OPTIONS**119247 The *unget* utility shall conform to XBD [Section 12.2](#) (on page 215).

119248 The following options shall be supported:

119249 **-r** *SID* Uniquely identify which delta is no longer intended. (This would have been  
 119250 specified by *get* as the new delta.) The use of this option is necessary only if two or  
 119251 more outstanding *get* commands for editing on the same SCCS file were done by  
 119252 the same person (login name).

119253 **-s** Suppress the writing to standard output of the intended delta's *SID*.

119254 **-n** Retain the file that was obtained by *get*, which would normally be removed from  
 119255 the current directory.

119256 **OPERANDS**

119257 The following operands shall be supported:

119258 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *unget*  
 119259 utility shall behave as though each file in the directory were specified as a named  
 119260 file, except that non-SCCS files (last component of the pathname does not begin  
 119261 with **s**.) and unreadable files shall be silently ignored.

119262 If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
 119263 each line of the standard input shall be taken to be the name of an SCCS file to be  
 119264 processed. Non-SCCS files and unreadable files shall be silently ignored.

119265 **STDIN**

119266 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
 119267 line of the text file shall be interpreted as an SCCS pathname.

119268 **INPUT FILES**

119269 Any SCCS files processed shall be files of an unspecified format.

119270 **ENVIRONMENT VARIABLES**119271 The following environment variables shall affect the execution of *unget*:

119272 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 119273 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 119274 variables used to determine the values of locale categories.)

119275 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 119276 internationalization variables.

119277 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 119278 characters (for example, single-byte as opposed to multi-byte characters in  
 119279 arguments and input files).

119280 **LC\_MESSAGES**

119281 Determine the locale that should be used to affect the format and contents of  
 119282 diagnostic messages written to standard error.

- 119283 *NLSPATH* Determine the location of messages objects and message catalogs.
- 119284 **ASYNCHRONOUS EVENTS**
- 119285 Default.
- 119286 **STDOUT**
- 119287 The standard output shall consist of a line for each file, in the following format:
- 119288 "%s\n", <*SID removed from file*>
- 119289 If there is more than one named file or if a directory or standard input is named, each pathname shall be written before each of the preceding lines:
- 119291 "\n%s:\n", <*pathname*>
- 119292 **STDERR**
- 119293 The standard error shall be used only for diagnostic messages.
- 119294 **OUTPUT FILES**
- 119295 Any SCCS files updated shall be files of an unspecified format. During processing of a *file*, a locking *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and deleted. The *p-file* and *g-file*, as described in *get*, shall be deleted.
- 119298 **EXTENDED DESCRIPTION**
- 119299 None.
- 119300 **EXIT STATUS**
- 119301 The following exit values shall be returned:
- 119302 0 Successful completion.
- 119303 >0 An error occurred.
- 119304 **CONSEQUENCES OF ERRORS**
- 119305 Default.
- 119306 **APPLICATION USAGE**
- 119307 None.
- 119308 **EXAMPLES**
- 119309 None.
- 119310 **RATIONALE**
- 119311 None.
- 119312 **FUTURE DIRECTIONS**
- 119313 If this utility is directed to create a new directory entry that contains any bytes that have the encoded value of a <newline> character, implementations are encouraged to treat this as an error. A future version of this standard may require implementations to treat this as an error.
- 119316 **SEE ALSO**
- 119317 *delta*, *get*, *sact*
- 119318 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)
- 119319 **CHANGE HISTORY**
- 119320 First released in Issue 2.

119321 **Issue 6**

119322 The normative text is reworded to avoid use of the term “must” for application requirements.

119323 **Issue 7**

119324 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

119325 **Issue 8**

119326 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
119327 filenames containing any bytes that have the encoded value of a <newline> character.

119328 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.



119329 **NAME**

119330            `uniq` — report or filter out repeated lines in a file

119331 **SYNOPSIS**

119332            `uniq [-c|-d|-u] [-f fields] [-s char] [input_file [output_file]]`

119333 **DESCRIPTION**

119334            The *uniq* utility shall read an input file comparing adjacent lines, and write one copy of each  
 119335            input line on the output. The second and succeeding copies of repeated adjacent input lines shall  
 119336            not be written. The trailing <newline> of each line in the input shall be ignored when doing  
 119337            comparisons.

119338            Repeated lines in the input shall not be detected if they are not adjacent.

119339 **OPTIONS**

119340            The *uniq* utility shall conform to XBD [Section 12.2](#) (on page 215), except that '+' may be  
 119341            recognized as an option delimiter as well as '-'.

119342            The following options shall be supported:

119343            **-c**            Precede each output line with a count of the number of times the line occurred in  
 119344            the input.

119345            **-d**            Suppress the writing of lines that are not repeated in the input.

119346            **-f *fields***       Ignore the first *fields* fields on each input line when doing comparisons, where  
 119347            *fields* is a positive decimal integer. A field is the maximal string matched by the  
 119348            basic regular expression:

119349            `[[:blank:]]*^[[:blank:]]*`

119350            If the *fields* option-argument specifies more fields than appear on an input line, a  
 119351            null string shall be used for comparison.

119352            **-s *chars***       Ignore the first *chars* characters when doing comparisons, where *chars* shall be a  
 119353            positive decimal integer. If specified in conjunction with the **-f** option, the first  
 119354            *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-  
 119355            argument specifies more characters than remain on an input line, a null string shall  
 119356            be used for comparison.

119357            **-u**            Suppress the writing of lines that are repeated in the input.

119358 **OPERANDS**

119359            The following operands shall be supported:

119360            *input\_file*       A pathname of the input file. If the *input\_file* operand is not specified, or if the  
 119361            *input\_file* is '-', the standard input shall be used.

119362            *output\_file*       A pathname of the output file. If the *output\_file* operand is not specified, the  
 119363            standard output shall be used. The results are unspecified if the file named by  
 119364            *output\_file* is the file named by *input\_file*.

119365 **STDIN**

119366            The standard input shall be used only if no *input\_file* operand is specified or if *input\_file* is '-'.

119367            See the INPUT FILES section.

119368 **INPUT FILES**

119369            The input file shall be a text file.

119370 **ENVIRONMENT VARIABLES**

119371 The following environment variables shall affect the execution of *unig*:

119372 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 119373 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 119374 variables used to determine the values of locale categories.)

119375 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 119376 internationalization variables.

119377 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 119378 characters (for example, single-byte as opposed to multi-byte characters in  
 119379 arguments and input files) and which characters constitute a <blank> in the  
 119380 current locale.

119381 *LC\_MESSAGES*

119382 Determine the locale that should be used to affect the format and contents of  
 119383 diagnostic messages written to standard error.

119384 XSI *NLSPATH* Determine the location of messages objects and message catalogs.

119385 **ASYNCHRONOUS EVENTS**

119386 Default.

119387 **STDOUT**

119388 The standard output shall be used if no *output\_file* operand is specified, and shall be used if the  
 119389 *output\_file* operand is '-' and the implementation treats the '-' as meaning standard output.  
 119390 Otherwise, the standard output shall not be used. See the OUTPUT FILES section.

119391 **STDERR**

119392 The standard error shall be used only for diagnostic messages.

119393 **OUTPUT FILES**

119394 If the *-c* option is specified, the output file shall be empty or each line shall be of the form:

119395 "%d %s", <number of duplicates>, <line>

119396 otherwise, the output file shall be empty or each line shall be of the form:

119397 "%s", <line>

119398 **EXTENDED DESCRIPTION**

119399 None.

119400 **EXIT STATUS**

119401 The following exit values shall be returned:

119402 0 Successful completion.

119403 >0 An error occurred.

119404 **CONSEQUENCES OF ERRORS**

119405 Default.

119406 **APPLICATION USAGE**

119407 The *sort* utility can be used to cause repeated lines to be adjacent in the input file.

119408 If the collating sequence of the current locale does not have a total ordering of all characters, the  
 119409 behavior of `sort | uniq` differs from `sort -u`, as *uniq* treats lines as duplicates only if they  
 119410 are identical, whereas `sort -u` treats lines as duplicates if they collate equally.

119411 When using *uniq* to process pathnames, it is recommended that `LC_ALL`, or at least `LC_CTYPE`  
 119412 and `LC_COLLATE`, are set to `POSIX` or `C` in the environment, since pathnames can contain byte  
 119413 sequences that do not form valid characters in some locales, in which case the utility's behavior  
 119414 would be undefined. In the `POSIX` locale each byte is a valid single-byte character, and therefore  
 119415 this problem is avoided.

119416 **EXAMPLES**

119417 The following input file data (but flushed left) was used for a test series on *uniq*:

```
119418 #01 foo0 bar0 fool bar1
119419 #02 bar0 fool bar1 fool
119420 #03 foo0 bar0 fool bar1
119421 #04
119422 #05 foo0 bar0 fool bar1
119423 #06 foo0 bar0 fool bar1
119424 #07 bar0 fool bar1 foo0
```

119425 What follows is a series of test invocations of the *uniq* utility that use a mixture of *uniq* options  
 119426 against the input file data. These tests verify the meaning of *adjacent*. The *uniq* utility views the  
 119427 input data as a sequence of strings delimited by `'\n'`. Accordingly, for the *fieldsth* member of  
 119428 the sequence, *uniq* interprets unique or repeated adjacent lines strictly relative to the *fields+1th*  
 119429 member.

- 119430 1. This first example tests the line counting option, comparing each line of the input file data  
 119431 starting from the second field:

```
119432 uniq -c -f 1 uniq_0I.t
119433     1 #01 foo0 bar0 fool bar1
119434     1 #02 bar0 fool bar1 fool
119435     1 #03 foo0 bar0 fool bar1
119436     1 #04
119437     2 #05 foo0 bar0 fool bar1
119438     1 #07 bar0 fool bar1 foo0
```

119439 The number '2', prefixing the fifth line of output, signifies that the *uniq* utility detected a  
 119440 pair of repeated lines. Given the input data, this can only be true when *uniq* is run using  
 119441 the `-f 1` option (which shall cause *uniq* to ignore the first field on each input line).

- 119442 2. The second example tests the option to suppress unique lines, comparing each line of the  
 119443 input file data starting from the second field:

```
119444 uniq -d -f 1 uniq_0I.t
119445 #05 foo0 bar0 fool bar1
```

- 119446 3. This test suppresses repeated lines, comparing each line of the input file data starting  
 119447 from the second field:

```
119448 uniq -u -f 1 uniq_0I.t
119449 #01 foo0 bar0 fool bar1
119450 #02 bar0 fool bar1 fool
119451 #03 foo0 bar0 fool bar1
```

119452 #04  
119453 #07 bar0 fool bar1 foo0

119454 4. This suppresses unique lines, comparing each line of the input file data starting from the  
119455 third character:

119456 `uniq -d -s 2 uniq_0I.t`

119457 In the last example, the *uniq* utility found no input matching the above criteria.

#### 119458 RATIONALE

119459 Some historical implementations have limited lines to be 1 080 bytes in length, which does not  
119460 meet the implied {LINE\_MAX} limit.

119461 Earlier versions of this standard allowed the *-number* and *+number* options. These options are no  
119462 longer specified by POSIX.1-2024 but may be present in some implementations.

#### 119463 FUTURE DIRECTIONS

119464 If this utility is directed to create a new directory entry that contains any bytes that have the  
119465 encoded value of a <newline> character, implementations are encouraged to treat this as an  
119466 error. A future version of this standard may require implementations to treat this as an error.

#### 119467 SEE ALSO

119468 *comm*, *sort*

119469 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

#### 119470 CHANGE HISTORY

119471 First released in Issue 2.

#### 119472 Issue 6

119473 The obsolescent SYNOPSIS and associated text are removed.

119474 The normative text is reworded to avoid use of the term “must” for application requirements.

119475 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/40 is applied, adding *LC\_COLLATE* to the  
119476 ENVIRONMENT VARIABLES section, and changing “the application shall ensure that” in the  
119477 OUTPUT FILES section.

#### 119478 Issue 7

119479 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized  
119480 as an option delimiter in the OPTIONS section.

119481 Austin Group Interpretation 1003.1-2001 #092 is applied.

119482 Austin Group Interpretation 1003.1-2001 #133 is applied, clarifying the behavior of the trailing  
119483 <newline>.

119484 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

119485 SD5-XCU-ERN-141 is applied, updating the EXAMPLES section.

119486 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0199 [963] and XCU/TC2-2008/0200  
119487 [663] are applied.

#### 119488 Issue 8

119489 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
119490 filenames containing any bytes that have the encoded value of a <newline> character.

119491 Austin Group Defect 1070 is applied, changing the APPLICATION USAGE section.

119492 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

119493

Austin Group Defect 1492 is applied, changing the EXIT STATUS section.

119494 **NAME**

119495 unlink — call the *unlink()* function

119496 **SYNOPSIS**

119497 XSI unlink *file*

119498 **DESCRIPTION**

119499 The *unlink* utility shall perform the function call:

119500 unlink(*file*);

119501 A user may need appropriate privileges to invoke the *unlink* utility.

119502 **OPTIONS**

119503 None.

119504 **OPERANDS**

119505 The following operands shall be supported:

119506 *file* The pathname of an existing file.

119507 **STDIN**

119508 Not used.

119509 **INPUT FILES**

119510 Not used.

119511 **ENVIRONMENT VARIABLES**

119512 The following environment variables shall affect the execution of *unlink*:

119513 *LANG* Provide a default value for the internationalization variables that are unset or null.  
119514 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
119515 variables used to determine the values of locale categories.)

119516 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
119517 internationalization variables.

119518 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
119519 characters (for example, single-byte as opposed to multi-byte characters in  
119520 arguments).

119521 *LC\_MESSAGES*  
119522 Determine the locale that should be used to affect the format and contents of  
119523 diagnostic messages written to standard error.

119524 *NLSPATH* Determine the location of messages objects and message catalogs.

119525 **ASYNCHRONOUS EVENTS**

119526 Default.

119527 **STDOUT**

119528 None.

119529 **STDERR**

119530 The standard error shall be used only for diagnostic messages.

119531 **OUTPUT FILES**

119532 None.

119533 **EXTENDED DESCRIPTION**

119534 None.

119535 **EXIT STATUS**

119536 The following exit values shall be returned:

119537 0 Successful completion.

119538 &gt;0 An error occurred.

119539 **CONSEQUENCES OF ERRORS**

119540 Default.

119541 **APPLICATION USAGE**

119542 None.

119543 **EXAMPLES**

119544 None.

119545 **RATIONALE**

119546 None.

119547 **FUTURE DIRECTIONS**

119548 None.

119549 **SEE ALSO**119550 *link, rm*119551 XBD [Chapter 8](#) (on page 167)119552 XSH *unlink()*119553 **CHANGE HISTORY**

119554 First released in Issue 5.

119555 **Issue 8**119556 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

119557 **NAME**

119558 uucp — system-to-system copy

119559 **SYNOPSIS**119560 UU `uucp [-cCdfjmr] [-n user] source-file... destination-file`119561 **DESCRIPTION**

119562 The *uucp* utility shall copy files named by the *source-file* argument to the *destination-file* argument.  
 119563 The files named can be on local or remote systems.

119564 The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For  
 119565 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
 119566 filenames need not be portable to non-internationalized systems, and so on. Under these  
 119567 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
 119568 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used,  
 119569 and that only characters defined in the portable filename character set be used for naming files.  
 119570 The protocol for transfer of files is unspecified by POSIX.1-2024.

119571 Typical implementations of this utility require a communications line configured to use XBD  
 119572 [Chapter 11](#) (on page 199), but other communications means may be used. On systems where  
 119573 there are no available communications means (either temporarily or permanently), this utility  
 119574 shall write an error message describing the problem and exit with a non-zero exit status.

119575 **OPTIONS**119576 The *uucp* utility shall conform to XBD [Section 12.2](#) (on page 215).

119577 The following options shall be supported:

- 119578 **-c** Do not copy local file to the spool directory for transfer to the remote machine  
 119579 (default).
- 119580 **-C** Force the copy of local files to the spool directory for transfer.
- 119581 **-d** Make all necessary directories for the file copy (default).
- 119582 **-f** Do not make intermediate directories for the file copy.
- 119583 **-j** Write the job identification string to standard output. This job identification can be  
 119584 used by *uustat* to obtain the status or terminate a job.
- 119585 **-m** Send mail to the requester when the copy is completed.
- 119586 **-n user** Notify *user* on the remote system that a file was sent.
- 119587 **-r** Do not start the file transfer; just queue the job.

119588 **OPERANDS**

119589 The following operands shall be supported:

119590 *destination-file, source-file*

119591 A pathname of a file to be copied to, or from, respectively. Either name can be a  
 119592 pathname on the local machine, or can have the form:

119593 *system-name!pathname*

119594 where *system-name* is taken from a list of system names that *uucp* knows about.  
 119595 The destination *system-name* can also be a list of names such as:

119596 *system-name!system-name!...!system-name!pathname*

119597 in which case, an attempt is made to send the file via the specified route to the



119598 destination. Care should be taken to ensure that intermediate nodes in the route  
119599 are willing to forward information.

119600 The shell pattern matching notation characters '?', '\*', and "[...]" appearing  
119601 in *pathname* shall be expanded on the appropriate system.

119602 Pathnames can be one of:

- 119603 1. An absolute pathname.
- 119604 2. A pathname preceded by *~user* where *user* is a login name on the specified  
119605 system and is replaced by that user's login directory. Note that if an invalid  
119606 login is specified, the default is to the public directory (called *PUBDIR*; the  
119607 actual location of *PUBDIR* is implementation-defined).
- 119608 3. A pathname preceded by *~/destination* where *destination* is appended to  
119609 *PUBDIR*.

119610 **Note:** This destination is treated as a filename unless more than one file is being  
119611 transferred by this request or the destination is already a directory. To  
119612 ensure that it is a directory, follow the destination with a '/'. For  
119613 example, *~/dan/* as the destination makes the directory **PUBDIR/dan** if it  
119614 does not exist and puts the requested files in that directory.

- 119615 4. Anything else shall be prefixed by the current directory.

119616 If the result is an erroneous pathname for the remote system, the copy shall fail. If  
119617 the *destination-file* is a directory, the last part of the *source-file* name shall be used.

119618 The read, write, and execute permissions given by *uucp* are implementation-  
119619 defined.

#### 119620 STDIN

119621 Not used.

#### 119622 INPUT FILES

119623 The files to be copied are regular files.

#### 119624 ENVIRONMENT VARIABLES

119625 The following environment variables shall affect the execution of *uucp*:

119626 *LANG* Provide a default value for the internationalization variables that are unset or null.  
119627 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
119628 variables used to determine the values of locale categories.)

119629 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
119630 internationalization variables.

#### 119631 *LC\_COLLATE*

119632 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
119633 character collating elements within bracketed filename patterns.

119634 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
119635 characters (for example, single-byte as opposed to multi-byte characters in  
119636 arguments and input files) and the behavior of character classes within bracketed  
119637 filename patterns (for example, "'[:lower:]'\*").

#### 119638 *LC\_MESSAGES*

119639 Determine the locale that should be used to affect the format and contents of  
119640 diagnostic messages written to standard error, and informative messages written  
119641 to standard output.

- 119642 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 119643 **ASYNCHRONOUS EVENTS**
- 119644 Default.
- 119645 **STDOUT**
- 119646 Not used.
- 119647 **STDERR**
- 119648 The standard error shall be used only for diagnostic messages.
- 119649 **OUTPUT FILES**
- 119650 The output files (which may be on other systems) are copies of the input files.
- 119651 If **-m** is used, mail files are modified.
- 119652 **EXTENDED DESCRIPTION**
- 119653 None.
- 119654 **EXIT STATUS**
- 119655 The following exit values shall be returned:
- 119656 0 Successful completion.
- 119657 >0 An error occurred.
- 119658 **CONSEQUENCES OF ERRORS**
- 119659 Default.
- 119660 **APPLICATION USAGE**
- 119661 This utility is part of the UUCP Utilities option and need not be supported by all  
119662 implementations.
- 119663 The domain of remotely accessible files can (and for obvious security reasons usually should) be  
119664 severely restricted.
- 119665 Note that the '!' character in addresses has to be escaped when using *cs*h as a command  
119666 interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary,  
119667 but may be used.
- 119668 As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate  
119669 system. On an internationalized system, this is done under the control of local settings of  
119670 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
119671 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
119672 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
119673 need not be supported on non-internationalized systems.
- 119674 **EXAMPLES**
- 119675 None.
- 119676 **RATIONALE**
- 119677 None.
- 119678 **FUTURE DIRECTIONS**
- 119679 If this utility is directed to create a new directory entry that contains any bytes that have the  
119680 encoded value of a <newline> character, implementations are encouraged to treat this as an  
119681 error. A future version of this standard may require implementations to treat this as an error.

119682 **SEE ALSO**119683 *mailx, uuencode, uustat, uux*

119684 XBD Chapter 8 (on page 167), Chapter 11 (on page 199), Section 12.2 (on page 215)

119685 **CHANGE HISTORY**

119686 First released in Issue 2.

119687 **Issue 6**119688 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.119689 The UN margin codes and associated shading are removed from the *-C*, *-f*, *-j*, *-n*, and *-r*  
119690 options in response to The Open Group Base Resolution bwg2001-003.119691 **Issue 7**

119692 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

119693 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

119694 **Issue 8**119695 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
119696 filenames containing any bytes that have the encoded value of a <newline> character.119697 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.119698 Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

119699 **NAME**

119700 uudecode — decode a binary file

119701 **SYNOPSIS**119702 uudecode [-o *outfile*] [*file*]119703 **DESCRIPTION**

119704 The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data  
 119705 created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data  
 119706 compatible with one of the formats specified in *uuencode*, and determine the pathname for the  
 119707 output file from the *-o* option if given, otherwise from the input data. If the pathname for the  
 119708 output file is either of the magic cookies *-* or */dev/stdout*, *uudecode* shall write the decoded file to  
 119709 standard output, otherwise it shall attempt to create or overwrite the file named by the  
 119710 pathname. The file access permission bits and contents for the file to be produced shall be  
 119711 contained in the input data. The mode bits of the created file (other than standard output) shall  
 119712 be set from the file access permission bits contained in the data; that is, other attributes of the  
 119713 mode, including the file mode creation mask (see *umask*), shall not affect the file being produced.  
 119714 If either of the *op* characters '+' and '-' (see *chmod*) are specified in symbolic mode, the initial  
 119715 mode on which those operations are based is unspecified.

119716 If the pathname of the file resolves to an existing file and the user does not have write  
 119717 permission on that file, *uudecode* shall terminate with an error. If the pathname of the file resolves  
 119718 to an existing file and the user has write permission on that file, the existing file shall be  
 119719 overwritten and, if possible, the mode bits of the file (other than standard output) shall be set as  
 119720 described above; if the mode bits cannot be set, *uudecode* shall not treat this as an error.

119721 If the input data was produced by *uuencode* on a system with a different number of bits per byte  
 119722 than on the target system, the results of *uudecode* are unspecified.

119723 **OPTIONS**119724 The *uudecode* utility shall conform to XBD [Section 12.2](#) (on page 215).

119725 The following option shall be supported by the implementation:

119726 *-o outfile* A pathname of a file that shall be used instead of any pathname contained in the  
 119727 input data. Specifying an *outfile* option-argument of *-* or */dev/stdout* shall indicate  
 119728 standard output.

119729 **OPERANDS**

119730 The following operand shall be supported:

119731 *file* The pathname of a file containing the output of *uuencode*.119732 **STDIN**

119733 See the INPUT FILES section.

119734 **INPUT FILES**119735 The input files shall be files containing the output of *uuencode*.119736 **ENVIRONMENT VARIABLES**119737 The following environment variables shall affect the execution of *uudecode*:

119738 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 119739 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 119740 variables used to determine the values of locale categories.)

119741 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 119742 internationalization variables.

- 119743 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
- 119744
- 119745
- 119746 *LC\_MESSAGES*
- 119747 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 119748
- 119749 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 119750 **ASYNCHRONOUS EVENTS**
- 119751 Default.
- 119752 **STDOUT**
- 119753 If the pathname specified for the output file is `-` or `/dev/stdout`, the standard output shall be in the same format as the file originally encoded by *uuencode*. Otherwise, the standard output shall not be used.
- 119754
- 119755
- 119756 **STDERR**
- 119757 The standard error shall be used only for diagnostic messages.
- 119758 **OUTPUT FILES**
- 119759 The output file shall be in the same format as the file originally encoded by *uuencode*.
- 119760 **EXTENDED DESCRIPTION**
- 119761 None.
- 119762 **EXIT STATUS**
- 119763 The following exit values shall be returned:
- 119764 0 Successful completion.
- 119765 >0 An error occurred.
- 119766 **CONSEQUENCES OF ERRORS**
- 119767 Default.
- 119768 **APPLICATION USAGE**
- 119769 The user who is invoking *uudecode* must have write permission on any file being created.
- 119770 The output of *uuencode* is essentially an encoded bit stream that is not cognizant of byte boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source, if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only data that is meaningful for such a transfer between architectures is generally character data.
- 119771
- 119772
- 119773
- 119774 In order to create an output file named `-`, it needs to be specified using an alternative pathname, for example, `-o ./-`, since `-` alone is considered a magic cookie by *uudecode*. Likewise, in order to write to an output file named `/dev/stdout` it also needs to be specified as, for example, `-o ///dev/stdout`.
- 119775
- 119776
- 119777
- 119778 **EXAMPLES**
- 119779 None.
- 119780 **RATIONALE**
- 119781 Input files are not necessarily text files, as stated by an early proposal. Although the *uuencode* output is a text file, that output could have been wrapped within another file or mail message that is not a text file.
- 119782
- 119783
- 119784 The `-o` option is not historical practice, but was added at the request of WG15 so that the user could override the target pathname without having to edit the input data itself.
- 119785

119786 In early drafts, the `[-o outfile]` option-argument allowed the use of `-` to mean standard output.  
119787 The standard developers did not wish to overload the meaning of `-` in this manner, resulting in  
119788 previous versions only using `/dev/stdout` for this purpose. POSIX.1-2024 now allows it as most  
119789 implementations were already supporting `-` as an extension. The file `/dev/stdout` exists as a  
119790 special file on most modern systems. However, the `/dev/stdout` syntax in *uudecode* does not refer  
119791 to a new file. It is just a magic cookie to specify standard output.

#### 119792 FUTURE DIRECTIONS

119793 If this utility is directed to create a new directory entry that contains any bytes that have the  
119794 encoded value of a `<newline>` character, implementations are encouraged to treat this as an  
119795 error. A future version of this standard may require implementations to treat this as an error.

#### 119796 SEE ALSO

119797 [\*chmod\*](#), [\*umask\*](#), [\*uuencode\*](#)

119798 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 119799 CHANGE HISTORY

119800 First released in Issue 4.

#### 119801 Issue 6

119802 This utility is marked as part of the User Portability Utilities option.

119803 The `-o outfile` option is added, as specified in the IEEE P1003.2b draft standard.

119804 The normative text is reworded to avoid use of the term “must” for application requirements.

119805 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/35 is applied, clarifying in the  
119806 DESCRIPTION that the initial mode used if either of the *op* characters is '+' or '-' is  
119807 unspecified.

#### 119808 Issue 7

119809 The *uudecode* utility is moved from the User Portability Utilities option to the Base. User  
119810 Portability Utilities is now an option for interactive utilities.

119811 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

119812 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0201 [635] is applied.

#### 119813 Issue 8

119814 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
119815 filenames containing any bytes that have the encoded value of a `<newline>` character.

119816 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

119817 Austin Group Defect 1544 is applied, changing the `-o` option to require that an option-argument  
119818 of `-` is treated as meaning standard output.

119819 **NAME**

119820 uuencode — encode a binary file

119821 **SYNOPSIS**119822 uuencode [-m] [*file*] *decode\_pathname*119823 **DESCRIPTION**

119824 The *uuencode* utility shall write an encoded version of the named input file, or standard input if  
 119825 no *file* is specified, to standard output. The output shall be encoded using one of the algorithms  
 119826 described in the STDOUT section and shall include the file access permission bits (in *chmod* octal  
 119827 or symbolic notation) of the input file and the *decode\_pathname*, for re-creation of the file on  
 119828 another system that conforms to this volume of POSIX.1-2024.

119829 **OPTIONS**119830 The *uuencode* utility shall conform to XBD [Section 12.2](#) (on page 215).

119831 The following option shall be supported by the implementation:

119832 **-m** Encode the output using the MIME Base64 algorithm described in STDOUT. If **-m**  
 119833 is not specified, the historical algorithm described in STDOUT shall be used.

119834 **OPERANDS**

119835 The following operands shall be supported:

119836 *decode\_pathname*

119837 The pathname of the file into which the *uudecode* utility shall place the decoded  
 119838 file. Specifying a *decode\_pathname* operand of **-** or **/dev/stdout** shall indicate that  
 119839 *uudecode* is to use standard output. If there are characters in *decode\_pathname* that  
 119840 are not in the portable filename character set the results are unspecified.

119841 *file* A pathname of the file to be encoded.119842 **STDIN**

119843 See the INPUT FILES section.

119844 **INPUT FILES**

119845 Input files can be files of any type.

119846 **ENVIRONMENT VARIABLES**119847 The following environment variables shall affect the execution of *uuencode*:

119848 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 119849 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 119850 variables used to determine the values of locale categories.)

119851 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 119852 internationalization variables.

119853 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 119854 characters (for example, single-byte as opposed to multi-byte characters in  
 119855 arguments and input files).

119856 **LC\_MESSAGES**

119857 Determine the locale that should be used to affect the format and contents of  
 119858 diagnostic messages written to standard error.

119859 **XSIX** **NLSPATH** Determine the location of messages objects and message catalogs.

119860 **ASYNCHRONOUS EVENTS**

119861 Default.

119862 **STDOUT**

119863 **uuencode Base64 Algorithm**

119864 The standard output shall be a text file (encoded in the character set of the current locale) that  
119865 begins with the line:

119866 "begin-base64Δ%sΔ\n", <mode>, <decode\_pathname>

119867 and ends with the line:

119868 "====\n"

119869 In both cases, the lines shall have no preceding or trailing <blank> characters.

119870 The encoding process represents 24-bit groups of input bits as output strings of four encoded  
119871 characters. Proceeding from left to right, a 24-bit input group shall be formed by concatenating  
119872 three 8-bit input groups. Each 24-bit input group then shall be treated as four concatenated 6-bit  
119873 groups, each of which shall be translated into a single digit in the Base64 alphabet. When  
119874 encoding a bit stream via the Base64 encoding, the bit stream shall be presumed to be ordered  
119875 with the most-significant bit first. That is, the first bit in the stream shall be the high-order bit in  
119876 the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit  
119877 group is used as an index into an array of 64 printable characters, as shown in Table 3-22.

119878 **Table 3-22 uuencode Base64 Values**

| Value | Encoding | Value | Encoding | Value | Encoding | Value | Encoding |
|-------|----------|-------|----------|-------|----------|-------|----------|
| 0     | A        | 17    | R        | 34    | i        | 51    | z        |
| 1     | B        | 18    | S        | 35    | j        | 52    | 0        |
| 2     | C        | 19    | T        | 36    | k        | 53    | 1        |
| 3     | D        | 20    | U        | 37    | l        | 54    | 2        |
| 4     | E        | 21    | V        | 38    | m        | 55    | 3        |
| 5     | F        | 22    | W        | 39    | n        | 56    | 4        |
| 6     | G        | 23    | X        | 40    | o        | 57    | 5        |
| 7     | H        | 24    | Y        | 41    | p        | 58    | 6        |
| 8     | I        | 25    | Z        | 42    | q        | 59    | 7        |
| 9     | J        | 26    | a        | 43    | r        | 60    | 8        |
| 10    | K        | 27    | b        | 44    | s        | 61    | 9        |
| 11    | L        | 28    | c        | 45    | t        | 62    | +        |
| 12    | M        | 29    | d        | 46    | u        | 63    | /        |
| 13    | N        | 30    | e        | 47    | v        | (pad) | =        |
| 14    | O        | 31    | f        | 48    | w        |       |          |
| 15    | P        | 32    | g        | 49    | x        |       |          |
| 16    | Q        | 33    | h        | 50    | y        |       |          |

119897 The character referenced by the index shall be placed in the output string.

119898 The output stream (encoded bytes) shall be represented in lines of no more than 76 characters  
119899 each. All line breaks or other characters not found in the table shall be ignored by decoding  
119900 software (see *uudecode*).

119901 Special processing shall be performed if fewer than 24 bits are available at the end of a message  
119902 or encapsulated part of a message. A full encoding quantum shall always be completed at the



119903 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall  
 119904 be added (on the right) to form an integral number of 6-bit groups. Output character positions  
 119905 that are not required to represent actual input data shall be set to the character '='. Since all  
 119906 Base64 input is an integral number of octets, only the following cases can arise:

- 119907 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit  
 119908 of encoded output shall be an integral multiple of 4 characters with no '=' padding.
- 119909 2. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded  
 119910 output shall be three characters followed by one '=' padding character.
- 119911 3. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded  
 119912 output shall be two characters followed by two '=' padding characters.

119913 A terminating "====" evaluates to nothing and denotes the end of the encoded data.

### 119914 uuencode Historical Algorithm

119915 The standard output shall be a text file (encoded in the character set of the current locale) that  
 119916 begins with the line:

```
119917 "beginΔ%sΔ%s\n" <mode>, <decode_pathname>
```

119918 and ends with the line:

```
119919 "end\n"
```

119920 In both cases, the lines shall have no preceding or trailing <blank> characters.

119921 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input  
 119922 and writes four characters of output by splitting the input at six-bit intervals into four octets,  
 119923 containing data in the lower six bits only. These octets shall be converted to characters in the  
 119924 ISO/IEC 646:1991 standard encoded character set by adding a value of 0x20 to each octet, so  
 119925 that each octet is in the range [0x20,0x5f], and then optionally replacing any 0x20 octets with  
 119926 0x60. If necessary, these characters shall then be translated into the corresponding character  
 119927 codes for the codeset in use in the current locale. For example, the octet 0x41, representing 'A',  
 119928 would be translated to 'A' in the current codeset, such as 0xc1 if it were EBCDIC; the octet 0x20,  
 119929 representing <space>, would optionally be replaced with 0x60, representing '`', and then  
 119930 translated to either <space> (0x40 if EBCDIC) or '`' (0x79 if EBCDIC), respectively.

119931 Where the bits of two octets are combined, the least significant bits of the first octet shall be  
 119932 shifted left and combined with the most significant bits of the second octet shifted right. Thus  
 119933 the three octets *A*, *B*, *C* shall be converted into the four octets:

```
119934 0x20 + (( A >> 2 ) & 0x3F)
119935 0x20 + (((A << 4) | ((B >> 4) & 0xF)) & 0x3F)
119936 0x20 + (((B << 2) | ((C >> 6) & 0x3)) & 0x3F)
119937 0x20 + (( C ) & 0x3F)
```

119938 before any replacement of 0x20 with 0x60 and translation into the local character set.

119939 Each encoded line shall contain a length character, equal to the number of characters to be  
 119940 decoded plus 0x20 translated to the local character set as described above, followed by between  
 119941 1 and 45, inclusive, encoded characters. The last encoded line, or the **begin** line if the input is  
 119942 empty, shall be followed by a line containing only a <space> or '`' character before the  
 119943 terminating <newline>.

119944 **STDERR**

119945 The standard error shall be used only for diagnostic messages.

119946 **OUTPUT FILES**

119947 None.

119948 **EXTENDED DESCRIPTION**

119949 None.

119950 **EXIT STATUS**

119951 The following exit values shall be returned:

119952 0 Successful completion.

119953 >0 An error occurred.

119954 **CONSEQUENCES OF ERRORS**

119955 Default.

119956 **APPLICATION USAGE**

119957 The file is expanded by 35 percent (each three octets become four, plus control information)  
119958 causing it to take longer to transmit.

119959 Since this utility is intended to create files to be used for data interchange between systems with  
119960 possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991  
119961 standard was chosen for a midpoint in the algorithm as a known reference point. The output  
119962 from *uuencode* is a text file on the local system. If the output were in the ISO/IEC 646:1991  
119963 standard codeset, it might not be a text file (at least because the <newline> characters might not  
119964 match), and the goal of creating a text file would be defeated. If this text file was then carried to  
119965 another machine with the same codeset, it would be perfectly compatible with that system's  
119966 *uudecode*. If it was transmitted over a mail system or sent to a machine with a different codeset,  
119967 it is assumed that, as for every other text file, some translation mechanism would convert it (by  
119968 the time it reached a user on the other system) into an appropriate codeset. This translation only  
119969 makes sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard  
119970 representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives,  
119971 intermixed with other text files in the same codeset.

119972 Since *uudecode* treats a *decode\_pathname* of `-` to mean decode to standard output, in order to  
119973 specify that a file named `-` is to be created, *decode\_pathname* should be specified using an  
119974 alternative pathname, for example `./-`. Likewise, in order to specify that a file with the  
119975 pathname `/dev/stdout` is to be written, *decode\_pathname* should be specified as, for example,  
119976 `///dev/stdout`.

119977 **EXAMPLES**

119978 None.

119979 **RATIONALE**

119980 A new algorithm was added at the request of the international community to parallel work in  
119981 RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding  
119982 is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A  
119983 65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented  
119984 per printable character. (The extra 65th character, `'`, is used to signify a special processing  
119985 function.)

119986 This subset has the important property that it is represented identically in all versions of the  
119987 ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also  
119988 represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not  
119989 share this property, which is the reason that a second algorithm was added to the ISO POSIX-2

- 119990 standard.
- 119991 The string "====" was used for the termination instead of the end used in the original format  
119992 because the latter is a string that could be valid encoded input.
- 119993 In an early draft, the `-m` option was named `-b` (for Base64), but it was renamed to reflect its  
119994 relationship to the RFC 2045. A `-u` was also present to invoke the default algorithm, but since  
119995 this was not historical practice, it was omitted as being unnecessary.
- 119996 See the RATIONALE section in *uudecode* for the derivation of the `/dev/stdout` symbol.
- 119997 Historically the encoding used only octets in the range [0x20,0x5f], and thus the encoded lines  
119998 could contain trailing spaces, which were at risk of being stripped by whatever transport  
119999 method was used to send the file. To avoid this problem some implementations use 0x60  
120000 instead of 0x20, resulting in ' ` ' characters instead of spaces in the output, and implementations  
120001 are encouraged to do this.
- 120002 **FUTURE DIRECTIONS**
- 120003 None.
- 120004 **SEE ALSO**
- 120005 *chmod, mailx, uudecode*
- 120006 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)
- 120007 **CHANGE HISTORY**
- 120008 First released in Issue 4.
- 120009 **Issue 6**
- 120010 This utility is marked as part of the User Portability Utilities option.
- 120011 The Base64 algorithm and the ability to output to `/dev/stdout` are added as specified in the  
120012 IEEE P1003.2b draft standard.
- 120013 **Issue 7**
- 120014 The *uuencode* utility is moved from the User Portability Utilities option to the Base. User  
120015 Portability Utilities is now an option for interactive utilities.
- 120016 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 120017 **Issue 8**
- 120018 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.
- 120019 Austin Group Defect 1140 is applied, changing the description of the *uuencode* historical  
120020 algorithm.
- 120021 Austin Group Defect 1544 is applied, changing the OPERANDS and APPLICATION USAGE  
120022 sections.

120023 **NAME**

120024 uustat — uucp status enquiry and job control

120025 **SYNOPSIS**120026 UU uustat [-q|-k *jobid*|-r *jobid*]120027 uustat [-s *system*] [-u *user*]120028 **DESCRIPTION**120029 The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or  
120030 provide general status on *uucp* connections to other systems.120031 When no options are given, *uustat* shall write to standard output the status of all *uucp* requests  
120032 issued by the current user.120033 Typical implementations of this utility require a communications line configured to use XBD  
120034 [Chapter 11](#) (on page 199), but other communications means may be used. On systems where  
120035 there are no available communications means (either temporarily or permanently), this utility  
120036 shall write an error message describing the problem and exit with a non-zero exit status.120037 **OPTIONS**120038 The *uustat* utility shall conform to XBD [Section 12.2](#) (on page 215).

120039 The following options shall be supported:

120040 **-q** Write the jobs queued for each machine.120041 **-k *jobid*** Kill the *uucp* request whose job identification is *jobid*. The application shall ensure  
120042 that the killed *uucp* request belongs to the person invoking *uustat* unless that user  
120043 has appropriate privileges.120044 **-r *jobid*** Rejuvenate *jobid*. The files associated with *jobid* are touched so that their  
120045 modification time is set to the current time. This prevents the cleanup program  
120046 from deleting the job until the jobs modification time reaches the limit imposed by  
120047 the program.120048 **-s *system*** Write the status of all *uucp* requests for remote system *system*.120049 **-u *user*** Write the status of all *uucp* requests issued by *user*.120050 **OPERANDS**

120051 None.

120052 **STDIN**

120053 Not used.

120054 **INPUT FILES**

120055 None.

120056 **ENVIRONMENT VARIABLES**120057 The following environment variables shall affect the execution of *uustat*:120058 **LANG** Provide a default value for the internationalization variables that are unset or null.  
120059 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
120060 variables used to determine the values of locale categories.)120061 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
120062 internationalization variables.

- 120063 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 120064
- 120065
- 120066 *LC\_MESSAGES*
- 120067 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output.
- 120068
- 120069
- 120070 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 120071 **ASYNCHRONOUS EVENTS**
- 120072 Default.
- 120073 **STDOUT**
- 120074 The standard output shall consist of information about each job selected, in an unspecified format. The information shall include at least the job ID, the user ID or name, and the remote system name.
- 120075
- 120076
- 120077 **STDERR**
- 120078 The standard error shall be used only for diagnostic messages.
- 120079 **OUTPUT FILES**
- 120080 None.
- 120081 **EXTENDED DESCRIPTION**
- 120082 None.
- 120083 **EXIT STATUS**
- 120084 The following exit values shall be returned:
- 120085 0 Successful completion.
- 120086 >0 An error occurred.
- 120087 **CONSEQUENCES OF ERRORS**
- 120088 Default.
- 120089 **APPLICATION USAGE**
- 120090 This utility is part of the UUCP Utilities option and need not be supported by all implementations.
- 120091
- 120092 **EXAMPLES**
- 120093 None.
- 120094 **RATIONALE**
- 120095 None.
- 120096 **FUTURE DIRECTIONS**
- 120097 None.
- 120098 **SEE ALSO**
- 120099 *uucp*
- 120100 XBD [Chapter 8](#) (on page 167), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 215)
- 120101 **CHANGE HISTORY**
- 120102 First released in Issue 2.

120103 **Issue 6**

120104 The normative text is reworded to avoid use of the term “must” for application requirements.

120105 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

120106 The UN margin code and associated shading are removed from the *-q* option in response to The  
120107 Open Group Base Resolution bwg2001-003.

120108 **Issue 7**

120109 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

120110 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

120111 **Issue 8**

120112 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

120113 Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

120114 **NAME**

120115 uux — remote command execution

120116 **SYNOPSIS**120117 UU `uux [-jnp] command-string`120118 **DESCRIPTION**

120119 The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see  
 120120 [Section 2.9](#), on page 2499) on a specified system, and then send the standard output of the  
 120121 command to a file on a specified system. Only the first command of a pipeline can have a *system-*  
 120122 *name!* prefix. All other commands in the pipeline shall be executed on the system of the first  
 120123 command.

120124 The following restrictions are applicable to the shell pipeline processed by *uux*:

120125 • In gathering files from different systems, pathname expansion shall not be performed by  
 120126 *uux*. Thus, a request such as:

120127 `uux "c17 remsys!~/*.c"`

120128 would attempt to copy the file named literally `*.c` to the local system.

120129 • The redirection operators "`>>`", "`<<`", "`>|`", and "`>&`" shall not be accepted. Any use of  
 120130 these redirection operators shall cause this utility to write an error message describing the  
 120131 problem and exit with a non-zero exit status.

120132 • The reserved word `!` cannot be used at the head of the pipeline to modify the exit status.  
 120133 (See the *command-string* operand description below.)

120134 • Alias substitution shall not be performed.

120135 A filename can be specified as for *uucp*; it can be an absolute pathname, a pathname preceded by  
 120136 *~name* (which is replaced by the corresponding login directory), a pathname specified as *~/dest*  
 120137 (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is  
 120138 implementation-defined), or a simple filename (which is prefixed by *uux* with the current  
 120139 directory). See *uucp* for the details.

120140 The execution of commands on remote systems shall take place in an execution directory known  
 120141 to the *uucp* system. All files required for the execution shall be put into this directory unless they  
 120142 already reside on that machine. Therefore, the application shall ensure that non-local filenames  
 120143 (without path or machine reference) are unique within the *uux* request.

120144 The *uux* utility shall attempt to get all files to the execution system. For files that are output files,  
 120145 the application shall ensure that the filename is escaped using parentheses.

120146 The remote system shall notify the user by mail if the requested command on the remote system  
 120147 was disallowed or the files were not accessible. This notification can be turned off by the `-n`  
 120148 option.

120149 Typical implementations of this utility require a communications line configured to use XBD  
 120150 [Chapter 11](#) (on page 199), but other communications means may be used. On systems where  
 120151 there are no available communications means (either temporarily or permanently), this utility  
 120152 shall write an error message describing the problem and exit with a non-zero exit status.

120153 The *uux* utility cannot guarantee support for all character encodings in all circumstances. For  
 120154 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
 120155 filenames need not be portable to non-internationalized systems, and so on. Under these  
 120156 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
 120157 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used

120158 and that only characters defined in the portable filename character set be used for naming files.

#### 120159 OPTIONS

120160 The *uux* utility shall conform to XBD [Section 12.2](#) (on page 215).

120161 The following options shall be supported:

120162 **-j** Write the job identification string to standard output. This job identification can be  
120163 used by *uustat* to obtain the status or terminate a job.

120164 **-n** Do not notify the user if the command fails.

120165 **-p** Make the standard input to *uux* the standard input to the *command-string*.

#### 120166 OPERANDS

120167 The following operand shall be supported:

120168 *command-string*

120169 A string made up of one or more arguments that are similar to normal command  
120170 arguments, except that the command and any filenames can be prefixed by *system-*  
120171 *name!*. A null *system-name* shall be interpreted as the local system.

#### 120172 STDIN

120173 The standard input shall not be used unless the **-** or **-p** option is specified; in those cases, the  
120174 standard input shall be made the standard input of the *command-string*.

#### 120175 INPUT FILES

120176 Input files shall be selected according to the contents of *command-string*.

#### 120177 ENVIRONMENT VARIABLES

120178 The following environment variables shall affect the execution of *uux*:

120179 **LANG** Provide a default value for the internationalization variables that are unset or null.  
120180 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
120181 variables used to determine the values of locale categories.)

120182 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
120183 internationalization variables.

120184 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
120185 characters (for example, single-byte as opposed to multi-byte characters in  
120186 arguments).

120187 **LC\_MESSAGES**

120188 Determine the locale that should be used to affect the format and contents of  
120189 diagnostic messages written to standard error.

120190 **XS** **NLSPATH** Determine the location of messages objects and message catalogs.

#### 120191 ASYNCHRONOUS EVENTS

120192 Default.

#### 120193 STDOUT

120194 The standard output shall not be used unless the **-j** option is specified; in that case, the job  
120195 identification string shall be written to standard output in the following format:

120196 "%s\n", <*jobid*>



120197 **STDERR**

120198 The standard error shall be used only for diagnostic messages.

120199 **OUTPUT FILES**

120200 Output files shall be created or written, or both, according to the contents of *command-string*.

120201 If **-n** is not used, mail files shall be modified following any command or file-access failures on  
120202 the remote system.

120203 **EXTENDED DESCRIPTION**

120204 None.

120205 **EXIT STATUS**

120206 The following exit values shall be returned:

120207 0 Successful completion.

120208 >0 An error occurred.

120209 **CONSEQUENCES OF ERRORS**

120210 Default.

120211 **APPLICATION USAGE**

120212 This utility is part of the UUCP Utilities option and need not be supported by all  
120213 implementations.

120214 Note that, for security reasons, many installations limit the list of commands executable on  
120215 behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail  
120216 via *uux*.

120217 Any characters special to the command interpreter should be quoted either by quoting the entire  
120218 *command-string* or quoting the special characters as individual arguments.

120219 As noted in *uucp*, shell pattern matching notation characters appearing in pathnames are  
120220 expanded on the appropriate local system. This is done under the control of local settings of  
120221 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
120222 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
120223 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
120224 need not be supported on non-internationalized systems.

120225 **EXAMPLES**

120226 1. The following command gets **file1** from system **a** and **file2** from system **b**, executes *diff* on  
120227 the local system, and puts the results in **file.diff** in the local *PUBDIR* directory. (*PUBDIR*  
120228 is the *uucp* public directory on the local system.)

```
120229 uux "!diff a!/usr/file1 b!/a4/file2 >!~/file.diff"
```

120230 2. The following command fails because *uux* places all files copied to a system in the same  
120231 working directory. Although the files **xyz** are from two different systems, their filenames  
120232 are the same and conflict.

```
120233 uux "!diff a!/usr1/xyz b!/usr2/xyz >!~/xyz.diff"
```

120234 3. The following command succeeds (assuming *diff* is permitted on system **a**) because the  
120235 file local to system **a** is not copied to the working directory, and hence does not conflict  
120236 with the file from system **c**.

```
120237 uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"
```

- 120238 **RATIONALE**  
120239 None.
- 120240 **FUTURE DIRECTIONS**  
120241 None.
- 120242 **SEE ALSO**  
120243 [Chapter 2](#) (on page 2472), [uucp](#), [uuencode](#), [uustat](#)  
120244 [XBD Chapter 8](#) (on page 167), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 215)
- 120245 **CHANGE HISTORY**  
120246 First released in Issue 2.
- 120247 **Issue 6**  
120248 The obsolescent SYNOPSIS is removed.  
120249 The normative text is reworded to avoid use of the term “must” for application requirements.  
120250 The UN margin code and associated shading are removed from the -j option in response to The  
120251 Open Group Base Resolution bwg2001-003.
- 120252 **Issue 7**  
120253 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.
- 120254 **Issue 8**  
120255 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.  
120256 Austin Group Defect 1516 is applied, adding XSI shading to text relating to *NLSPATH*.

120257 **NAME**120258 val — validate SCCS files (**DEVELOPMENT**)120259 **SYNOPSIS**

```
120260 XSI val -
120261 val [-s] [-m name] [-r SID] [-y type] file...
```

120262 **DESCRIPTION**

120263 The *val* utility shall determine whether the specified *file* is an SCCS file meeting the  
 120264 characteristics specified by the options.

120265 **OPTIONS**

120266 The *val* utility shall conform to XBD [Section 12.2](#) (on page 215), except that the usage of the '-'  
 120267 operand is not strictly as intended by the guidelines (that is, reading options and operands from  
 120268 standard input).

120269 The following options shall be supported:

- 120270 **-m name** Specify a *name*, which is compared with the SCCS %M% keyword in *file*; see *get*.
- 120271 **-r SID** Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be  
 120272 made to determine whether the *SID* is ambiguous (for example, **-r 1** is ambiguous  
 120273 because it physically does not exist but implies 1.1, 1.2, and so on, which may exist)  
 120274 or invalid (for example, **-r 1.0** or **-r 1.1.0** are invalid because neither case can exist  
 120275 as a valid delta number). If the *SID* is valid and not ambiguous, a check shall be  
 120276 made to determine whether it actually exists.
- 120277 **-s** Silence the diagnostic message normally written to standard output for any error  
 120278 that is detected while processing each named file on a given command line.
- 120279 **-y type** Specify a *type*, which shall be compared with the SCCS %Y% keyword in *file*; see  
 120280 *get*.

120281 **OPERANDS**

120282 The following operands shall be supported:

- 120283 *file* A pathname of an existing SCCS file. If exactly one *file* operand appears, and it is  
 120284 '-', the standard input shall be read: each line shall be independently processed  
 120285 as if it were a command line argument list. (However, the line is not subjected to  
 120286 any of the shell word expansions, such as parameter expansion or quote removal.)

120287 **STDIN**

120288 The standard input shall be a text file used only when the *file* operand is specified as '-'.

120289 **INPUT FILES**

120290 Any SCCS files processed shall be files of an unspecified format.

120291 **ENVIRONMENT VARIABLES**

120292 The following environment variables shall affect the execution of *val*:

- 120293 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 120294 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 120295 variables used to determine the values of locale categories.)
- 120296 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 120297 internationalization variables.

120298 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 120299 characters (for example, single-byte as opposed to multi-byte characters in  
 120300 arguments and input files).

120301 **LC\_MESSAGES**  
 120302 Determine the locale that should be used to affect the format and contents of  
 120303 diagnostic messages written to standard error, and informative messages written  
 120304 to standard output.

120305 **NLSPATH** Determine the location of messages objects and message catalogs.

120306 **ASYNCHRONOUS EVENTS**  
 120307 Default.

120308 **STDOUT**  
 120309 The standard output shall consist of informative messages about either:  
 120310 1. Each file processed  
 120311 2. Each command line read from standard input

120312 If the standard input is not used, for each *file* operand yielding a discrepancy, the output line  
 120313 shall have the following format:  
 120314 "%s: %s\n", <pathname>, <unspecified string>

120315 If the standard input is used, for each input line yielding a discrepancy, the output shall have the  
 120316 following format:  
 120317 "%s\n\n %s: %s\n", <input>, <pathname>, <unspecified string>

120318 where <input> is the input line minus its terminating <newline>.

120319 **STDERR**  
 120320 Not used.

120321 **OUTPUT FILES**  
 120322 None.

120323 **EXTENDED DESCRIPTION**  
 120324 None.

120325 **EXIT STATUS**  
 120326 The 8-bit code returned by *val* shall be a disjunction of the possible errors; that is, it can be  
 120327 interpreted as a bit string where set bits are interpreted as follows:

120328 0x80 = Missing file argument.  
 120329 0x40 = Unknown or duplicate option.  
 120330 0x20 = Corrupted SCCS file.  
 120331 0x10 = Cannot open file or file not SCCS.  
 120332 0x08 = *SID* is invalid or ambiguous.  
 120333 0x04 = *SID* does not exist.  
 120334 0x02 = %Y%, -y mismatch.  
 120335 0x01 = %M%, -m mismatch.

120336 Note that *val* can process two or more files on a given command line and can process multiple  
 120337 command lines (when reading the standard input). In these cases an aggregate code shall be  
 120338 returned: a logical OR of the codes generated for each command line and file processed.

120339 **CONSEQUENCES OF ERRORS**

120340 Default.

120341 **APPLICATION USAGE**

120342 Since the *val* exit status sets the 0x80 bit, shell applications checking "\$?" cannot tell if it  
 120343 terminated due to a missing file argument or receipt of a signal.

120344 **EXAMPLES**

120345 In a directory with three SCCS files—*s.x* (of *t* type `text`), *s.y*, and *s.z* (a corrupted file)—the  
 120346 following command could produce the output shown:

```
120347 val - <<EOF
120348 -y source s.x
120349 -m y s.y
120350 s.z
120351 EOF
120352 -y source s.x

120353 s.x: %Y%, -y mismatch
120354 s.z
120355 s.z: corrupted SCCS file
```

120356 **RATIONALE**

120357 None.

120358 **FUTURE DIRECTIONS**

120359 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 120360 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
 120361 format being used, implementations are encouraged to treat this as an error. A future version of  
 120362 this standard may require implementations to treat this as an error.

120363 **SEE ALSO**120364 *admin, delta, get, prs*

120365 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

120366 **CHANGE HISTORY**

120367 First released in Issue 2.

120368 **Issue 6**

120369 The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT  
 120370 STATUS.

120371 **Issue 7**

120372 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

120373 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0147 [416] and XCU/TC1-2008/0148  
 120374 [416] are applied.

120375 **Issue 8**

120376 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
 120377 directed to display a pathname that contains any bytes that have the encoded value of a  
 120378 `<newline>` character when `<newline>` is a terminator or separator in the output format being  
 120379 used.

120380 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

120381 **NAME**

120382 vi — screen-oriented (visual) display editor

120383 **SYNOPSIS**120384 UP `vi [-rR] [-c command] [-t tagstring] [-w size] [file...]`120385 **DESCRIPTION**120386 This utility shall be provided on systems that both support the User Portability Utilities option  
120387 and define the POSIX2\_CHAR\_TERM symbol. On other systems it is optional.120388 The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the  
120389 editor are described in POSIX.1-2024; see the line editor *ex* for additional editing capabilities  
120390 used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex* commands from  
120391 within *vi*.120392 This reference page uses the term *edit buffer* to describe the current working text. No specific  
120393 implementation is implied by this term. All editing changes are performed on the edit buffer,  
120394 and no changes to it shall affect any file until an editor command writes the file.120395 When using *vi*, the terminal screen acts as a window into the editing buffer. Changes made to  
120396 the editing buffer shall be reflected in the screen display; the position of the cursor on the screen  
120397 shall indicate the position within the editing buffer.120398 Certain terminals do not have all the capabilities necessary to support the complete *vi* definition.  
120399 When these commands cannot be supported on such terminals, this condition shall not produce  
120400 an error message such as “not an editor command” or report a syntax error. The implementation  
120401 may either accept the commands and produce results on the screen that are the result of an  
120402 unsuccessful attempt to meet the requirements of this volume of POSIX.1-2024 or report an error  
120403 describing the terminal-related deficiency.120404 **OPTIONS**120405 The *vi* utility shall conform to XBD [Section 12.2](#) (on page 215), except that '+' may be  
120406 recognized as an option delimiter as well as '-'.  
120407 The following options shall be supported:

120407 The following options shall be supported:

120408 `-c command` See the *ex* command description of the `-c` option.120409 `-r` See the *ex* command description of the `-r` option.120410 `-R` See the *ex* command description of the `-R` option.120411 `-t tagstring` See the *ex* command description of the `-t` option.120412 `-w size` See the *ex* command description of the `-w` option.120413 **OPERANDS**120414 See the OPERANDS section of the *ex* command for a description of the operands supported by  
120415 the *vi* command.120416 **STDIN**120417 If standard input is not a terminal device, the results are undefined. The standard input consists  
120418 of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.120419 If a read from the standard input returns an error, or if the editor detects an end-of-file condition  
120420 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

120421 **INPUT FILES**

120422 See the INPUT FILES section of the *ex* command for a description of the input files supported by  
120423 the *vi* command.

120424 **ENVIRONMENT VARIABLES**

120425 See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables  
120426 that affect the execution of the *vi* command.

120427 **ASYNCHRONOUS EVENTS**

120428 See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the  
120429 execution of the *vi* command.

120430 **STDOUT**

120431 If standard output is not a terminal device, undefined results occur.

120432 Standard output may be used for writing prompts to the user, for informational messages, and  
120433 for writing lines from the file.

120434 **STDERR**

120435 If standard output is not a terminal device, undefined results occur.

120436 The standard error shall be used only for diagnostic messages.

120437 **OUTPUT FILES**

120438 See the OUTPUT FILES section of the *ex* command for a description of the output files  
120439 supported by the *vi* command.

120440 **EXTENDED DESCRIPTION**

120441 If the terminal does not have the capabilities necessary to support an unspecified portion of the  
120442 *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after  
120443 initialization, *vi* shall be in command mode; text input mode can be entered by one of several  
120444 commands used to insert or change text. In text input mode, <ESC> can be used to return to  
120445 command mode; other uses of <ESC> are described later in this section; see [Terminate  
120446 Command or Input Mode](#) (on page 3536).

120447 **Initialization in *ex* and *vi***

120448 See [Initialization in \*ex\* and \*vi\*](#) (on page 2842) for a description of *ex* and *vi* initialization for the *vi*  
120449 utility.

120450 **Command Descriptions in *vi***

120451 The following symbols are used in this reference page to represent arguments to commands.

120452 *buffer* See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility;  
120453 see [Command Descriptions in \*ex\*](#) (on page 2852).

120454 In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*]  
120455 preceding the command name, they can be specified in either order.

120456 *count* A positive integer used as an optional argument to most commands, either to give a  
120457 repeat count or as a size. This argument is optional and shall default to 1 unless  
120458 otherwise specified.

120459 The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R,  
120460 <control>-], %, &, ^, D, m, M, Q, u, U, and ZZ do not have *count* as an optional  
120461 argument. Regardless, it shall not be an error to specify a *count* to these commands, and  
120462 any specified *count* shall be ignored.

120463 *motion* An optional trailing argument used by the **!**, **<**, **>**, **c**, **d**, and **y** commands, which is used  
 120464 to indicate the region of text that shall be affected by the command. The motion can be  
 120465 either one of the command characters repeated or one of several other *vi* commands  
 120466 (listed in the following table). Each of the applicable commands specifies the region of  
 120467 text matched by repeating the command; each command that can be used as a motion  
 120468 command specifies the region of text it affects.

120469 Commands that take *motion* arguments operate on either lines or characters, depending  
 120470 on the circumstances. When operating on lines, all lines that fall partially or wholly  
 120471 within the text region specified for the command shall be affected. When operating on  
 120472 characters, only the exact characters in the specified text region shall be affected. Each  
 120473 motion command specifies this individually.

120474 When commands that may be motion commands are not used as motion commands,  
 120475 they shall set the current position to the current line and column as specified.

120476 The following commands shall be valid cursor motion commands:

|        |                   |   |    |   |   |   |
|--------|-------------------|---|----|---|---|---|
| 120477 | <apostrophe>      | ( | -  | j | H |   |
| 120478 | <carriage-return> | ) | \$ | k | L |   |
| 120479 | <comma>           | [ | [  | % | l | M |
| 120480 | <control>-H       | ] | ]  | _ | n | N |
| 120481 | <control>-N       | { | ;  | t | T |   |
| 120482 | <control>-P       | } | ?  | w | W |   |
| 120483 | <grave-accent>    | ^ | b  | B |   |   |
| 120484 | <newline>         | + | e  | E |   |   |
| 120485 | <space>           |   | f  | F |   |   |
| 120486 | <zero>            | / | h  | G |   |   |

120487 Any *count* that is specified to a command that has an associated motion command shall  
 120488 be applied to the motion command. If a *count* is applied to both the command and its  
 120489 associated motion command, the effect shall be multiplicative.

120490 The following symbols are used in this section to specify locations in the edit buffer:

120491 *current character*

120492 The character that is currently indicated by the cursor.

120493 *end of a line*

120494 The point located between the last non-<newline> (if any) and the terminating  
 120495 <newline> of a line. For an empty line, this location coincides with the beginning of the  
 120496 line.

120497 *end of the edit buffer*

120498 The location corresponding to the end of the last line in the edit buffer.

120499 The following symbols are used in this section to specify command actions:

120500 *bigword* In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

- 120501 1. A maximal sequence of non-<blank> characters preceded and followed by  
 120502 <blank> characters or the beginning or end of a line or the edit buffer
- 120503 2. One or more sequential blank lines
- 120504 3. The first character in the edit buffer



- 120505 4. The last non-`<newline>` in the edit buffer
- 120506 *word* In the POSIX locale, *vi* shall recognize five kinds of words:
- 120507 1. A maximal sequence of letters, digits, and underscores, delimited at both ends
- 120508 by:
- 120509 — Characters other than letters, digits, or underscores
- 120510 — The beginning or end of a line
- 120511 — The beginning or end of the edit buffer
- 120512 2. A maximal sequence of characters other than letters, digits, underscores, or
- 120513 `<blank>` characters, delimited at both ends by:
- 120514 — A letter, digit, underscore
- 120515 — `<blank>` characters
- 120516 — The beginning or end of a line
- 120517 — The beginning or end of the edit buffer
- 120518 3. One or more sequential blank lines
- 120519 4. The first character in the edit buffer
- 120520 5. The last non-`<newline>` in the edit buffer
- 120521 *section boundary*
- 120522 A *section boundary* is one of the following:
- 120523 1. A line whose first character is a `<form-feed>`
- 120524 2. A line whose first character is an open curly brace (`' { '`)
- 120525 3. A line whose first character is a `<period>` and whose second and third characters
- 120526 match a two-character pair in the **sections** edit option (see *ex*)
- 120527 4. A line whose first character is a `<period>` and whose only other character
- 120528 matches the first character of a two-character pair in the **sections** edit option,
- 120529 where the second character of the two-character pair is a `<space>`
- 120530 5. The first line of the edit buffer
- 120531 6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a
- 120532 `]]` or `}` command; otherwise, the last non-`<newline>` of the last line of the edit
- 120533 buffer
- 120534 *paragraph boundary*
- 120535 A *paragraph boundary* is one of the following:
- 120536 1. A section boundary
- 120537 2. A line whose first character is a `<period>` and whose second and third characters
- 120538 match a two-character pair in the **paragraphs** edit option (see *ex*)
- 120539 3. A line whose first character is a `<period>` and whose only other character
- 120540 matches the first character of a two-character pair in the *paragraphs* edit option,
- 120541 where the second character of the two-character pair is a `<space>`

120542 4. One or more sequential blank lines

120543 *remembered search direction*

120544 See the description of *remembered search direction* in *ex*.

120545 *sentence boundary*

120546 A *sentence boundary* is one of the following:

- 120547 1. A paragraph boundary
- 120548 2. The first non-<blank> that occurs after a paragraph boundary
- 120549 3. The first non-<blank> that occurs after a <period> (' . '), <exclamation-mark>  
120550 (' ! '), or <question-mark> (' ? '), followed by two <space> characters or the end  
120551 of a line; any number of closing parenthesis (' ) '), closing brackets (' ] '),  
120552 double-quote (' " '), or single-quote (<apostrophe>) characters can appear  
120553 between the punctuation mark and the two <space> characters or end-of-line

120554 In the remainder of the description of the *vi* utility, the term “buffer line” refers to a line in the  
120555 edit buffer and the term “display line” refers to the line or lines on the display screen used to  
120556 display one buffer line. The term “current line” refers to a specific “buffer line”.

120557 If there are display lines on the screen for which there are no corresponding buffer lines because  
120558 they correspond to lines that would be after the end of the file, they shall be displayed as a single  
120559 <tilde> (' ~ ') character, plus the terminating <newline>.

120560 The last line of the screen shall be used to report errors or display informational messages. It  
120561 shall also be used to display the input for “line-oriented commands” (*/*, *?*, *:*, and *!*). When a line-  
120562 oriented command is executed, the editor shall enter text input mode on the last line on the  
120563 screen, using the respective command characters as prompt characters. (In the case of the *!*  
120564 command, the associated motion shall be entered by the user before the editor enters text input  
120565 mode.) The line entered by the user shall be terminated by a <newline>, a  
120566 non-<control>-V-escaped <carriage-return>, or unescaped <ESC>. It is unspecified if more  
120567 characters than require a display width minus one column number of screen columns can be  
120568 entered.

120569 If any command is executed that overwrites a portion of the screen other than the last line of the  
120570 screen (for example, the *ex* **suspend** or *!* commands), other than the *ex* **shell** command, the user  
120571 shall be prompted for a character before the screen is refreshed and the edit session continued.

120572 <tab> characters shall take up the number of columns on the screen set by the **tabstop** edit  
120573 option (see *ex*), unless there are less than that number of columns before the display margin that  
120574 will cause the displayed line to be folded; in this case, they shall only take up the number of  
120575 columns up to that boundary.

120576 The cursor shall be placed on the current line and relative to the current column as specified by  
120577 each command described in the following sections.

120578 In open mode, if the current line is not already displayed, then it shall be displayed.

120579 In visual mode, if the current line is not displayed, then the lines that are displayed shall be  
120580 expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be  
120581 displayed. If the screen is redrawn, no more than the number of display lines specified by the  
120582 value of the **window** edit option shall be displayed (unless the current line cannot be completely  
120583 displayed in the number of display lines specified by the **window** edit option) and the current  
120584 line shall be positioned as close to the center of the displayed lines as possible (within the  
120585 constraints imposed by the distance of the line from the beginning or end of the edit buffer). If  
120586 the current line is before the first line in the display and the screen is scrolled, an unspecified

120587 portion of the current line shall be placed on the first line of the display. If the current line is after  
 120588 the last line in the display and the screen is scrolled, an unspecified portion of the current line  
 120589 shall be placed on the last line of the display.

120590 In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into  
 120591 the lines at the bottom of the display that are available for its presentation, the editor may choose  
 120592 not to display any portion of the line. The lines of the display that do not contain text from the  
 120593 edit buffer for this reason shall each consist of a single '@' character.

120594 In visual mode, the editor may choose for unspecified reasons to not update lines in the display  
 120595 to correspond to the underlying edit buffer text. The lines of the display that do not correctly  
 120596 correspond to text from the edit buffer for this reason shall consist of a single '@' character (plus  
 120597 the terminating <newline>), and the <control>-R command shall cause the editor to update the  
 120598 screen to correctly represent the edit buffer.

120599 Open and visual mode commands that set the current column set it to a column position in the  
 120600 display, and not a character position in the line. In this case, however, the column position in the  
 120601 display shall be calculated for an infinite width display; for example, the column related to a  
 120602 character that is part of a line that has been folded onto additional screen lines is offset from the  
 120603 display line column where the buffer line begins, not from the beginning of a particular display  
 120604 line.

120605 The display cursor column in the display is based on the value of the current column, as follows,  
 120606 with each rule applied in turn:

- 120607 1. If the current column is after the last display line column used by the displayed line, the  
 120608 display cursor column shall be set to the last display line column occupied by the last  
 120609 non-<newline> in the current line; otherwise, the display cursor column shall be set to the  
 120610 current column.
- 120611 2. If the character of which some portion is displayed in the display line column specified by  
 120612 the display cursor column requires more than a single display line column:
  - 120613 a. If in text input mode, the display cursor column shall be adjusted to the first  
 120614 display line column in which any portion of that character is displayed.
  - 120615 b. Otherwise, the display cursor column shall be adjusted to the last display line  
 120616 column in which any portion of that character is displayed.

120617 The current column shall not be changed by these adjustments to the display cursor column.

120618 If an error occurs during the parsing or execution of a *vi* command:

- 120619 • The terminal shall be alerted. Execution of the *vi* command shall stop, and the cursor (for  
 120620 example, the current line and column) shall not be further modified.
- 120621 • Unless otherwise specified by the following command sections, it is unspecified whether  
 120622 an informational message shall be displayed.
- 120623 • Any partially entered *vi* command shall be discarded.
- 120624 • If the *vi* command resulted from a **map** expansion, all characters from that **map** expansion  
 120625 shall be discarded, except as otherwise specified by the **map** command (see *ex*).
- 120626 • If the *vi* command resulted from the execution of a buffer, no further commands caused by  
 120627 the execution of the buffer shall be executed.

120628 **Page Backwards**120629 *Synopsis:* [count] <control>-B120630 If in open mode, the <control>-B command shall behave identically to the **z** command.  
120631 Otherwise, if the current line is the first line of the edit buffer, it shall be an error.120632 If the **window** edit option is less than 3, display a screen where the last line of the display shall  
120633 be some portion of:120634 *(current first line) -1*

120635 otherwise, display a screen where the first line of the display shall be some portion of:

120636 *(current first line) - count x ((window edit option) -2)*120637 If this calculation would result in a line that is before the first line of the edit buffer, the first line  
120638 of the display shall display some portion of the first line of the edit buffer.120639 *Current line:* If no lines from the previous display remain on the screen, set to the last line of the  
120640 display; otherwise, set to *(line - the number of new lines displayed on this screen)*.120641 *Current column:* Set to non-<blank>.120642 **Scroll Forward**120643 *Synopsis:* [count] <control>-D

120644 If the current line is the last line of the edit buffer, it shall be an error.

120645 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D  
120646 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*  
120647 shall default to the value of the **scroll** edit option.120648 If in open mode, write lines starting with the line after the current line, until *count* lines or the  
120649 last line of the file have been written.120650 *Current line:* If the current line + *count* is past the last line of the edit buffer, set to the last line of  
120651 the edit buffer; otherwise, set to the current line + *count*.120652 *Current column:* Set to non-<blank>.120653 **Scroll Forward by Line**120654 *Synopsis:* [count] <control>-E

120655 Display the line count lines after the last line currently displayed.

120656 If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines  
120657 after the last line currently displayed, the last line of the display shall display some portion of  
120658 the last line of the edit buffer.120659 *Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first  
120660 line displayed.120661 *Current column:* Unchanged.

120662 **Page Forward**120663 *Synopsis:* `[count] <control>-F`120664 If in open mode, the `<control>-F` command shall behave identically to the `z` command.  
120665 Otherwise, if the current line is the last line of the edit buffer, it shall be an error.120666 If the **window** edit option is less than 3, display a screen where the first line of the display shall  
120667 be some portion of:120668 `(current last line) +1`

120669 otherwise, display a screen where the first line of the display shall be some portion of:

120670 `(current first line) + count x ((window edit option) -2)`120671 If this calculation would result in a line that is after the last line of the edit buffer, the last line of  
120672 the display shall display some portion of the last line of the edit buffer.120673 *Current line:* If no lines from the previous display remain on the screen, set to the first line of the  
120674 display; otherwise, set to `(line + the number of new lines displayed on this screen)`.120675 *Current column:* Set to non-`<blank>`.120676 **Display Information**120677 *Synopsis:* `<control>-G`120678 This command shall be equivalent to the `ex file` command.120679 **Move Cursor Backwards**120680 *Synopsis:* `[count] <control>-H`120681 `[count] h`120682 the current *erase* character (see `stty`)120683 If there are no characters before the current character on the current line, it shall be an error. If  
120684 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
120685 number of previous characters on the line.

120686 If used as a motion command:

- 120687 1. The text region shall be from the character before the starting cursor up to and including  
120688 the *count*th character before the starting cursor.
- 120689 2. Any text copied to a buffer shall be in character mode.

120690 If not used as a motion command:

120691 *Current line:* Unchanged.120692 *Current column:* Set to `(column - the number of columns occupied by count characters ending  
120693 with the previous current column)`.

120694 **Move Down**

120695 *Synopsis:*    [*count*] <newline>  
 120696            [*count*] <control>-J  
 120697            [*count*] <control>-M  
 120698            [*count*] <control>-N  
 120699            [*count*] j  
 120700            [*count*] <carriage-return>  
 120701            [*count*] +

120702 If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

120703 If used as a motion command:

- 120704     1. The text region shall include the starting line and the next *count* – 1 lines.
- 120705     2. Any text copied to a buffer shall be in line mode.

120706 If not used as a motion command:

120707 *Current line:* Set to *current line*+ *count*.

120708 *Current column:* Set to non-<blank> for the <carriage-return>, <control>-M, and + commands;  
 120709 otherwise, unchanged.

120710 **Clear and Redisplay**

120711 *Synopsis:*    <control>-L

120712 If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay  
 120713 the screen.

120714 *Current line:* Unchanged.

120715 *Current column:* Unchanged.

120716 **Move Up**

120717 *Synopsis:*    [*count*] <control>-P  
 120718            [*count*] k  
 120719            [*count*] –

120720 If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

120721 If used as a motion command:

- 120722     1. The text region shall include the starting line and the previous *count* lines.
- 120723     2. Any text copied to a buffer shall be in line mode.

120724 If not used as a motion command:

120725 *Current line:* Set to *current line* – *count*.

120726 *Current column:* Set to non-<blank> for the – command; otherwise, unchanged.

**120727 Redraw Screen**

120728 *Synopsis:*     <control>-R

120729     If any lines have been deleted from the display screen and flagged as deleted on the terminal  
120730     using the @ convention (see the beginning of the EXTENDED DESCRIPTION section), they shall  
120731     be redisplayed to match the contents of the edit buffer.

120732     It is unspecified whether lines flagged with @ because they do not fit on the terminal display  
120733     shall be affected.

120734     *Current line:* Unchanged.

120735     *Current column:* Unchanged.

**120736 Scroll Backward**

120737 *Synopsis:*     [*count*] <control>-U

120738     If the current line is the first line of the edit buffer, it shall be an error.

120739     If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D  
120740     or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*  
120741     shall default to the value of the **scroll** edit option.

120742     *Current line:* If *count* is greater than the current line, set to 1; otherwise, set to the current line -  
120743     *count*.

120744     *Current column:* Set to non-<blank>.

**120745 Scroll Backward by Line**

120746 *Synopsis:*     [*count*] <control>-Y

120747     Display the line *count* lines before the first line currently displayed.

120748     If the current line is the first line of the edit buffer, it shall be an error. If this calculation would  
120749     result in a line that is before the first line of the edit buffer, the first line of the display shall  
120750     display some portion of the first line of the edit buffer.

120751     *Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first  
120752     line displayed.

120753     *Current column:* Unchanged.

**120754 Edit the Alternate File**

120755 *Synopsis:*     <control>-^

120756     This command shall be equivalent to the *ex edit* command, with the alternate pathname as its  
120757     argument.

120758 **Terminate Command or Input Mode**

120759 *Synopsis:*     <ESC>

120760 If a partial *vi* command (as defined by at least one, non-*count* character) has been entered,  
120761 discard the *count* and the command character(s).

120762 Otherwise, if no command characters have been entered, and the <ESC> was the result of a map  
120763 expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall  
120764 not be an error.

120765 Otherwise, it shall be an error.

120766 *Current line:* Unchanged.

120767 *Current column:* Unchanged.

120768 **Search for tagstring**

120769 *Synopsis:*     <control>-]

120770 If the current character is not a word or <blank>, it shall be an error.

120771 This command shall be equivalent to the *ex tag* command, with the argument to that command  
120772 defined as follows.

120773 If the current character is a <blank>:

- 120774     1. Skip all <blank> characters after the cursor up to the end of the line.
- 120775     2. If the end of the line is reached, it shall be an error.

120776 Then, the argument to the *ex tag* command shall be the current character and all subsequent  
120777 characters, up to the first non-word character or the end of the line.

120778 **Move Cursor Forward**

120779 *Synopsis:*     [*count*] <space>  
120780                [*count*] 1 (ell)

120781 If there are less than *count* non-<newline> characters after the cursor on the current line, *count*  
120782 shall be adjusted to the number of non-<newline> characters after the cursor on the line.

120783 If used as a motion command:

- 120784     1. If the current or *count*th character after the cursor is the last non-<newline> in the line, the  
120785       text region shall be comprised of the current character up to and including the last  
120786       non-<newline> in the line. Otherwise, the text region shall be from the current character  
120787       up to, but not including, the *count*th character after the cursor.
- 120788     2. Any text copied to a buffer shall be in character mode.

120789 If not used as a motion command:

120790 If there are no non-<newline> characters after the current character on the current line, it shall be  
120791 an error.

120792 *Current line:* Unchanged.

120793 *Current column:* Set to the last column that displays any portion of the *count*th character after the  
120794 current character.



120795 **Replace Text with Results from Shell Command**

120796 *Synopsis:* `[count] ! motion shell-commands <newline>`

120797 If the motion command is the `!` command repeated:

- 120798 1. If the edit buffer is empty and no *count* was supplied, the command shall be the  
 120799 equivalent of the `ex :read !` command, with the text input, and no text shall be copied to  
 120800 any buffer.
- 120801 2. Otherwise:
- 120802 a. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be  
 120803 an error.
- 120804 b. The text region shall be from the current line up to and including the next *count* -1  
 120805 lines.

120806 Otherwise, the text region shall be the lines in which any character of the text region specified by  
 120807 the motion command appear.

120808 Any text copied to a buffer shall be in line mode.

120809 This command shall be equivalent to the `ex !` command for the specified lines.

120810 **Move Cursor to End-of-Line**

120811 *Synopsis:* `[count] $`

120812 It shall be an error if there are less than (*count* -1) lines after the current line in the edit buffer.

120813 If used as a motion command:

- 120814 1. If *count* is 1:
- 120815 a. It shall be an error if the line is empty.
- 120816 b. Otherwise, the text region shall consist of all characters from the starting cursor to  
 120817 the last non-`<newline>` in the line, inclusive, and any text copied to a buffer shall  
 120818 be in character mode.
- 120819 2. Otherwise, if the starting cursor position is at or before the first non-`<blank>` in the line,  
 120820 the text region shall consist of the current and the next *count* -1 lines, and any text saved  
 120821 to a buffer shall be in line mode.
- 120822 3. Otherwise, the text region shall consist of all characters from the starting cursor to the last  
 120823 non-`<newline>` in the line that is *count* -1 lines forward from the current line, and any text  
 120824 copied to a buffer shall be in character mode.

120825 If not used as a motion command:

120826 *Current line:* Set to the *current line* + *count* -1.

120827 *Current column:* The current column is set to the last display line column of the last  
 120828 non-`<newline>` in the line, or column position 1 if the line is empty.

120829 The current column shall be adjusted to be on the last display line column of the last  
 120830 non-`<newline>` of the current line as subsequent commands change the current line, until a  
 120831 command changes the current column.

120832       **Move to Matching Character**

120833       *Synopsis:*       %

120834       If the character at the current position is not a parenthesis, bracket, or curly brace, search  
120835       forward in the line to the first one of those characters. If no such character is found, it shall be an  
120836       error.

120837       The matching character shall be the parenthesis, bracket, or curly brace matching the  
120838       parenthesis, bracket, or curly brace, respectively, that was at the current position or that was  
120839       found on the current line.

120840       Matching shall be determined as follows, for an open parenthesis:

- 120841           1. Set a counter to 1.
- 120842           2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.
- 120843           3. If the end of the edit buffer is reached, it shall be an error.
- 120844           4. If an open parenthesis is found, increment the counter by 1.
- 120845           5. If a close parenthesis is found, decrement the counter by 1.
- 120846           6. If the counter is zero, the current character is the matching character.

120847       Matching for a close parenthesis shall be equivalent, except that the search shall be backwards,  
120848       from the starting character to the beginning of the buffer, a close parenthesis shall increment the  
120849       counter by 1, and an open parenthesis shall decrement the counter by 1.

120850       Matching for brackets and curly braces shall be equivalent, except that searching shall be done  
120851       for open and close brackets or open and close curly braces. It is implementation-defined whether  
120852       other characters are searched for and matched as well.

120853       If used as a motion command:

- 120854           1. If the matching cursor was after the starting cursor in the edit buffer, and the starting  
120855           cursor position was at or before the first non-<blank> non-<newline> in the starting line,  
120856           and the matching cursor position was at or after the last non-<blank> non-<newline> in  
120857           the matching line, the text region shall consist of the current line to the matching line,  
120858           inclusive, and any text copied to a buffer shall be in line mode.
- 120859           2. If the matching cursor was before the starting cursor in the edit buffer, and the starting  
120860           cursor position was at or after the last non-<blank> non-<newline> in the starting line,  
120861           and the matching cursor position was at or before the first non-<blank> non-<newline> in  
120862           the matching line, the text region shall consist of the current line to the matching line,  
120863           inclusive, and any text copied to a buffer shall be in line mode.
- 120864           3. Otherwise, the text region shall consist of the starting character to the matching character,  
120865           inclusive, and any text copied to a buffer shall be in character mode.

120866       If not used as a motion command:

120867       *Current line:* Set to the line where the matching character is located.

120868       *Current column:* Set to the last column where any portion of the matching character is displayed.

120869       **Repeat Substitution**

120870       *Synopsis:*       &

120871       Repeat the previous substitution command. This command shall be equivalent to the *ex &*  
120872       command with the current line as its addresses, and without *options, count, or flags*.

120873       **Return to Previous Context at Beginning of Line**

120874       *Synopsis:*       ' *character*

120875       It shall be an error if there is no line in the edit buffer marked by *character*.

120876       If used as a motion command:

- 120877           1. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
120878           and the marked cursor in the edit buffer shall be logically swapped.
- 120879           2. The text region shall consist of the starting line up to and including the marked line, and  
120880           any text copied to a buffer shall be in line mode.

120881       If not used as a motion command:

120882       *Current line:* Set to the line referenced by the mark.

120883       *Current column:* Set to non-<blank>.

120884       **Return to Previous Context**

120885       *Synopsis:*       ` *character*

120886       It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer  
120887       contains a character in the saved numbered character position, it shall be as if the marked  
120888       position is the first non-<blank>.

120889       If used as a motion command:

- 120890           1. It shall be an error if the marked cursor references the same character in the edit buffer as  
120891           the starting cursor.
- 120892           2. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
120893           and the marked cursor in the edit buffer shall be logically swapped.
- 120894           3. If the starting line is empty or the starting cursor is at or before the first non-<blank>  
120895           non-<newline> of the starting line, and the marked cursor line is empty or the marked  
120896           cursor references the first character of the marked cursor line, the text region shall consist  
120897           of all lines containing characters from the starting cursor to the line before the marked  
120898           cursor line, inclusive, and any text copied to a buffer shall be in line mode.
- 120899           4. Otherwise, if the marked cursor line is empty or the marked cursor references a character  
120900           at or before the first non-<blank> non-<newline> of the marked cursor line, the region of  
120901           text shall be from the starting cursor to the last non-<newline> of the line before the  
120902           marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.
- 120903           5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked  
120904           cursor (exclusive), and any text copied to a buffer shall be in character mode.

120905       If not used as a motion command:

120906       *Current line:* Set to the line referenced by the mark.

120907       *Current column:* Set to the last column in which any portion of the character referenced by the

120908 mark is displayed.

### 120909 **Return to Previous Section**

120910 *Synopsis:* [ *count* ] [ [

120911 Move the cursor backward through the edit buffer to the first character of the previous section  
120912 boundary, *count* times.

120913 If used as a motion command:

- 120914 1. If the starting cursor was at the first character of the starting line or the starting line was  
120915 empty, and the first character of the boundary was the first character of the boundary line,  
120916 the text region shall consist of the current line up to and including the line where the  
120917 *count*th next boundary starts, and any text copied to a buffer shall be in line mode.
- 120918 2. If the boundary was the last line of the edit buffer or the last non-<newline> of the last  
120919 line of the edit buffer, the text region shall consist of the last character in the edit buffer up  
120920 to and including the starting character, and any text saved to a buffer shall be in character  
120921 mode.
- 120922 3. Otherwise, the text region shall consist of the starting character up to but not including  
120923 the first character in the *count*th next boundary, and any text copied to a buffer shall be in  
120924 character mode.

120925 If not used as a motion command:

120926 *Current line:* Set to the line where the *count*th next boundary in the edit buffer starts.

120927 *Current column:* Set to the last column in which any portion of the first character of the *count*th  
120928 next boundary is displayed, or column position 1 if the line is empty.

### 120929 **Move to Next Section**

120930 *Synopsis:* [ *count* ] ] ]

120931 Move the cursor forward through the edit buffer to the first character of the next section  
120932 boundary, *count* times.

120933 If used as a motion command:

- 120934 1. If the starting cursor was at the first character of the starting line or the starting line was  
120935 empty, and the first character of the boundary was the first character of the boundary line,  
120936 the text region shall consist of the current line up to and including the line where the  
120937 *count*th previous boundary starts, and any text copied to a buffer shall be in line mode.
- 120938 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first  
120939 character in the edit buffer up to but not including the starting character, and any text  
120940 copied to a buffer shall be in character mode.
- 120941 3. Otherwise, the text region shall consist of the first character in the *count*th previous  
120942 section boundary up to but not including the starting character, and any text copied to a  
120943 buffer shall be in character mode.

120944 If not used as a motion command:

120945 *Current line:* Set to the line where the *count*th previous boundary in the edit buffer starts.

120946 *Current column:* Set to the last column in which any portion of the first character of the *count*th  
120947 previous boundary is displayed, or column position 1 if the line is empty.

120948 **Move to First Non-<blank> Position on Current Line**

120949 *Synopsis:*     ^

120950 If used as a motion command:

- 120951     1. If the line has no non-<blank> non-<newline> characters, or if the cursor is at the first  
120952       non-<blank> non-<newline> of the line, it shall be an error.
- 120953     2. If the cursor is before the first non-<blank> non-<newline> of the line, the text region  
120954       shall be comprised of the current character, up to, but not including, the first non-<blank>  
120955       non-<newline> of the line.
- 120956     3. If the cursor is after the first non-<blank> non-<newline> of the line, the text region shall  
120957       be from the character before the starting cursor up to and including the first non-<blank>  
120958       non-<newline> of the line.
- 120959     4. Any text copied to a buffer shall be in character mode.

120960 If not used as a motion command:

120961 *Current line:* Unchanged.

120962 *Current column:* Set to non-<blank>.

120963 **Current and Line Above**

120964 *Synopsis:*     [*count*] \_

120965 If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an error.

120966 If used as a motion command:

- 120967     1. If *count* is less than 2, the text region shall be the current line.
- 120968     2. Otherwise, the text region shall include the starting line and the next *count* -1 lines.
- 120969     3. Any text copied to a buffer shall be in line mode.

120970 If not used as a motion command:

120971 *Current line:* Set to current line + *count* -1.

120972 *Current column:* Set to non-<blank>.

120973 **Move Back to Beginning of Sentence**

120974 *Synopsis:*     [*count*] (

120975 Move backward to the beginning of a sentence. This command shall be equivalent to the [[  
120976 command, with the exception that sentence boundaries shall be used instead of section  
120977 boundaries.

120978 **Move Forward to Beginning of Sentence**

120979 *Synopsis:*     [*count*] )

120980 Move forward to the beginning of a sentence. This command shall be equivalent to the ]]  
120981 command, with the exception that sentence boundaries shall be used instead of section  
120982 boundaries.

120983      **Move Back to Preceding Paragraph**

120984      *Synopsis:*      [*count*] {

120985      Move back to the beginning of the preceding paragraph. This command shall be equivalent to  
120986      the `[[` command, with the exception that paragraph boundaries shall be used instead of section  
120987      boundaries.

120988      **Move Forward to Next Paragraph**

120989      *Synopsis:*      [*count*] }

120990      Move forward to the beginning of the next paragraph. This command shall be equivalent to the  
120991      `]]` command, with the exception that paragraph boundaries shall be used instead of section  
120992      boundaries.

120993      **Move to Specific Column Position**

120994      *Synopsis:*      [*count*] |

120995      For the purposes of this command, lines that are too long for the current display and that have  
120996      been folded shall be treated as having a single, 1-based, number of columns.

120997      If there are less than *count* columns in which characters from the current line are displayed on  
120998      the screen, *count* shall be adjusted to be the last column in which any portion of the line is  
120999      displayed on the screen.

121000      If used as a motion command:

- 121001      1. If the line is empty, or the cursor character is the same as the character on the *count*th  
121002      column of the line, it shall be an error.
- 121003      2. If the cursor is before the *count*th column of the line, the text region shall be comprised of  
121004      the current character, up to but not including the character on the *count*th column of the  
121005      line.
- 121006      3. If the cursor is after the *count*th column of the line, the text region shall be from the  
121007      character before the starting cursor up to and including the character on the *count*th  
121008      column of the line.
- 121009      4. Any text copied to a buffer shall be in character mode.

121010      If not used as a motion command:

121011      *Current line:* Unchanged.

121012      *Current column:* Set to the last column in which any portion of the character that is displayed in  
121013      the *count* column of the line is displayed.

121014      **Reverse Find Character**

121015      *Synopsis:*      [*count*] ,

121016      If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or  
121017      **T** command, respectively, with the specified *count* and the same search character.

121018      If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

121019 **Repeat**121020 *Synopsis:* [count] .

121021 Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or **~** command. It shall  
 121022 be an error if none of these commands have been executed. Commands (other than commands  
 121023 that enter text input mode) executed as a result of map expansions, shall not change the value of  
 121024 the last repeatable command.

121025 Repeated commands with associated motion commands shall repeat the motion command as  
 121026 well; however, any specified *count* shall replace the *count*(s) that were originally specified to the  
 121027 repeated command or its associated motion command.

121028 If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall  
 121029 not set the remembered search character for the **;** and **,** commands.

121030 If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric  
 121031 buffer named with a number less than 9, the buffer associated with the repeated command shall  
 121032 be set to be the buffer named by the name of the previous buffer logically incremented by 1.

121033 If the repeated character is a text input command, the input text associated with that command  
 121034 is repeated literally:

- 121035 • Input characters are neither macro or abbreviation-expanded.
- 121036 • Input characters are not interpreted in any special way with the exception that <newline>,  
 121037 <carriage-return>, and <control>-T behave as described in [Input Mode Commands in vi](#)  
 121038 (on page 3561).

121039 *Current line:* Set as described for the repeated command.

121040 *Current column:* Set as described for the repeated command.

121041 **Find Regular Expression**121042 *Synopsis:* /

121043 If the input line contains no non-<newline> characters, it shall be equivalent to a line containing  
 121044 only the last regular expression encountered. The enhanced regular expressions supported by *vi*  
 121045 are described in [Regular Expressions in ex](#) (on page 2875).

121046 Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed  
 121047 by an address offset or a *vi z* command.

121048 If the regular expression is not the last regular expression on the line, or if a line offset or **z**  
 121049 command is specified, the regular expression shall be terminated by an unescaped **'/'**  
 121050 character, which shall not be used as part of the regular expression. If the regular expression is  
 121051 not the first regular expression on the line, it shall be preceded by zero or more <blank>  
 121052 characters, a <semicolon>, zero or more <blank> characters, and a leading **'/'** character, which  
 121053 shall not be interpreted as part of the regular expression. It shall be an error to precede any  
 121054 regular expression with any characters other than these.

121055 Each search shall begin from the character after the first character of the last match (or, if it is the  
 121056 first search, after the cursor). If the **wraps**can edit option is set, the search shall continue to the  
 121057 character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be  
 121058 an error if any search fails to find a match, and an informational message to this effect shall be  
 121059 displayed.

121060 An optional address offset (see [Addressing in ex](#), on page 2845) can be specified after the last  
 121061 regular expression by including a trailing **'/'** character after the regular expression and

121062 specifying the address offset. This offset shall be from the line containing the match for the last  
 121063 regular expression specified. It shall be an error if the line offset would indicate a line address  
 121064 less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be  
 121065 supported. It shall be an error to follow the address offset with any other characters than  
 121066 <blank> characters.

121067 If not used as a motion command, an optional **z** command (see [Redraw Window](#), on page 3560)  
 121068 can be specified after the last regular expression by including a trailing '/' character after the  
 121069 regular expression, zero or more <blank> characters, a 'z', zero or more <blank> characters, an  
 121070 optional new **window** edit option value, zero or more <blank> characters, and a location  
 121071 character. The effect shall be as if the **z** command was executed after the / command. It shall be  
 121072 an error to follow the **z** command with any other characters than <blank> characters.

121073 The remembered search direction shall be set to forward.

121074 If used as a motion command:

- 121075 1. It shall be an error if the last match references the same character in the edit buffer as the  
 121076 starting cursor.
- 121077 2. If any address offset is specified, the last match shall be adjusted by the specified offset as  
 121078 described previously.
- 121079 3. If the starting cursor is after the last match, then the locations of the starting cursor and  
 121080 the last match in the edit buffer shall be logically swapped.
- 121081 4. If any address offset is specified, the text region shall consist of all lines containing  
 121082 characters from the starting cursor to the last match line, inclusive, and any text copied to  
 121083 a buffer shall be in line mode.
- 121084 5. Otherwise, if the starting line is empty or the starting cursor is at or before the first  
 121085 non-<blank> non-<newline> of the starting line, and the last match line is empty or the  
 121086 last match starts at the first character of the last match line, the text region shall consist of  
 121087 all lines containing characters from the starting cursor to the line before the last match  
 121088 line, inclusive, and any text copied to a buffer shall be in line mode.
- 121089 6. Otherwise, if the last match line is empty or the last match begins at a character at or  
 121090 before the first non-<blank> non-<newline> of the last match line, the region of text shall  
 121091 be from the current cursor to the last non-<newline> of the line before the last match line,  
 121092 inclusive, and any text copied to a buffer shall be in character mode.
- 121093 7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first  
 121094 character of the last match (exclusive), and any text copied to a buffer shall be in character  
 121095 mode.

121096 If not used as a motion command:

121097 *Current line*: If a match is found, set to the last matched line plus the address offset, if any;  
 121098 otherwise, unchanged.

121099 *Current column*: Set to the last column on which any portion of the first character in the last  
 121100 matched string is displayed, if a match is found; otherwise, unchanged.



121101 **Move to First Character in Line**121102 *Synopsis:* 0 (zero)121103 Move to the first character on the current line. The character '0' shall not be interpreted as a  
121104 command if it is immediately preceded by a digit.

121105 If used as a motion command:

- 121106 1. If the cursor character is the first character in the line, it shall be an error.
- 121107 2. The text region shall be from the character before the cursor character up to and including  
121108 the first character in the line.
- 121109 3. Any text copied to a buffer shall be in character mode.

121110 If not used as a motion command:

121111 *Current line:* Unchanged.121112 *Current column:* The last column in which any portion of the first character in the line is  
121113 displayed, or if the line is empty, unchanged.121114 **Execute an ex Command**121115 *Synopsis:* :121116 Execute one or more *ex* commands.121117 If any portion of the screen other than the last line of the screen was overwritten by any *ex*  
121118 command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from  
121119 the user, and shall then read a character. This action may also be taken for other, unspecified  
121120 reasons.121121 If the next character entered is a ':', another *ex* command shall be accepted and executed. Any  
121122 other character shall cause the screen to be refreshed and *vi* shall return to command mode.121123 *Current line:* As specified for the *ex* command.121124 *Current column:* As specified for the *ex* command.121125 **Repeat Find**121126 *Synopsis:* [count] ;121127 This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and  
121128 with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**,  
121129 **f**, **T**, or **t** command, it shall be an error.121130 **Shift Left**121131 *Synopsis:* [count] < *motion*

121132 If the motion command is the &lt; command repeated:

- 121133 1. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an  
121134 error.
- 121135 2. The text region shall be from the current line, up to and including the next *count* - 1 lines.

121136 Shift any line in the text region specified by the *count* and motion command one shiftwidth (see  
121137 the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* < command. The  
121138 unshifted lines shall be copied to the unnamed buffer in line mode.

121139 *Current line*: If the motion was from the current cursor position toward the end of the edit buffer,  
 121140 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
 121141 specified by the motion command.

121142 *Current column*: Set to non-<blank>.

### 121143 **Shift Right**

121144 *Synopsis*: `[count] > motion`

121145 If the motion command is the `>` command repeated:

- 121146 1. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an  
 121147 error.
- 121148 2. The text region shall be from the current line, up to and including the next *count* - 1 lines.

121149 Shift any line with characters in the text region specified by the *count* and motion command one  
 121150 shiftwidth (see the *ex* **shiftwidth** option) away from the start of the line, as described by the *ex* `>`  
 121151 command. The unshifted lines shall be copied into the unnamed buffer in line mode.

121152 *Current line*: If the motion was from the current cursor position toward the end of the edit buffer,  
 121153 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
 121154 specified by the motion command.

121155 *Current column*: Set to non-<blank>.

### 121156 **Scan Backwards for Regular Expression**

121157 *Synopsis*: `?`

121158 Scan backwards; the `?` command shall be equivalent to the `/` command (see [Find Regular](#)  
 121159 [Expression](#), on page 3543) with the following exceptions:

- 121160 1. The input prompt shall be a `'?'`.
- 121161 2. Each search shall begin from the character before the first character of the last match (or, if  
 121162 it is the first search, the character before the cursor character).
- 121163 3. The search direction shall be from the cursor toward the beginning of the edit buffer, and  
 121164 the **wrapscan** edit option shall affect whether the search wraps to the end of the edit  
 121165 buffer and continues.
- 121166 4. The remembered search direction shall be set to backward.

### 121167 **Execute**

121168 *Synopsis*: `@buffer`

121169 If the *buffer* is specified as `@`, the last buffer executed shall be used. If no previous buffer has been  
 121170 executed, it shall be an error.

121171 Behave as if the contents of the named buffer were entered as standard input. After each line of a  
 121172 line-mode buffer, and all but the last line of a character mode buffer, behave as if a `<newline>`  
 121173 were entered as standard input.

121174 If an error occurs during this process, an error message shall be written, and no more characters  
 121175 resulting from the execution of this command shall be processed.

121176 If a *count* is specified, behave as if that count were entered as user input before the characters  
 121177 from the `@` buffer were entered.

121178 *Current line*: As specified for the individual commands.

121179 *Current column*: As specified for the individual commands.

### 121180 **Reverse Case**

121181 *Synopsis*: [count] ~

121182 Reverse the case of the current character and the next *count* -1 characters, such that lowercase  
121183 characters that have uppercase counterparts shall be changed to uppercase characters, and  
121184 uppercase characters that have lowercase counterparts shall be changed to lowercase characters,  
121185 as prescribed by the current locale. No other characters shall be affected by this command.

121186 If there are less than *count* -1 characters after the cursor in the edit buffer, *count* shall be adjusted  
121187 to the number of characters after the cursor in the edit buffer minus 1.

121188 For the purposes of this command, the next character after the last non-<newline> on the line  
121189 shall be the next character in the edit buffer.

121190 *Current line*: Set to the line including the (*count*-1)th character after the cursor.

121191 *Current column*: Set to the last column in which any portion of the (*count*-1)th character after the  
121192 cursor is displayed.

### 121193 **Append**

121194 *Synopsis*: [count] a

121195 Enter text input mode after the current cursor position. No characters already in the edit buffer  
121196 shall be affected by this command. A *count* shall cause the input text to be appended *count* -1  
121197 more times to the end of the input.

121198 *Current line/column*: As specified for the text input commands (see [Input Mode Commands in vi](#),  
121199 on page 3561).

### 121200 **Append at End-of-Line**

121201 *Synopsis*: [count] A

121202 This command shall be equivalent to the *vi* command:

121203 \$ [ count ] a

121204 (see [Append](#)).

### 121205 **Move Backward to Preceding Word**

121206 *Synopsis*: [count] b

121207 With the exception that words are used as the delimiter instead of bigwords, this command shall  
121208 be equivalent to the **B** command.

## 121209 Move Backward to Preceding Bigword

121210 *Synopsis:* [count] B

121211 If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an  
 121212 error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count*  
 121213 shall be adjusted to the number of bigword beginnings between the cursor and the start of the  
 121214 edit buffer.

121215 If used as a motion command:

- 121216 1. The text region shall be from the first character of the *count*th previous bigword beginning  
 121217 up to but not including the cursor character.
- 121218 2. Any text copied to a buffer shall be in character mode.

121219 If not used as a motion command:

121220 *Current line:* Set to the line containing the *current column*.

121221 *Current column:* Set to the last column upon which any part of the first character of the *count*th  
 121222 previous bigword is displayed.

## 121223 Change

121224 *Synopsis:* [buffer] [count] c motion

121225 If the motion command is the **c** command repeated:

- 121226 1. The buffer text shall be in line mode.
- 121227 2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
 121228 error.
- 121229 3. The text region shall be from the current line up to and including the next *count* -1 lines.

121230 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

121231 The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text  
 121232 to be replaced contains characters from more than a single line, or the buffer text is in line mode,  
 121233 the replaced text shall be copied into the numeric buffers as well.

121234 If the buffer text is in line mode:

- 121235 1. Any lines that contain characters in the region shall be deleted, and the editor shall enter  
 121236 text input mode at the beginning of a new line which shall replace the first line deleted.
- 121237 2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent**  
 121238 characters on the first line deleted shall be inserted as if entered by the user.

121239 Otherwise, if characters from more than one line are in the region of text:

- 121240 1. The text shall be deleted.
- 121241 2. Any text remaining in the last line in the text region shall be appended to the first line in  
 121242 the region, and the last line in the region shall be deleted.
- 121243 3. The editor shall enter text input mode after the last character not deleted from the first  
 121244 line in the text region, if any; otherwise, on the first column of the first line in the region.

121245 Otherwise:

- 121246 1. If the glyph for '\$' is smaller than the region, the end of the region shall be marked with  
121247 a '\$'.
- 121248 2. The editor shall enter text input mode, overwriting the region of text.

121249 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),  
121250 on page 3561).

### 121251 **Change to End-of-Line**

121252 *Synopsis:* `[buffer] [count] C`

121253 This command shall be equivalent to the *vi* command:

121254 `[buffer] [count] c$`

121255 See the **c** command.

### 121256 **Delete**

121257 *Synopsis:* `[buffer] [count] d motion`

121258 If the motion command is the **d** command repeated:

- 121259 1. The buffer text shall be in line mode.
- 121260 2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
121261 error.
- 121262 3. The text region shall be from the current line up to and including the next *count* -1 lines.

121263 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

121264 If in open mode, and the current line is deleted, and the line remains on the display, an '@'  
121265 character shall be displayed as the first glyph of that line.

121266 Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be  
121267 deleted contains characters from more than a single line, or the buffer text is in line mode, the  
121268 deleted text shall be copied into the numeric buffers, as well.

121269 *Current line:* Set to the first text region line that appears in the edit buffer, unless that line has  
121270 been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit  
121271 buffer is empty.

121272 *Current column:*

- 121273 1. If the line is empty, set to column position 1.
- 121274 2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the  
121275 end of the edit buffer:
- 121276 a. If a character from the current line is displayed in the current column, set to the  
121277 last column that displays any portion of that character.
- 121278 b. Otherwise, set to the last column in which any portion of any character in the line  
121279 is displayed.
- 121280 3. Otherwise, if a character is displayed in the column that began the text region, set to the  
121281 last column that displays any portion of that character.

121282 4. Otherwise, set to the last column in which any portion of any character in the line is  
121283 displayed.

#### 121284 **Delete to End-of-Line**

121285 *Synopsis:* `[buffer] D`

121286 Delete the text from the current position to the end of the current line; equivalent to the *vi*  
121287 command:

121288 `[buffer] d$`

#### 121289 **Move to End-of-Word**

121290 *Synopsis:* `[count] e`

121291 With the exception that words are used instead of bigwords as the delimiter, this command shall  
121292 be equivalent to the **E** command.

#### 121293 **Move to End-of-Bigword**

121294 *Synopsis:* `[count] E`

121295 If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor  
121296 and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between  
121297 the cursor and the end of the edit buffer.

121298 If used as a motion command:

- 121299 1. The text region shall be from the last character of the *count*th next bigword up to and  
121300 including the cursor character.
- 121301 2. Any text copied to a buffer shall be in character mode.

121302 If not used as a motion command:

121303 *Current line:* Set to the line containing the current column.

121304 *Current column:* Set to the last column upon which any part of the last character of the *count*th  
121305 next bigword is displayed.

#### 121306 **Find Character in Current Line (Forward)**

121307 *Synopsis:* `[count] f character`

121308 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

121309 If used as a motion command:

- 121310 1. The text range shall be from the cursor character up to and including the *count*th  
121311 occurrence of the specified character after the cursor.
- 121312 2. Any text copied to a buffer shall be in character mode.

121313 If not used as a motion command:

121314 *Current line:* Unchanged.

121315 *Current column:* Set to the last column in which any portion of the *count*th occurrence of the  
121316 specified character after the cursor appears in the line.

121317 **Find Character in Current Line (Reverse)**121318 *Synopsis:* [count] F *character*121319 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

121320 If used as a motion command:

- 121321 1. The text region shall be from the *count*th occurrence of the specified character before the  
121322 cursor, up to, but not including the cursor character.
- 121323 2. Any text copied to a buffer shall be in character mode.

121324 If not used as a motion command:

121325 *Current line:* Unchanged.121326 *Current column:* Set to the last column in which any portion of the *count*th occurrence of the  
121327 specified character before the cursor appears in the line.121328 **Move to Line**121329 *Synopsis:* [count] G121330 If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than  
121331 the last line of the edit buffer, it shall be an error.

121332 If used as a motion command:

- 121333 1. The text region shall be from the cursor line up to and including the specified line.
- 121334 2. Any text copied to a buffer shall be in line mode.

121335 If not used as a motion command:

121336 *Current line:* Set to *count* if *count* is specified; otherwise, the last line.121337 *Current column:* Set to non-<blank>.121338 **Move to Top of Screen**121339 *Synopsis:* [count] H121340 If the beginning of the line *count* greater than the first line of which any portion appears on the  
121341 display does not exist, it shall be an error.

121342 If used as a motion command:

- 121343 1. If in open mode, the text region shall be the current line.
- 121344 2. Otherwise, the text region shall be from the starting line up to and including (the first line  
121345 of the display + *count* -1).
- 121346 3. Any text copied to a buffer shall be in line mode.

121347 If not used as a motion command:

121348 If in open mode, this command shall set the current column to non-&lt;blank&gt; and do nothing else.

121349 Otherwise, it shall set the current line and current column as follows.

121350 *Current line:* Set to (the first line of the display + *count* -1).121351 *Current column:* Set to non-<blank>.

121352        **Insert Before Cursor**

121353        *Synopsis:*        [count] i

121354        Enter text input mode before the current cursor position. No characters already in the edit buffer  
121355        shall be affected by this command. A *count* shall cause the input text to be appended *count* -1  
121356        more times to the end of the input.

121357        *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),  
121358        on page 3561).

121359        **Insert at Beginning of Line**

121360        *Synopsis:*        [count] I

121361        This command shall be equivalent to the *vi* command  $\wedge$ [count]i.

121362        **Join**

121363        *Synopsis:*        [count] J

121364        If the current line is the last line in the edit buffer, it shall be an error.

121365        This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex*  
121366        command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex*  
121367        command *count* value of *count* -1 for any other value of *count*, except that the current line and  
121368        column shall be set as follows.

121369        *Current line:* Unchanged.

121370        *Current column:* The last column in which any portion of the character following the last  
121371        character in the initial line is displayed, or the last non-<newline> in the line if no characters  
121372        were appended.

121373        **Move to Bottom of Screen**

121374        *Synopsis:*        [count] L

121375        If the beginning of the line *count* less than the last line of which any portion appears on the  
121376        display does not exist, it shall be an error.

121377        If used as a motion command:

- 121378            1. If in open mode, the text region shall be the current line.
- 121379            2. Otherwise, the text region shall include all lines from the starting cursor line to (the last  
121380            line of the display -(*count* -1)).
- 121381            3. Any text copied to a buffer shall be in line mode.

121382        If not used as a motion command:

- 121383            1. If in open mode, this command shall set the current column to non-<blank> and do  
121384            nothing else.
- 121385            2. Otherwise, it shall set the current line and current column as follows.

121386        *Current line:* Set to (the last line of the display -(*count* -1)).

121387        *Current column:* Set to non-<blank>.



121388 **Mark Position**121389 *Synopsis:* m letter121390 This command shall be equivalent to the *ex* **mark** command with the specified character as an  
121391 argument.121392 **Move to Middle of Screen**121393 *Synopsis:* M

121394 The middle line of the display shall be calculated as follows:

121395  $(\text{the top line of the display}) + ((\text{number of lines displayed}) + 1) / 2) - 1$ 

121396 If used as a motion command:

- 121397 1. If in open mode, the text region shall be the current line.
- 121398 2. Otherwise, the text region shall include all lines from the starting cursor line up to and  
121399 including the middle line of the display.
- 121400 3. Any text copied to a buffer shall be in line mode.

121401 If not used as a motion command:

121402 If in open mode, this command shall set the current column to non-&lt;blank&gt; and do nothing else.

121403 Otherwise, it shall set the current line and current column as follows.

121404 *Current line:* Set to the middle line of the display.121405 *Current column:* Set to non-<blank>.121406 **Repeat Regular Expression Find (Forward)**121407 *Synopsis:* n121408 If the remembered search direction was forward, the **n** command shall be equivalent to the *vi* /  
121409 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* ?  
121410 command with no characters entered by the user.121411 If the **n** command is used as a motion command for the **!** command, the editor shall not enter  
121412 text input mode on the last line on the screen, and shall behave as if the user entered a single  
121413 '!' character as the text input.121414 **Repeat Regular Expression Find (Reverse)**121415 *Synopsis:* N121416 Scan for the next match of the last pattern given to / or ?, but in the reverse direction; this is the  
121417 reverse of **n**.121418 If the remembered search direction was forward, the **N** command shall be equivalent to the *vi* ?  
121419 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* /  
121420 command with no characters entered by the user. If the **N** command is used as a motion  
121421 command for the **!** command, the editor shall not enter text input mode on the last line on the  
121422 screen, and shall behave as if the user entered a single **!** character as the text input.

121423 **Insert Empty Line Below**

121424 *Synopsis:*     ○

121425 Enter text input mode in a new line appended after the current line. A *count* shall cause the input  
121426 text to be appended *count* -1 more times to the end of the already added text, each time starting  
121427 on a new, appended line.

121428 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),  
121429 on page 3561).

121430 **Insert Empty Line Above**

121431 *Synopsis:*     ○

121432 Enter text input mode in a new line inserted before the current line. A *count* shall cause the input  
121433 text to be appended *count* -1 more times to the end of the already added text, each time starting  
121434 on a new, appended line.

121435 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),  
121436 on page 3561).

121437 **Put from Buffer Following**

121438 *Synopsis:*     [*buffer*] p

121439 If no *buffer* is specified, the unnamed buffer shall be used.

121440 If the buffer text is in line mode, the text shall be appended below the current line, and each line  
121441 of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
121442 appended *count* -1 more times to the end of the already added text, each time starting on a new,  
121443 appended line.

121444 If the buffer text is in character mode, the text shall be appended into the current line after the  
121445 cursor, and each line of the buffer other than the first and last shall become a new line in the edit  
121446 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the  
121447 already added text, each time starting after the last added character.

121448 *Current line:* If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.

121449 *Current column:* If the buffer text is in line mode:

- 121450     1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any  
121451       portion of the first non-<blank> in the line is displayed.
- 121452     2. If there is no non-<blank> in the first line of the buffer, set to the last column on which  
121453       any portion of the last non-<newline> in the first line of the buffer is displayed.

121454 If the buffer text is in character mode:

- 121455     1. If the text in the buffer is from more than a single line, then set to the last column on  
121456       which any portion of the first character from the buffer is displayed.
- 121457     2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any  
121458       portion of the last character from the buffer is displayed.
- 121459     3. Otherwise, set to the first column on which any portion of the first character from the  
121460       buffer is displayed.

121461 **Put from Buffer Before**121462 *Synopsis:* `[buffer] P`121463 If no *buffer* is specified, the unnamed buffer shall be used.

121464 If the buffer text is in line mode, the text shall be inserted above the current line, and each line of  
 121465 the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
 121466 appended *count* -1 more times to the end of the already added text, each time starting on a new,  
 121467 appended line.

121468 If the buffer text is in character mode, the text shall be inserted into the current line before the  
 121469 cursor, and each line of the buffer other than the first and last shall become a new line in the edit  
 121470 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the  
 121471 already added text, each time starting after the last added character.

121472 *Current line:* Unchanged.121473 *Current column:* If the buffer text is in line mode:

- 121474 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any  
 121475 portion of that character is displayed.
- 121476 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which  
 121477 any portion of the last non-<newline> in the first line of the buffer is displayed.

121478 If the buffer text is in character mode:

- 121479 1. If the text in the buffer is from more than a single line, then set to the last column on  
 121480 which any portion of the first character from the buffer is displayed.
- 121481 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any  
 121482 portion of the last character from the buffer is displayed.
- 121483 3. Otherwise, set to the first column on which any portion of the first character from the  
 121484 buffer is displayed.

121485 **Enter ex Mode**121486 *Synopsis:* `Q`121487 Leave visual or open mode and enter *ex* command mode.121488 *Current line:* Unchanged.121489 *Current column:* Unchanged.121490 **Replace Character**121491 *Synopsis:* `[count] r character`

121492 Replace the *count* characters at and after the cursor with the specified character. If there are less  
 121493 than *count* non-<newline> characters at and after the cursor on the line, it shall be an error.

121494 If character is <control>-V, any next character other than the <newline> shall be stripped of any  
 121495 special meaning and used as a literal character.

121496 If character is <ESC>, no replacement shall be made and the current line and current column  
 121497 shall be unchanged.

121498 If character is <carriage-return> or <newline>, *count* new lines shall be appended to the current  
 121499 line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be

121500 discarded, and any remaining characters after the cursor in the current line shall be moved to the  
 121501 last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same  
 121502 number of **autoindent** characters found on the line from which the command was executed.

121503 *Current line*: Unchanged unless the replacement character is a <carriage-return> or <newline>, in  
 121504 which case it shall be set to line + *count*.

121505 *Current column*: Set to the last column position on which a portion of the last replaced character  
 121506 is displayed, or if the replacement character caused new lines to be created, set to non-<blank>.

### 121507 **Replace Characters**

121508 *Synopsis*:       R

121509 Enter text input mode at the current cursor position possibly replacing text on the current line. A  
 121510 *count* shall cause the input text to be appended *count* - 1 more times to the end of the input.

121511 *Current line/column*: As specified for the text input commands (see [Input Mode Commands in vi](#),  
 121512 on page 3561).

### 121513 **Substitute Character**

121514 *Synopsis*:       [*buffer*] [*count*] s

121515 This command shall be equivalent to the *vi* command:

121516 [*buffer*] [*count*] c<space>

### 121517 **Substitute Lines**

121518 *Synopsis*:       [*buffer*] [*count*] S

121519 This command shall be equivalent to the *vi* command:

121520 [*buffer*] [*count*] c\_

### 121521 **Move Cursor to Before Character (Forward)**

121522 *Synopsis*:       [*count*] t *character*

121523 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

121524 If used as a motion command:

- 121525       1. The text region shall be from the cursor up to but not including the *count*th occurrence of  
 121526       the specified character after the cursor.
- 121527       2. Any text copied to a buffer shall be in character mode.

121528 If not used as a motion command:

121529 *Current line*: Unchanged.

121530 *Current column*: Set to the last column in which any portion of the character before the *count*th  
 121531 occurrence of the specified character after the cursor appears in the line.

121532 **Move Cursor to After Character (Reverse)**121533 *Synopsis:* [count] T character121534 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

121535 If used as a motion command:

- 121536 1. If the character before the cursor is the specified character, it shall be an error.
- 121537 2. The text region shall be from the character before the cursor up to but not including the
- 121538 *count*th occurrence of the specified character before the cursor.
- 121539 3. Any text copied to a buffer shall be in character mode.

121540 If not used as a motion command:

121541 *Current line:* Unchanged.121542 *Current column:* Set to the last column in which any portion of the character after the *count*th  
121543 occurrence of the specified character before the cursor appears in the line.121544 **Undo**121545 *Synopsis:* u121546 This command shall be equivalent to the *ex* **undo** command except that the current line and  
121547 current column shall be set as follows:121548 *Current line:* Set to the first line added or changed if any; otherwise, move to the line preceding  
121549 any deleted text if one exists; otherwise, move to line 1.121550 *Current column:* If undoing an *ex* command, set to the first non-<blank>.

121551 Otherwise, if undoing a text input command:

- 121552 1. If the command was a **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to
- 121553 the value it held when the text input command was entered.
- 121554 2. Otherwise, set to the last column in which any portion of the first character after the
- 121555 deleted text is displayed, or, if no non-<newline> characters follow the text deleted from
- 121556 this line, set to the last column in which any portion of the last non-<newline> in the line
- 121557 is displayed, or 1 if the line is empty.

121558 Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:

- 121559 1. If text was added or changed, set to the last column in which any portion of the first
- 121560 character added or changed is displayed.
- 121561 2. If text was deleted, set to the last column in which any portion of the first character after
- 121562 the deleted text is displayed, or, if no non-<newline> characters follow the deleted text,
- 121563 set to the last column in which any portion of the last non-<newline> in the line is
- 121564 displayed, or 1 if the line is empty.

121565 Otherwise, set to non-&lt;blank&gt;.

121566      **Undo Current Line**

121567      *Synopsis:*      U

121568      Restore the current line to its state immediately before the most recent time that it became the  
121569      current line.

121570      *Current line:* Unchanged.

121571      *Current column:* Set to the first column in the line in which any portion of the first character in  
121572      the line is displayed.

121573      **Move to Beginning of Word**

121574      *Synopsis:*      [count] w

121575      With the exception that words are used as the delimiter instead of bigwords, this command shall  
121576      be equivalent to the **W** command.

121577      **Move to Beginning of Bigword**

121578      *Synopsis:*      [count] W

121579      If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the  
121580      cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last  
121581      bigword in the edit buffer.

121582      If used as a motion command:

- 121583            1. If the associated command is **c**, *count* is 1, and the cursor is on a <blank>, the region of  
121584            text shall be the current character and no further action shall be taken.
- 121585            2. If there are less than *count* bigwords between the cursor and the end of the edit buffer,  
121586            then the command shall succeed, and the region of text shall include the last character of  
121587            the edit buffer.
- 121588            3. If there are <blank> characters or an end-of-line that precede the *count*th bigword, and the  
121589            associated command is **c**, the region of text shall be up to and including the last character  
121590            before the preceding <blank> characters or end-of-line.
- 121591            4. If there are <blank> characters or an end-of-line that precede the bigword, and the  
121592            associated command is **d** or **y**, the region of text shall be up to and including the last  
121593            <blank> before the start of the bigword or end-of-line.
- 121594            5. Any text copied to a buffer shall be in character mode.

121595      If not used as a motion command:

- 121596            1. If the cursor is on the last character of the edit buffer, it shall be an error.

121597      *Current line:* Set to the line containing the current column.

121598      *Current column:* Set to the last column in which any part of the first character of the *count*th next  
121599      bigword is displayed.

121600 **Delete Character at Cursor**121601 *Synopsis:* `[buffer] [count] x`121602 Delete the *count* characters at and after the current character into *buffer*, if specified, and into the  
121603 unnamed buffer.121604 If the line is empty, it shall be an error. If there are less than *count* non-<newline> characters at  
121605 and after the cursor on the current line, *count* shall be adjusted to the number of non-<newline>  
121606 characters at and after the cursor.121607 *Current line:* Unchanged.121608 *Current column:* If the line is empty, set to column position 1. Otherwise, if there were *count* or  
121609 less non-<newline> characters at and after the cursor on the current line, set to the last column  
121610 that displays any part of the last non-<newline> of the line. Otherwise, unchanged.121611 **Delete Character Before Cursor**121612 *Synopsis:* `[buffer] [count] X`121613 Delete the *count* characters before the current character into *buffer*, if specified, and into the  
121614 unnamed buffer.121615 If there are no characters before the current character on the current line, it shall be an error. If  
121616 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
121617 number of previous characters on the line.121618 *Current line:* Unchanged.121619 *Current column:* Set to (current column – the width of the deleted characters).121620 **Yank**121621 *Synopsis:* `[buffer] [count] y motion`121622 Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.121623 If the motion command is the *y* command repeated:

- 121624 1. The buffer shall be in line mode.
- 121625 2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an  
121626 error.
- 121627 3. The text region shall be from the current line up to and including the next *count* –1 lines.

121628 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

121629 *Current line:* If the motion was from the current cursor position toward the end of the edit buffer,  
121630 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
121631 specified by the motion command.121632 *Current column:*

- 121633 1. If the motion was from the current cursor position toward the end of the edit buffer,  
121634 unchanged.
- 121635 2. Otherwise, if the current line is empty, set to column position 1.
- 121636 3. Otherwise, set to the last column that displays any part of the first character in the file  
121637 that is part of the text region specified by the motion command.

121638 **Yank Current Line**121639 *Synopsis:* `[buffer] [count] Y`121640 This command shall be equivalent to the *vi* command:121641 `[buffer] [count] y_`121642 **Redraw Window**121643 If in open mode, the **z** command shall have the Synopsis:121644 *Synopsis:* `[count] z`

121645 If *count* is not specified, it shall default to the **window** edit option `-1`. The **z** command shall be  
 121646 equivalent to the *ex z* command, with a type character of `=` and a *count* of *count* `-2`, except that  
 121647 the current line and current column shall be set as follows, and the **window** edit option shall not  
 121648 be affected. If the calculation for the *count* argument would result in a negative number, the  
 121649 *count* argument to the *ex z* command shall be zero. A blank line shall be written after the last line  
 121650 is written.

121651 *Current line:* Unchanged.121652 *Current column:* Unchanged.121653 If not in open mode, the **z** command shall have the following Synopsis:121654 *Synopsis:* `[line] z [count] character`

121655 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the  
 121656 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

121657 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the  
 121658 *ex window* command), and the screen shall be redrawn.

121659 *line* shall be placed as specified by the following characters:121660 `<newline>`, `<carriage-return>`

121661 Place the beginning of the line on the first line of the display.

121662 . Place the beginning of the line in the center of the display. The middle line of the display  
 121663 shall be calculated as described for the **M** command.

121664 `-` Place an unspecified portion of the line on the last line of the display.

121665 + If *line* was specified, equivalent to the `<newline>` case. If *line* was not specified, display a  
 121666 screen where the first line of the display shall be (current last line) `+1`. If there are no lines  
 121667 after the last line in the display, it shall be an error.

121668 ^ If *line* was specified, display a screen where the last line of the display shall contain an  
 121669 unspecified portion of the first line of a display that had an unspecified portion of the  
 121670 specified line on the last line of the display. If this calculation results in a line before the  
 121671 beginning of the edit buffer, display the first screen of the edit buffer.

121672 Otherwise, display a screen where the last line of the display shall contain an unspecified  
 121673 portion of (current first line `-1`). If this calculation results in a line before the beginning of  
 121674 the edit buffer, it shall be an error.



121675 *Current line*: If *line* and the '^' character were specified:

121676 1. If the first screen was displayed as a result of the command attempting to display lines

121677 before the beginning of the edit buffer: if the first screen was already displayed,

121678 unchanged; otherwise, set to (current first line -1).

121679 2. Otherwise, set to the last line of the display.

121680 If *line* and the '+' character were specified, set to the first line of the display.

121681 Otherwise, if *line* was specified, set to *line*.

121682 Otherwise, unchanged.

121683 *Current column*: Set to non-<blank>.

## 121684 **Exit**

121685 *Synopsis*: ZZ

121686 This command shall be equivalent to the *ex xit* command with no addresses, trailing **!**, or

121687 filename (see the *ex xit* command).

## 121688 **Input Mode Commands in vi**

121689 In text input mode, the current line shall consist of zero or more of the following categories, plus

121690 the terminating <newline>:

121691 1. Characters preceding the text input entry point

121692 Characters in this category shall not be modified during text input mode.

121693 2. **autoindent** characters

121694 **autoindent** characters shall be automatically inserted into each line that is created in text

121695 input mode, either as a result of entering a <newline> or <carriage-return> while in text

121696 input mode, or as an effect of the command itself; for example, **O** or **o** (see the *ex*

121697 **autoindent** command), as if entered by the user.

121698 It shall be possible to erase **autoindent** characters with the <control>-D command; it is

121699 unspecified whether they can be erased by <control>-H, <control>-U, and <control>-W

121700 characters. Erasing any **autoindent** character turns the glyph into erase-columns and

121701 deletes the character from the edit buffer, but does not change its representation on the

121702 screen.

121703 3. Text input characters

121704 Text input characters are the characters entered by the user. Erasing any text input

121705 character turns the glyph into erase-columns and deletes the character from the edit

121706 buffer, but does not change its representation on the screen.

121707 Each text input character entered by the user (that does not have a special meaning) shall

121708 be treated as follows:

121709 a. The text input character shall be appended to the last character in the edit buffer

121710 from the first, second, or third categories.

121711 b. If there are no erase-columns on the screen, the text input command was the **R**

121712 command, and characters in the fifth category from the original line follow the

121713 cursor, the next such character shall be deleted from the edit buffer. If the

121714 **slowopen** edit option is not set, the corresponding glyph on the screen shall

- 121715 become erase-columns.
- 121716 c. If there are erase-columns on the screen, as many columns as they occupy, or as are  
121717 necessary, shall be overwritten to display the text input character. (If only part of a  
121718 multi-column glyph is overwritten, the remainder shall be left on the screen, and  
121719 continue to be treated as erase-columns; it is unspecified whether the remainder of  
121720 the glyph is modified in any way.)
- 121721 d. If additional display line columns are needed to display the text input character:
- 121722 i. If the **slowopen** edit option is set, the text input characters shall be  
121723 displayed on subsequent display line columns, overwriting any characters  
121724 displayed in those columns.
- 121725 ii. Otherwise, any characters currently displayed on or after the column on the  
121726 display line where the text input character is to be displayed shall be  
121727 pushed ahead the number of display line columns necessary to display the  
121728 rest of the text input character.

#### 121729 4. Erase-columns

121730 Erase-columns are not logically part of the edit buffer, appearing only on the screen, and  
121731 may be overwritten on the screen by subsequent text input characters. When text input  
121732 mode ends, all erase-columns shall no longer appear on the screen.

121733 Erase-columns are initially the region of text specified by the **c** command (see [Change](#), on  
121734 page 3548); however, erasing **autoindent** or text input characters causes the glyphs of the  
121735 erased characters to be treated as erase-columns.

#### 121736 5. Characters following the text region for the **c** command, or the text input entry point for 121737 all other commands

121738 Characters in this category shall not be modified during text input mode, except as  
121739 specified in category 3.b. for the **R** text input command, or as <blank> characters deleted  
121740 when a <newline> or <carriage-return> is entered.

121741 It is unspecified whether it is an error to attempt to erase past the beginning of a line that was  
121742 created by the entry of a <newline> or <carriage-return> during text input mode. If it is not an  
121743 error, the editor shall behave as if the erasing character was entered immediately after the last  
121744 text input character entered on the previous line, and all of the non-<newline> characters on the  
121745 current line shall be treated as erase-columns.

121746 When text input mode is entered, or after a text input mode character is entered (except as  
121747 specified for the special characters below), the cursor shall be positioned as follows:

- 121748 1. On the first column that displays any part of the first erase-column, if one exists
- 121749 2. Otherwise, if the **slowopen** edit option is set, on the first display line column after the last  
121750 character in the first, second, or third categories, if one exists
- 121751 3. Otherwise, the first column that displays any part of the first character in the fifth  
121752 category, if one exists
- 121753 4. Otherwise, the display line column after the last character in the first, second, or third  
121754 categories, if one exists
- 121755 5. Otherwise, on column position 1

121756 The characters that are updated on the screen during text input mode are unspecified, other than  
121757 that the last text input character shall always be updated, and, if the **slowopen** edit option is not

121758 set, the current cursor character shall always be updated.

121759 The following specifications are for command characters entered during text input mode.

121760 **NUL**

121761 *Synopsis:* NUL

121762 If the first character of the text input is a NUL, the most recently input text shall be input as if  
 121763 entered by the user, and then text input mode shall be exited. The text shall be input literally;  
 121764 that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted  
 121765 in any special manner. It is unspecified whether implementations shall support more than 256  
 121766 bytes of remembered input text.

121767 **<control>-D**

121768 *Synopsis:* <control>-D

121769 The <control>-D character shall have no special meaning when in text input mode for a line-  
 121770 oriented command (see [Command Descriptions in vi](#), on page 3527).

121771 This command need not be supported on block-mode terminals.

121772 If the cursor does not follow an **autoindent** character, or an **autoindent** character and a '0' or  
 121773 '^' character:

- 121774 1. If the cursor is in column position 1, the <control>-D character shall be discarded and no  
 121775 further action taken.
- 121776 2. Otherwise, the <control>-D character shall have no special meaning.

121777 If the last input character was a '0', the cursor shall be moved to column position 1.

121778 Otherwise, if the last input character was a '^', the cursor shall be moved to column position 1.  
 121779 In addition, the **autoindent** level for the next input line shall be derived from the same line from  
 121780 which the **autoindent** level for the current input line was derived.

121781 Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the  
 121782 *ex* **shiftwidth** command) boundary.

121783 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
 121784 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on  
 121785 page 3561).

121786 *Current line:* Unchanged.

121787 *Current column:* Set to 1 if the <control>-D was preceded by a '^' or '0'; otherwise, set to  
 121788 (column -1) - ((column -2) % **shiftwidth**).

121789 **<control>-H**

121790 *Synopsis:* <control>-H

121791 If in text input mode for a line-oriented command, and there are no characters to erase, text  
 121792 input mode shall be terminated, no further action shall be done for this command, and the  
 121793 current line and column shall be unchanged.

121794 If there are characters other than **autoindent** characters that have been input on the current line  
 121795 before the cursor, the cursor shall move back one character.

121796 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is

121797 implementation-defined whether the <control>-H command is an error or if the cursor moves  
 121798 back one **autoindent** character.

121799 Otherwise, if the cursor is in column position 1 and there are previous lines that have been  
 121800 input, it is implementation-defined whether the <control>-H command is an error or if it is  
 121801 equivalent to entering <control>-H after the last input character on the previous input line.

121802 Otherwise, it shall be an error.

121803 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
 121804 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on  
 121805 page 3561).

121806 The current erase character (see *stty*) shall cause an equivalent action to the <control>-H  
 121807 command, unless the previously inserted character was a <backslash>, in which case it shall be  
 121808 as if the literal current erase character had been inserted instead of the <backslash>.

121809 *Current line*: Unchanged, unless previously input lines are erased, in which case it shall be set to  
 121810 line -1.

121811 *Current column*: Set to the first column that displays any portion of the character backed up over.

121812 **<newline>**

121813 *Synopsis*:     <newline>  
 121814                 <carriage-return>  
 121815                 <control>-J  
 121816                 <control>-M

121817 If input was part of a line-oriented command, text input mode shall be terminated and the  
 121818 command shall continue execution with the input provided.

121819 Otherwise, terminate the current line. If there are no characters other than **autoindent** characters  
 121820 on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the  
 121821 **autoindent** characters in the line are modified by entering these characters.

121822 Continue text input mode on a new line appended after the current line. If the **slowopen** edit  
 121823 option is set, the lines on the screen below the current line shall not be pushed down, but the  
 121824 first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the  
 121825 screen below the current line shall be pushed down.

121826 If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be  
 121827 added as a prefix to the line as described by the *ex* **autoindent** edit option.

121828 All columns after the cursor that are erase-columns (as described in [Input Mode Commands in vi](#),  
 121829 on page 3561) shall be discarded.

121830 If the **autoindent** edit option is set, all <blank> characters immediately following the cursor shall  
 121831 be discarded.

121832 All remaining characters after the cursor shall be transferred to the new line, positioned after  
 121833 any **autoindent** characters.

121834 *Current line*: Set to current line +1.

121835 *Current column*: Set to the first column that displays any portion of the first character after the  
 121836 **autoindent** characters on the new line, if any, or the first column position after the last  
 121837 **autoindent** character, if any, or column position 1.

121838        **<control>-T**121839        *Synopsis:*        <control>-T121840        The <control>-T character shall have no special meaning when in text input mode for a line-oriented command (see [Command Descriptions in vi](#), on page 3527).

121841        This command need not be supported on block-mode terminals.

121842        Behave as if the user entered the minimum number of <blank> characters necessary to move the cursor forward to the column position after the next **shiftwidth** (see the *ex* **shiftwidth** command) boundary.121843        *Current line:* Unchanged.121844        *Current column:* Set to  $column + \mathbf{shiftwidth} - ((column - 1) \% \mathbf{shiftwidth})$ .121848        **<control>-U**121849        *Synopsis:*        <control>-U121850        If there are characters other than **autoindent** characters that have been input on the current line before the cursor, the cursor shall move to the first character input after the **autoindent** characters.121851        Otherwise, if there are **autoindent** characters on the current line before the cursor, it is implementation-defined whether the <control>-U command is an error or if the cursor moves to the first column position on the line.

121852        Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined whether the &lt;control&gt;-U command is an error or if it is equivalent to entering &lt;control&gt;-U after the last input character on the previous input line.

121853        Otherwise, it shall be an error.

121854        All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on page 3561).121855        The current *kill* character (see *stty*) shall cause an equivalent action to the <control>-U command, unless the previously inserted character was a <backslash>, in which case it shall be as if the literal current *kill* character had been inserted instead of the <backslash>.121856        *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to line -1.121857        *Current column:* Set to the first column that displays any portion of the last character backed up over.121866        **<control>-V**121867        *Synopsis:*        <control>-V

121868        &lt;control&gt;-Q

121869        Allow the entry of any subsequent character, other than &lt;control&gt;-J or the &lt;newline&gt;, as a literal character, removing any special meaning that it may have to the editor in text input mode. If a &lt;control&gt;-V or &lt;control&gt;-Q is entered before a &lt;control&gt;-J or &lt;newline&gt;, the &lt;control&gt;-V or &lt;control&gt;-Q character shall be discarded, and the &lt;control&gt;-J or &lt;newline&gt; shall behave as described in the &lt;newline&gt; command character during input mode.

121870        For purposes of the display only, the editor shall behave as if a '^' character was entered, and

121879 the cursor shall be positioned as if overwriting the '^' character. When a subsequent character  
 121880 is entered, the editor shall behave as if that character was entered instead of the original  
 121881 <control>-V or <control>-Q character.

121882 *Current line:* Unchanged.

121883 *Current column:* Unchanged.

#### 121884 <control>-W

121885 *Synopsis:* <control>-W

121886 If there are characters other than **autoindent** characters that have been input on the current line  
 121887 before the cursor, the cursor shall move back over the last word preceding the cursor (including  
 121888 any <blank> characters between the end of the last word and the current cursor); the cursor shall  
 121889 not move to before the first character after the end of any **autoindent** characters.

121890 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
 121891 implementation-defined whether the <control>-W command is an error or if the cursor moves to  
 121892 the first column position on the line.

121893 Otherwise, if the cursor is in column position 1 and there are previous lines that have been  
 121894 input, it is implementation-defined whether the <control>-W command is an error or if it is  
 121895 equivalent to entering <control>-W after the last input character on the previous input line.

121896 Otherwise, it shall be an error.

121897 All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
 121898 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on  
 121899 page 3561).

121900 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to  
 121901 line -1.

121902 *Current column:* Set to the first column that displays any portion of the last character backed up  
 121903 over.

#### 121904 <ESC>

121905 *Synopsis:* <ESC>

121906 If input was part of a line-oriented command:

- 121907 1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to  
 121908 command mode. The terminal shall be alerted.
- 121909 2. If <ESC> was entered, text input mode shall be terminated and the command shall  
 121910 continue execution with the input provided.

121911 Otherwise, terminate text input mode and return to command mode.

121912 Any **autoindent** characters entered on newly created lines that have no other non-<newline>  
 121913 characters shall be deleted.

121914 Any leading **autoindent** and <blank> characters on newly created lines shall be rewritten to be  
 121915 the minimum number of <blank> characters possible.

121916 The screen shall be redisplayed as necessary to match the contents of the edit buffer.

121917 *Current line:* Unchanged.

121918 *Current column:*

- 121919 1. If there are text input characters on the current line, the column shall be set to the last  
121920 column where any portion of the last text input character is displayed.
- 121921 2. Otherwise, if a character is displayed in the current column, unchanged.
- 121922 3. Otherwise, set to column position 1.

#### 121923 **EXIT STATUS**

121924 The following exit values shall be returned:

- 121925 0 Successful completion.
- 121926 >0 An error occurred.

#### 121927 **CONSEQUENCES OF ERRORS**

121928 When an unrecoverable error is encountered it shall be equivalent to a SIGHUP asynchronous  
121929 event.

121930 Otherwise, when an error is encountered, the editor shall behave as specified in [Command](#)  
121931 [Descriptions in vi](#) (on page 3527).

#### 121932 **APPLICATION USAGE**

121933 None.

#### 121934 **EXAMPLES**

121935 None.

#### 121936 **RATIONALE**

121937 See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility  
121938 specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been  
121939 implemented as a single utility, this is not required by POSIX.1-2024.

121940 It is recognized that portions of *vi* would be difficult, if not impossible, to implement  
121941 satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing,  
121942 thus it is not a mandatory requirement that such features should work on all terminals. It is the  
121943 intention, however, that a *vi* implementation should provide the full set of capabilities on all  
121944 terminals capable of supporting them.

121945 Historically, *vi* exited immediately if the standard input was not a terminal. POSIX.1-2024  
121946 permits, but does not require, this behavior. An end-of-file condition is not equivalent to an end-  
121947 of-file character. A common end-of-file character, <control>-D, is historically a *vi* command.

121948 The text in the STDOUT section reflects the usage of the verb *display* in this section; some  
121949 implementations of *vi* use standard output to write to the terminal, but POSIX.1-2024 does not  
121950 require that to be the case.

121951 Historically, implementations reverted to open mode if the terminal was incapable of supporting  
121952 full visual mode. POSIX.1-2024 requires this behavior. Historically, the open mode of *vi* behaved  
121953 roughly equivalently to the visual mode, with the exception that only a single line from the edit  
121954 buffer (one “buffer line”) was kept current at any time. This line was normally displayed on the  
121955 next-to-last line of a terminal with cursor addressing (and the last line performed its normal  
121956 visual functions for line-oriented commands and messages). In addition, some few commands  
121957 behaved differently in open mode than in visual mode. POSIX.1-2024 requires conformance to  
121958 historical practice.

121959 Historically, *ex* and *vi* implementations have expected text to proceed in the usual  
121960 European/Latin order of left to right, top to bottom. There is no requirement in POSIX.1-2024  
121961 that this be the case. The specification was deliberately written using words like “before”,

121962 ``after'', ``first'', and ``last'' in order to permit implementations to support the natural text order  
121963 of the language.

121964 Historically, lines past the end of the edit buffer were marked with single <tilde> ('~')  
121965 characters; that is, if the one-based display was 20 lines in length, and the last line of the file was  
121966 on line one, then lines 2-20 would contain only a single '~' character.

121967 Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it  
121968 did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the  
121969 bottom of the screen, the screen lines where the line would have been displayed were displayed  
121970 as single '@' characters, instead of displaying part of the line. POSIX.1-2024 permits, but does  
121971 not require, this behavior. Implementations are encouraged to attempt always to display a  
121972 complete line at the bottom of the screen when doing scrolling or screen positioning by buffer  
121973 lines.

121974 Historically, lines marked with '@' were also used to minimize output to dumb terminals over  
121975 slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen  
121976 that were not close to the cursor were simply marked with an '@' sign instead of being updated  
121977 to match the current text. POSIX.1-2024 permits, but does not require this feature because it is  
121978 used ever less frequently as terminals become smarter and connections are faster.

### 121979 Initialization in *ex* and *vi*

121980 Historically, *vi* always had a line in the edit buffer, even if the edit buffer was ``empty''. For  
121981 example:

- 121982 1. The *ex* command = executed from visual mode wrote ``1'' when the buffer was empty.
- 121983 2. Writes from visual mode of an empty edit buffer wrote files of a single character (a  
121984 <newline>), while writes from *ex* mode of an empty edit buffer wrote empty files.
- 121985 3. Put and read commands into an empty edit buffer left an empty line at the top of the edit  
121986 buffer.

121987 For consistency, POSIX.1-2024 does not permit any of these behaviors.

121988 Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was  
121989 modified if it was not originally set. POSIX.1-2024 does not permit this behavior.

### 121990 Command Descriptions in *vi*

121991 Motion commands are among the most complicated aspects of *vi* to describe. With some  
121992 exceptions, the text region and buffer type effect of a motion command on a *vi* command are  
121993 described on a case-by-case basis. The descriptions of text regions in POSIX.1-2024 are not  
121994 intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to a  
121995 region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks can  
121996 be in either direction, and, if the **wrapsan** option is set, so can movements to search points.  
121997 Historically, lines are always stored into buffers in text order; that is, from the start of the edit  
121998 buffer to the end. POSIX.1-2024 requires conformance to historical practice.

121999 Historically, command counts were applied to any associated motion, and were multiplicative to  
122000 any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as **c6w**.  
122001 POSIX.1-2024 requires this behavior. Historically, *vi* commands that used bigwords, words,  
122002 paragraphs, and sentences as objects treated groups of empty lines, or lines that contained only  
122003 <blank> characters, inconsistently. Some commands treated them as a single entity, while others  
122004 treated each line separately. For example, the **w**, **W**, and **B** commands treated groups of empty  
122005 lines as individual words; that is, the command would move the cursor to each new empty line.



122006 The **e** and **E** commands treated groups of empty lines as a single word; that is, the first use  
 122007 would move past the group of lines. The **b** command would just beep at the user, or if done from  
 122008 the start of the line as a motion command, fail in unexpected ways. If the lines contained only (or  
 122009 ended with) <blank> characters, the **w** and **W** commands would just beep at the user, the **E** and  
 122010 **e** commands would treat the group as a single word, and the **B** and **b** commands would treat the  
 122011 lines as individual words. For consistency and simplicity of specification, POSIX.1-2024 requires  
 122012 that all *vi* commands treat groups of empty or blank lines as a single entity, and that movement  
 122013 through lines ending with <blank> characters be consistent with other movements.

122014 Historically, *vi* documentation indicated that any number of double-quotes were skipped after  
 122015 punctuation marks at sentence boundaries; however, implementations only skipped single-  
 122016 quotes. POSIX.1-2024 requires both to be skipped.

122017 Historically, the first and last characters in the edit buffer were word boundaries. This historical  
 122018 practice is required by POSIX.1-2024.

122019 Historically, *vi* attempted to update the minimum number of columns on the screen possible,  
 122020 which could lead to misleading information being displayed. POSIX.1-2024 makes no  
 122021 requirements other than that the current character being entered is displayed correctly, leaving  
 122022 all other decisions in this area up to the implementation.

122023 Historically, lines were arbitrarily folded between columns of any characters that required  
 122024 multiple column positions on the screen, with the exception of tabs, which terminated at the  
 122025 right-hand margin. POSIX.1-2024 permits the former and requires the latter. Implementations  
 122026 that do not arbitrarily break lines between columns of characters that occupy multiple column  
 122027 positions should not permit the cursor to rest on a column that does not contain any part of a  
 122028 character.

122029 The historical *vi* had a problem in that all movements were by buffer lines, not by display or  
 122030 screen lines. This is often the right thing to do; for example, single line movements, such as **j** or  
 122031 **k**, should work on buffer lines. Commands like **dj**, or **j.**, where **.** is a change command, only  
 122032 make sense for buffer lines. It is not, however, the right thing to do for screen motion or scrolling  
 122033 commands like <control>-D, <control>-F, and **H**. If the window is fairly small, using buffer lines  
 122034 in these cases can result in completely random motion; for example, **1<control>-D** can result in a  
 122035 completely changed screen, without any overlap. This is clearly not what the user wanted. The  
 122036 problem is even worse in the case of the **H**, **L**, and **M** commands—as they position the cursor at  
 122037 the first non-<blank> of the line, they may all refer to the same location in large lines, and will  
 122038 result in no movement at all.

122039 In addition, if the line is larger than the screen, using buffer lines can make it impossible to  
 122040 display parts of the line—there are not any commands that do not display the beginning of the  
 122041 line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at the  
 122042 same time, the user suffers. Finally, the page and half-page scrolling commands historically  
 122043 moved to the first non-<blank> in the new line. If the line is approximately the same size as the  
 122044 screen, this is inadequate because the cursor before and after a <control>-D command will refer  
 122045 to the same location on the screen.

122046 Implementations of *ex* and *vi* exist that do not have these problems because the relevant  
 122047 commands (<control>-B, <control>-D, <control>-F, <control>-U, <control>-Y, <control>-E, **H**, **L**,  
 122048 and **M**) operate on display (screen) lines, not (edit) buffer lines.

122049 POSIX.1-2024 does not permit this behavior by default because the standard developers believed  
 122050 that users would find it too confusing. However, historical practice has been relaxed. For  
 122051 example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part of a  
 122052 line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of the  
 122053 line was displayed, and the screen lines corresponding to the line contained single '@'

122054 characters. This behavior is permitted, but not required by POSIX.1-2024, so that it is possible for  
122055 implementations to support long lines in small screens more reasonably without changing the  
122056 commands to be oriented to the display (instead of oriented to the buffer). POSIX.1-2024 also  
122057 permits implementations to refuse to edit any edit buffer containing a line that will not fit on the  
122058 screen in its entirety.

122059 The display area (for example, the value of the **window** edit option) has historically been  
122060 “grown”, or expanded, to display new text when local movements are done in displays where  
122061 the number of lines displayed is less than the maximum possible. Expansion has historically  
122062 been the first choice, when the target line is less than the maximum possible expansion value  
122063 away. Scrolling has historically been the next choice, done when the target line is less than half a  
122064 display away, and otherwise, the screen was redrawn. There were exceptions, however, in that *ex*  
122065 commands generally always caused the screen to be redrawn. POSIX.1-2024 does not specify a  
122066 standard behavior because there may be external issues, such as connection speed, the number  
122067 of characters necessary to redraw as opposed to scroll, or terminal capabilities that  
122068 implementations will have to accommodate.

122069 The current line in POSIX.1-2024 maps one-to-one to a buffer line in the file. The current column  
122070 does not. There are two different column values that are described by POSIX.1-2024. The first is  
122071 the current column value as set by many of the *vi* commands. This value is remembered for the  
122072 lifetime of the editor. The second column value is the actual position on the screen where the  
122073 cursor rests. The two are not always the same. For example, when the cursor is backed by a  
122074 multi-column character, the actual cursor position on the screen has historically been the last  
122075 column of the character in command mode, and the first column of the character in input mode.

122076 Commands that set the current line, but that do not set the current cursor value (for example, **j**  
122077 and **k**) attempt to get as close as possible to the remembered column position, so that the cursor  
122078 tends to restrict itself to a vertical column as the user moves around in the edit buffer.  
122079 POSIX.1-2024 requires conformance to historical practice, requiring that the display location of  
122080 the cursor on the display line be adjusted from the current column value as necessary to support  
122081 this historical behavior.

122082 Historically, only a single line (and for some terminals, a single line minus 1 column) of  
122083 characters could be entered by the user for the line-oriented commands; that is, **:**, **!**, **/**, or **?**.  
122084 POSIX.1-2024 permits, but does not require, this limitation.

122085 Historically, “soft” errors in *vi* caused the terminal to be alerted, but no error message was  
122086 displayed. As a general rule, no error message was displayed for errors in command execution  
122087 in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when  
122088 a searched-for object was not found. Examples of soft errors included **h** at the left margin,  
122089 **<control>-B** or **[** at the beginning of the file, **2G** at the end of the file, and so on. In addition,  
122090 errors such as **%**, **]**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well.  
122091 Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n**  
122092 displayed an error message if no previous regular expression had been specified, and **;** did not  
122093 display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in  
122094 this area might reasonably be based on a runtime evaluation of the speed of a network  
122095 connection. Finally, some implementations have provided error messages for soft errors in order  
122096 to assist naive users, based on the value of a verbose edit option. POSIX.1-2024 does not list  
122097 specific errors for which an error message shall be displayed. Implementations should conform  
122098 to historical practice in the absence of any strong reason to diverge.

122099

**Page Backwards**

122100

122101

122102

122103

122104

122105

122106

122107

122108

The <control>-B and <control>-F commands historically considered it an error to attempt to page past the beginning or end of the file, whereas the <control>-D and <control>-U commands simply moved to the beginning or end of the file. For consistency, POSIX.1-2024 requires the latter behavior for all four commands. All four commands still consider it an error if the current line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of the file. Historically, the <control>-B and <control>-F commands skip two lines in order to include overlapping lines when a single command is entered. This makes less sense in the presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation used by historical implementations of the *vi* editor for <control>-B was:

122109

```
((current first line) - count x (window edit option)) +2
```

122110

and for <control>-F was:

122111

```
((current first line) + count x (window edit option)) -2
```

122112

122113

122114

122115

This calculation does not work well when intermixing commands with and without counts; for example, 3<control>-F is not equivalent to entering the <control>-F command three times, and is not reversible by entering the <control>-B command three times. For consistency with other *vi* commands that take counts, POSIX.1-2024 requires a different calculation.

122116

**Scroll Forward**

122117

122118

The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll** command. 4BSD used:

122119

```
((window edit option) +1) /2
```

122120

122121

122122

122123

while System V used the value of the **scroll** edit option. The System V version is specified by POSIX.1-2024 because the standard developers believed that it was more intuitive and permitted the user a method of setting the scroll value initially without also setting the number of lines that are displayed.

122124

**Scroll Forward by Line**

122125

122126

122127

122128

122129

Historically, the <control>-E and <control>-Y commands considered it an error if the last and first lines, respectively, were already on the screen. POSIX.1-2024 requires conformance to historical practice. Historically, the <control>-E and <control>-Y commands had no effect in open mode. For simplicity and consistency of specification, POSIX.1-2024 requires that they behave as usual, albeit with a single line screen.

122130

**Clear and Redisplay**

122131

122132

122133

122134

122135

The historical <control>-L command refreshed the screen exactly as it was supposed to be currently displayed, replacing any '@' characters for lines that had been deleted but not updated on the screen with refreshed '@' characters. The intent of the <control>-L command is to refresh when the screen has been accidentally overwritten; for example, by a **write** command from another user, or modem noise.

### 122136 **Redraw Screen**

122137 The historical <control>-R command redisplayed only when necessary to update lines that had  
 122138 been deleted but not updated on the screen and that were flagged with '@' characters. There is  
 122139 no requirement that the screen be in any way refreshed if no lines of this form are currently  
 122140 displayed. POSIX.1-2024 permits implementations to extend this command to refresh lines on  
 122141 the screen flagged with '@' characters because they are too long to be displayed in the current  
 122142 framework; however, the current line and column need not be modified.

### 122143 **Search for tagstring**

122144 Historically, the first non-<blank> at or after the cursor was the first character, and all  
 122145 subsequent characters that were word characters, up to the end of the line, were included. For  
 122146 example, with the cursor on the leading <space> or on the '#' character in the text "#bar@",  
 122147 the tag was "#bar". On the character 'b' it was "bar", and on the 'a' it was "ar".  
 122148 POSIX.1-2024 requires this behavior.

### 122149 **Replace Text with Results from Shell Command**

122150 Historically, the <, >, and ! commands considered most cursor motions other than line-oriented  
 122151 motions an error; for example, the command >/foo<CR> succeeded, while the command >I  
 122152 failed, even though the text region described by the two commands might be identical. For  
 122153 consistency, all three commands only consider entire lines and not partial lines, and the region is  
 122154 defined as any line that contains a character that was specified by the motion.

### 122155 **Move to Matching Character**

122156 Other matching characters have been left implementation-defined in order to allow extensions  
 122157 such as matching '<' and '>' for searching HTML, or #ifdef, #else, and #endif for searching C  
 122158 source.

### 122159 **Repeat Substitution**

122160 POSIX.1-2024 requires that any c and g flags specified to the previous substitute command be  
 122161 ignored; however, the r flag may still apply, if supported by the implementation.

### 122162 **Return to Previous (Context or Section)**

122163 The [, ], (, ), {, and } commands are all affected by "section boundaries", but in some historical  
 122164 implementations not all of the commands recognize the same section boundaries. This is a bug,  
 122165 not a feature, and a unique section-boundary algorithm was not described for each command.  
 122166 One special case that is preserved is that the sentence command moves to the end of the last line  
 122167 of the edit buffer while the other commands go to the beginning, in order to preserve the  
 122168 traditional character cut semantics of the sentence command. Historically, vi section boundaries  
 122169 at the beginning and end of the edit buffer were the first non-<blank> on the first and last lines  
 122170 of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit  
 122171 buffer if one exists. To increase consistency with other section locations, this has been simplified  
 122172 by POSIX.1-2024 to the first character of the first and last lines of the edit buffer, or the first and  
 122173 the last lines of the edit buffer if they are empty.

122174 Sentence boundaries were problematic in the historical vi. They were not only the boundaries as  
 122175 defined for the section and paragraph commands, but they were the first non-<blank> that  
 122176 occurred after those boundaries, as well. Historically, the vi section commands were  
 122177 documented as taking an optional window size as a count preceding the command. This was not  
 122178 implemented in historical versions, so POSIX.1-2024 requires that the count repeat the command,

122179 for consistency with other *vi* commands.

### 122180 **Repeat**

122181 Historically, mapped commands other than text input commands could not be repeated using  
122182 the **period** command. POSIX.1-2024 requires conformance to historical practice.

122183 The restrictions on the interpretation of special characters (for example, <control>-H) in the  
122184 repetition of text input mode commands is intended to match historical practice. For example,  
122185 given the input sequence:

```
122186 iab<control>-H<control>-H<control>-Hdef<escape>
```

122187 the user should be informed of an error when the sequence is first entered, but not during a  
122188 command repetition. The character <control>-T is specifically exempted from this restriction.  
122189 Historical implementations of *vi* ignored <control>-T characters that were input in the original  
122190 command during command repetition. POSIX.1-2024 prohibits this behavior.

### 122191 **Find Regular Expression**

122192 Historically, commands did not affect the line searched to or from if the motion command was a  
122193 search (*/*, *?*, **N**, **n**) and the final position was the start/end of the line. There were some special  
122194 cases and *vi* was not consistent. POSIX.1-2024 does not permit this behavior, for consistency.  
122195 Historical implementations permitted but were unable to handle searches as motion commands  
122196 that wrapped (that is, due to the edit option **wrapsan**) to the original location. POSIX.1-2024  
122197 requires that this behavior be treated as an error.

122198 Historically, the syntax `"/RE/0"` was used to force the command to cut text in line mode.  
122199 POSIX.1-2024 requires conformance to historical practice.

122200 Historically, in open mode, a **z** specified to a search command redisplayed the current line  
122201 instead of displaying the current screen with the current line highlighted. For consistency and  
122202 simplicity of specification, POSIX.1-2024 does not permit this behavior.

122203 Historically, trailing **z** commands were permitted and ignored if entered as part of a search used  
122204 as a motion command. For consistency and simplicity of specification, POSIX.1-2024 does not  
122205 permit this behavior.

### 122206 **Execute an ex Command**

122207 Historically, *vi* implementations restricted the commands that could be entered on the colon  
122208 command line (for example, **append** and **change**), and some other commands were known to  
122209 cause them to fail catastrophically. For consistency, POSIX.1-2024 does not permit these  
122210 restrictions. When executing an *ex* command by entering `:`, it is not possible to enter a <newline>  
122211 as part of the command because it is considered the end of the command. A different approach  
122212 is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual mode with  
122213 the *ex* **vi** command). In *ex* command mode, the single-line limitation does not exist. So, for  
122214 example, the following is valid:

```
122215 Q
122216 s/break here/break\
122217 here/
122218 vi
```

122219 POSIX.1-2024 requires that, if the *ex* command overwrites any part of the screen that would be  
122220 erased by a refresh, *vi* pauses for a character from the user. Historically, this character could be  
122221 any character; for example, a character input by the user before the message appeared, or even a

122222 mapped character. This is probably a bug, but implementations that have tried to be more  
122223 rigorous by requiring that the user enter a specific character, or that the user enter a character  
122224 after the message was displayed, have been forced by user indignation back into historical  
122225 behavior. POSIX.1-2024 requires conformance to historical practice.

#### 122226 **Shift Left (Right)**

122227 Refer to the Rationale for the **!** and **/** commands. Historically, the **<** and **>** commands sometimes  
122228 moved the cursor to the first non-**<blank>** (for example if the command was repeated or with **\_**  
122229 as the motion command), and sometimes left it unchanged. POSIX.1-2024 does not permit this  
122230 inconsistency, requiring instead that the cursor always move to the first non-**<blank>**.  
122231 Historically, the **<** and **>** commands did not support buffer arguments, although some  
122232 implementations allow the specification of an optional buffer. This behavior is neither required  
122233 nor disallowed by POSIX.1-2024.

#### 122234 **Execute**

122235 Historically, buffers could execute other buffers, and loops, infinite and otherwise, were  
122236 possible. POSIX.1-2024 requires conformance to historical practice. The *\*buffer* syntax of *ex* is not  
122237 required in *vi*, because it is not historical practice and has been used in some *vi* implementations  
122238 to support additional scripting languages.

#### 122239 **Reverse Case**

122240 Historically, the **~** command ignored any associated *count*, and acted only on the characters in the  
122241 current line. For consistency with other *vi* commands, POSIX.1-2024 requires that an associated  
122242 *count* act on the next *count* characters, and that the command move to subsequent lines if  
122243 warranted by *count*, to make it possible to modify large pieces of text in a reasonably efficient  
122244 manner. There exist *vi* implementations that optionally require an associated motion command  
122245 for the **~** command. Implementations supporting this functionality are encouraged to base it on  
122246 the **tildedop** edit option and handle the text regions and cursor positioning identically to the  
122247 **yank** command.

#### 122248 **Append**

122249 Historically, *counts* specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line  
122250 *count* times, and did not repeat the subsequent lines of the input text. POSIX.1-2024 requires that  
122251 the entire text input be repeated *count* times.

#### 122252 **Move Backward to Preceding Word**

122253 Historically, *vi* became confused if word commands were used as motion commands in empty  
122254 files. POSIX.1-2024 requires that this be an error. Historical implementations of *vi* had a large  
122255 number of bugs in the word movement commands, and they varied greatly in behavior in the  
122256 presence of empty lines, “words” made up of a single character, and lines containing only  
122257 **<blank>** characters. For consistency and simplicity of specification, POSIX.1-2024 does not  
122258 permit this behavior.

**122259 Change to End-of-Line**

122260 Some historical implementations of the **C** command did not behave as described by  
122261 POSIX.1-2024 when the **\$** key was remapped because they were implemented by pushing the **\$**  
122262 key onto the input queue and reprocessing it. POSIX.1-2024 does not permit this behavior.  
122263 Historically, the **C**, **S**, and **s** commands did not copy replaced text into the numeric buffers. For  
122264 consistency and simplicity of specification, POSIX.1-2024 requires that they behave like their  
122265 respective **c** commands in all respects.

**122266 Delete**

122267 Historically, lines in open mode that were deleted were scrolled up, and an **@** glyph written over  
122268 the beginning of the line. In the case of terminals that are incapable of the necessary cursor  
122269 motions, the editor erased the deleted line from the screen. POSIX.1-2024 requires conformance  
122270 to historical practice; that is, if the terminal cannot display the **'@'** character, the line cannot  
122271 remain on the screen.

**122272 Delete to End-of-Line**

122273 Some historical implementations of the **D** command did not behave as described by  
122274 POSIX.1-2024 when the **\$** key was remapped because they were implemented by pushing the **\$**  
122275 key onto the input queue and reprocessing it. POSIX.1-2024 does not permit this behavior.

**122276 Join**

122277 An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent. POSIX.1-2024  
122278 requires conformance to historical practice. The *vi* **J** command is specified in terms of the *ex* **join**  
122279 command with an *ex* command *count* value. The address correction for a *count* that is past the  
122280 end of the edit buffer is necessary for historical compatibility for both *ex* and *vi*.

**122281 Mark Position**

122282 Historical practice is that only lowercase letters, plus backquote and single-quote, could be used  
122283 to mark a cursor position. POSIX.1-2024 requires conformance to historical practice, but  
122284 encourages implementations to support other characters as marks as well.

**122285 Repeat Regular Expression Find (Forward and Reverse)**

122286 Historically, the **N** and **n** commands could not be used as motion components for the **c**  
122287 command. With the exception of the **cN** command, which worked if the search crossed a line  
122288 boundary, the text region would be discarded, and the user would not be in text input mode. For  
122289 consistency and simplicity of specification, POSIX.1-2024 does not permit this behavior.

**122290 Insert Empty Line (Below and Above)**

122291 Historically, counts to the **O** and **o** commands were used as the number of physical lines to  
122292 open, if the terminal was dumb and the **slowopen** option was not set. This was intended to  
122293 minimize traffic over slow connections and repainting for dumb terminals. POSIX.1-2024 does  
122294 not permit this behavior, requiring that a *count* to the open command behave as for other text  
122295 input commands. This change to historical practice was made for consistency, and because a  
122296 superset of the functionality is provided by the **slowopen** edit option.

### 122297 **Put from Buffer (Following and Before)**

122298 Historically, *counts* to the **p** and **P** commands were ignored if the buffer was a line mode buffer,  
 122299 but were (mostly) implemented as described in POSIX.1-2024 if the buffer was a character mode  
 122300 buffer. Because implementations exist that do not have this limitation, and because pasting lines  
 122301 multiple times is generally useful, POSIX.1-2024 requires that *count* be supported for all **p** and **P**  
 122302 commands.

122303 Historical implementations of *vi* were widely known to have major problems in the **p** and **P**  
 122304 commands, particularly when unusual regions of text were copied into the edit buffer. The  
 122305 standard developers viewed these as bugs, and they are not permitted for consistency and  
 122306 simplicity of specification.

122307 Historically, a **P** or **p** command (or an *ex put* command executed from open or visual mode)  
 122308 executed in an empty file, left an empty line as the first line of the file. For consistency and  
 122309 simplicity of specification, POSIX.1-2024 does not permit this behavior.

### 122310 **Replace Character**

122311 Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as  
 122312 arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return>  
 122313 argument, for which it replaced *count* characters with a single <newline>. POSIX.1-2024 does  
 122314 not permit these inconsistencies.

122315 Historically, the **r** command permitted the <control>-V escaping of entered characters, such as  
 122316 <ESC> and the <carriage-return>; however, it required two leading <control>-V characters  
 122317 instead of one. POSIX.1-2024 requires that this be changed for consistency with the other text  
 122318 input commands of *vi*.

122319 Historically, it is an error to enter the **r** command if there are less than *count* characters at or after  
 122320 the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r**  
 122321 command on empty lines, it would require that too large a *count* be adjusted to match the  
 122322 number of characters at or after the cursor for consistency, which is sufficiently different from  
 122323 historical practice to be avoided. POSIX.1-2024 requires conformance to historical practice.

### 122324 **Replace Characters**

122325 Historically, if there were **autoindent** characters in the line on which the **R** command was run,  
 122326 and **autoindent** was set, the first <newline> would be properly indented and no characters  
 122327 would be replaced by the <newline>. Each additional <newline> would replace *n* characters,  
 122328 where *n* was the number of characters that were needed to indent the rest of the line to the  
 122329 proper indentation level. This behavior is a bug and is not permitted by POSIX.1-2024.

### 122330 **Undo**

122331 Historical practice for cursor positioning after undoing commands was mixed. In most cases,  
 122332 when undoing commands that affected a single line, the cursor was moved to the start of added  
 122333 or changed text, or immediately after deleted text. However, if the user had moved from the line  
 122334 being changed, the column was either set to the first non-<blank>, returned to the origin of the  
 122335 command, or remained unchanged. When undoing commands that affected multiple lines or  
 122336 entire lines, the cursor was moved to the first character in the first line restored. As an example  
 122337 of how inconsistent this was, a search, followed by an **o** text input command, followed by an  
 122338 **undo** would return the cursor to the location where the **o** command was entered, but a **cw**  
 122339 command followed by an **o** command followed by an **undo** would return the cursor to the first  
 122340 non-<blank> of the line. POSIX.1-2024 requires the most useful of these behaviors, and discards  
 122341 the least useful, in the interest of consistency and simplicity of specification.



122342 **Yank**

122343 Historically, the **yank** command did not move to the end of the motion if the motion was in the  
 122344 forward direction. It moved to the end of the motion if the motion was in the backward  
 122345 direction, except for the **\_** command, or for the **G** and **'** commands when the end of the motion  
 122346 was on the current line. This was further complicated by the fact that for a number of motion  
 122347 commands, the **yank** command moved the cursor but did not update the screen; for example, a  
 122348 subsequent command would move the cursor from the end of the motion, even though the  
 122349 cursor on the screen had not reflected the cursor movement for the **yank** command.  
 122350 POSIX.1-2024 requires that all **yank** commands associated with backward motions move the  
 122351 cursor to the end of the motion for consistency, and specifically, to make **'** commands as motions  
 122352 consistent with search patterns as motions.

122353 **Yank Current Line**

122354 Some historical implementations of the **Y** command did not behave as described by  
 122355 POSIX.1-2024 when the **'\_'** key was remapped because they were implemented by pushing the  
 122356 **'\_'** key onto the input queue and reprocessing it. POSIX.1-2024 does not permit this behavior.

122357 **Redraw Window**

122358 Historically, the **z** command always redrew the screen. This is permitted but not required by  
 122359 POSIX.1-2024, because of the frequent use of the **z** command in macros such as **map n nz.** for  
 122360 screen positioning, instead of its use to change the screen size. The standard developers  
 122361 believed that expanding or scrolling the screen offered a better interface for users. The ability to  
 122362 redraw the screen is preserved if the optional new window size is specified, and in the  
 122363 **<control>-L** and **<control>-R** commands.

122364 The semantics of **z^** are confusing at best. Historical practice is that the screen before the screen  
 122365 that ended with the specified line is displayed. POSIX.1-2024 requires conformance to historical  
 122366 practice.

122367 Historically, the **z** command would not display a partial line at the top or bottom of the screen. If  
 122368 the partial line would normally have been displayed at the bottom of the screen, the command  
 122369 worked, but the partial line was replaced with **'@'** characters. If the partial line would normally  
 122370 have been displayed at the top of the screen, the command would fail. For consistency and  
 122371 simplicity of specification, POSIX.1-2024 does not permit this behavior.

122372 Historically, the **z** command with a line specification of 1 ignored the command. For consistency  
 122373 and simplicity of specification, POSIX.1-2024 does not permit this behavior.

122374 Historically, the **z** command did not set the cursor column to the first non-**<blank>** for the  
 122375 character if the first screen was to be displayed, and was already displayed. For consistency and  
 122376 simplicity of specification, POSIX.1-2024 does not permit this behavior.

122377 **Input Mode Commands in vi**

122378 Historical implementations of **vi** did not permit the user to erase more than a single line of input,  
 122379 or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent**  
 122380 characters. As there exist implementations of **vi** that do not have these limitations, both  
 122381 behaviors are permitted, but only historical practice is required. In the case of these extensions,  
 122382 **vi** is required to pause at the **autoindent** and previous line boundaries.

122383 Historical implementations of **vi** updated only the portion of the screen where the current cursor  
 122384 character was displayed. For example, consider the **vi** input keystrokes:

122385 `iabcd<escape>0C<tab>`

122386 Historically, the <tab> would overwrite the characters "abcd" when it was displayed. Other  
 122387 implementations replace only the 'a' character with the <tab>, and then push the rest of the  
 122388 characters ahead of the cursor. Both implementations have problems. The historical  
 122389 implementation is probably visually nicer for the above example; however, for the keystrokes:

122390 iabcd<ESC>OR<tab><ESC>

122391 the historical implementation results in the string "bcd" disappearing and then magically  
 122392 reappearing when the <ESC> character is entered. POSIX.1-2024 requires the former behavior  
 122393 when overwriting erase-columns—that is, overwriting characters that are no longer logically  
 122394 part of the edit buffer—and the latter behavior otherwise.

122395 Historical implementations of *vi* discarded the <control>-D and <control>-T characters when  
 122396 they were entered at places where their command functionality was not appropriate.  
 122397 POSIX.1-2024 requires that the <control>-T functionality always be available, and that  
 122398 <control>-D be treated as any other key when not operating on **autoindent** characters.

## 122399 NUL

122400 Some historical implementations of *vi* limited the number of characters entered using the NUL  
 122401 input character to 256 bytes. POSIX.1-2024 permits this limitation; however, implementations are  
 122402 encouraged to remove this limit.

## 122403 <control>-D

122404 See also Rationale for the input mode command <newline>. The hidden assumptions in the  
 122405 <control>-D command (and in the *vi* **autoindent** specification in general) is that <space>  
 122406 characters take up a single column on the screen and that <tab> characters are comprised of an  
 122407 integral number of <space> characters.

## 122408 <newline>

122409 Implementations are permitted to rewrite **autoindent** characters in the line when <newline>,  
 122410 <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are  
 122411 used, because historical implementations have both done so and found it necessary to do so. For  
 122412 example, a <control>-D when the cursor is preceded by a single <tab>, with **tabstop** set to 8, and  
 122413 **shiftwidth** set to 3, will result in the <tab> being replaced by several <space> characters.

## 122414 <control>-T

122415 See also the Rationale for the input mode command <newline>. Historically, <control>-T only  
 122416 worked if no non-<blank> characters had yet been input in the current input line. In addition,  
 122417 the characters inserted by <control>-T were treated as **autoindent** characters, and could not be  
 122418 erased using normal user erase characters. Because implementations exist that do not have  
 122419 these limitations, and as moving to a column boundary is generally useful, POSIX.1-2024  
 122420 requires that both limitations be removed.

- 122421           **<control>-V**
- 122422           Historically, *vi* used **^V**, regardless of the value of the literal-next character of the terminal.  
122423           POSIX.1-2024 requires conformance to historical practice.
- 122424           The uses described for **<control>-V** can also be accomplished with **<control>-Q**, which is useful  
122425           on terminals that use **<control>-V** for the down-arrow function. However, most historical  
122426           implementations use **<control>-Q** for the *termios* START character, so the editor will generally  
122427           not receive the **<control>-Q** unless **stty ixon** mode is set to off. (In addition, some historical  
122428           implementations of *vi* explicitly set **ixon** mode to on, so it was difficult for the user to set it to  
122429           off.) Any of the command characters described in POSIX.1-2024 can be made ineffective by their  
122430           selection as *termios* control characters, using the *stty* utility or other methods described in the  
122431           System Interfaces volume of POSIX.1-2024.
- 122432           **<ESC>**
- 122433           Historically, SIGINT alerted the terminal when used to end input mode. This behavior is  
122434           permitted, but not required, by POSIX.1-2024.
- 122435           **FUTURE DIRECTIONS**
- 122436           If this utility is directed to create a new directory entry that contains any bytes that have the  
122437           encoded value of a **<newline>** character, implementations are encouraged to treat this as an  
122438           error. A future version of this standard may require implementations to treat this as an error.
- 122439           **SEE ALSO**
- 122440           *ed, ex, stty*
- 122441           XBD [Section 12.2](#) (on page 215)
- 122442           **CHANGE HISTORY**
- 122443           First released in Issue 2.
- 122444           **Issue 5**
- 122445           The FUTURE DIRECTIONS section is added.
- 122446           **Issue 6**
- 122447           This utility is marked as part of the User Portability Utilities option.
- 122448           The APPLICATION USAGE section is added.
- 122449           The obsolescent SYNOPSIS is removed.
- 122450           The following new requirements on POSIX implementations derive from alignment with the  
122451           Single UNIX Specification:
- 122452
  - The **reindent** command description is added.

122453           The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft  
122454           standard.

122455           IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied.

122456           IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the **R** command in  
122457           a manner similar to the descriptions of other text input mode commands such as **i**, **o**, and **O**.

122458           The **-l** option is removed.

122459           IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/41 is applied, adding *[count]* to the  
122460           Synopsis for **[[**.

122461           IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/42 is applied, adding *[count]* to the

- 122462 Synopsis for `ll`.
- 122463 **Issue 7**
- 122464 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized  
122465 as an option delimiter in the `OPTIONS` section.
- 122466 Austin Group Interpretation 1003.1-2001 #087 is applied, updating the `Put from Buffer Before (P)`  
122467 command description to address multi-line requirements.
- 122468 `SD5-XCU-ERN-97` is applied, updating the `SYNOPSIS`.
- 122469 `POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0202 [812]` is applied.
- 122470 **Issue 8**
- 122471 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
122472 filenames containing any bytes that have the encoded value of a `<newline>` character.
- 122473 Austin Group Defect 1310 is applied, changing the `CONSEQUENCES OF ERRORS` section.
- 122474 Austin Group Defect 1676 is applied, changing the typeface of some bold text in the  
122475 `RATIONALE` section.

122476 **NAME**

122477           wait — await process completion

122478 **SYNOPSIS**122479           wait [*pid*...]122480 **DESCRIPTION**122481           The *wait* utility shall wait for one or more child processes whose process IDs are known in the  
122482           current shell execution environment (see [Section 2.13](#), on page 2522) to terminate.122483           If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the  
122484           invoking shell have terminated and exit with a zero exit status.122485           If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall  
122486           wait until all of them have terminated. If one or more *pid* operands are specified that represent  
122487           unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with  
122488           exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process  
122489           requested by the last *pid* operand.122490           Once a process ID that is known in the current shell execution environment (see [Section 2.13](#), on  
122491           page 2522) has been successfully waited for, it shall be removed from the list of process IDs that  
122492           are known in the current shell execution environment. If the process ID is associated with a  
122493           background job, the corresponding job shall also be removed from the list of background jobs.122494 **OPTIONS**

122495           None.

122496 **OPERANDS**

122497           The following operand shall be supported:

122498           *pid*           One of the following:

- 122499                           1. The unsigned decimal integer process ID of a child process whose  
122500                           termination the utility is to wait for.
- 122501                           2. A job ID (see [XBD Section 3.182](#), on page 57) that identifies a process group  
122502                           in the case of a job-control background job, or a process ID in the case of a  
122503                           non-job-control background job (if supported), to be waited for. The job ID  
122504                           notation is applicable only for invocations of *wait* in the current shell  
122505                           execution environment; see [Section 2.13](#) (on page 2522). The exit status of  
122506                           *wait* shall be determined by the exit status of the last pipeline to be executed.

122507           **Note:**       The job ID type of *pid* is only available on systems supporting the User  
122508                           Portability Utilities option or supporting non-job-control background  
122509                           jobs.

122510 **STDIN**

122511           Not used.

122512 **INPUT FILES**

122513           None.

122514 **ENVIRONMENT VARIABLES**122515           The following environment variables shall affect the execution of *wait*:

122516           *LANG*       Provide a default value for the internationalization variables that are unset or null.  
122517                           (See [XBD Section 8.2](#) (on page 169) for the precedence of internationalization  
122518                           variables used to determine the values of locale categories.)

|        |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 122519 | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 122520 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122521 | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 122522 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122523 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122524 | <i>LC_MESSAGES</i>            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122525 |                               | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 122526 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122527 | XSI <i>NLSPATH</i>            | Determine the location of messages objects and message catalogs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 122528 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122529 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 122530 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122531 |                               | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 122532 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122533 |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 122534 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122535 |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 122536 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122537 |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 122538 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122539 |                               | If one or more operands were specified, all of them have terminated or were not known in the invoking shell execution environment, and the status of the last operand specified is known, then the exit status of <i>wait</i> shall be the status of the last operand specified. If the process terminated abnormally due to the receipt of a signal, the exit status shall be greater than 128 and shall be distinct from the exit status generated by other signals, but the exact value is unspecified. (See the <i>kill -l</i> option.) Otherwise, the <i>wait</i> utility shall exit with one of the following values: |
| 122540 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122541 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122542 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122543 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122544 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122545 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122546 | 0                             | The <i>wait</i> utility was invoked with no operands and all process IDs known by the invoking shell have terminated.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 122547 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122548 | 1-126                         | The <i>wait</i> utility detected an error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 122549 | 127                           | The process ID specified by the last <i>pid</i> operand specified is not known in the invoking shell execution environment.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 122550 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122551 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122552 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 122553 | <b>APPLICATION USAGE</b>      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122554 |                               | This utility is required to be intrinsic. See <a href="#">Section 1.7</a> (on page 2470) for details.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 122555 |                               | On most implementations, <i>wait</i> is a shell built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 122556 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 122557 |                               | (wait)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 122558 |                               | nohup wait ...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 122559 |                               | find . -exec wait ... \;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 122560 |                               | it returns immediately because there are no known process IDs to wait for in those environments.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 122561 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

122562 The use of job ID notation is not dependent on job control being enabled. When job control has  
 122563 been disabled using *set +m*, *wait* can still be used to wait for the process group associated with a  
 122564 job-control background job, or the process ID associated with a non-control background job (if  
 122565 supported), using

```
122566 wait %<background job number>
```

122567 See also the RATIONALE for *jobs* and *kill*.

122568 The shell is allowed to discard the status of any process if it determines that the application  
 122569 cannot get the process ID for that process from the shell. It is also required to remember only  
 122570 {CHILD\_MAX} number of processes in this way. Since the only way to get the process ID from  
 122571 the shell is by using the '!' shell parameter, the shell is allowed to discard the status of an  
 122572 asynchronous AND-OR list if "\$!" was not referenced before another asynchronous AND-OR  
 122573 list was started. (This means that the shell only has to keep the status of the last asynchronous  
 122574 AND-OR list started if the application did not reference "\$!". If the implementation of the shell  
 122575 is smart enough to determine that a reference to "\$!" was not saved anywhere that the  
 122576 application can retrieve it later, it can use this information to trim the list of saved information.  
 122577 Note also that a successful call to *wait* with no operands discards the exit status of all  
 122578 asynchronous AND-OR lists.)

122579 If the exit status of *wait* is greater than 128, there is no way for the application to know if the  
 122580 waited-for process exited with that value or was killed by a signal. Since most utilities exit with  
 122581 small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just  
 122582 need to know that the asynchronous job failed; it does not matter whether it detected an error  
 122583 and failed or was killed and did not complete its job normally.

122584 Some historical shells returned from *wait* when a process stops instead of only when it  
 122585 terminates. This standard does not allow *wait* to return when a process stops for two reasons:

- 122586 1. The vast majority, if not all, shell scripts that use *wait* (without using an extension) expect  
 122587 it not to return until the process terminates.
- 122588 2. It is not possible to write a portable shell script that can correctly handle *wait* returning  
 122589 when a process stops, because an exit status indicating a process was stopped by a signal  
 122590 cannot be distinguished from one indicating that the process called *exit()* with the same  
 122591 value.

122592 The standard developers considered allowing interactive shells to return from *wait* when a  
 122593 process stops, since the interactive user would see a message which would allow them to tell  
 122594 whether the process stopped or terminated. However, they decided that it would be inadvisable  
 122595 to introduce an inconsistency between interactive and non-interactive shells, particularly as the  
 122596 most likely use of *wait* in an interactive shell is to try out commands before putting them in a  
 122597 shell script. Implementations can provide an extension that could be used to request that *wait*  
 122598 returns when a process stops. It is recommended that any such extension uses a different  
 122599 method of returning information about the wait status of the process so that the information can  
 122600 be unambiguous. One suitable method would be an option that takes a variable name as an  
 122601 option-argument. The named variable would be set to a numeric value and the exit status of  
 122602 wait would indicate whether this value is an exit value or a signal number, and whether the  
 122603 signal terminated the process or stopped it. Such an extension would also provide a way for  
 122604 shell scripts to obtain the full exit value (as would be returned by *waitid()*).

## 122605 EXAMPLES

122606 Although the exact value used when a process is terminated by a signal is unspecified, if it is  
 122607 known that a signal terminated a process, a script can still reliably determine which signal by  
 122608 using *kill* as shown by the following script:

```

122609     sleep 1000&
122610     pid=$!
122611     kill -kill $pid
122612     wait $pid
122613     echo $pid was terminated by a SIG$(kill -l $?) signal.

```

122614 If the following sequence of commands is run in less than 31 seconds:

```

122615     sleep 257 | sleep 31 &
122616     jobs -l %%

```

122617 either of the following commands returns the exit status of the second *sleep* in the pipeline:

```

122618     wait <pid of sleep 31>
122619     wait %%

```

#### 122620 RATIONALE

122621 The description of *wait* does not refer to the *waitpid()* function from the System Interfaces  
 122622 volume of POSIX.1-2024 because that would needlessly overspecify this interface. However, the  
 122623 wording means that *wait* is required to wait for an explicit process when it is given an argument  
 122624 so that the status information of other processes is not consumed. Historical implementations  
 122625 use the *wait()* function defined in the System Interfaces volume of POSIX.1-2024 until *wait()*  
 122626 returns the requested process ID or finds that the requested process does not exist. Because this  
 122627 means that a shell script could not reliably get the status of all background children if a second  
 122628 background job was ever started before the first job finished, it is recommended that the *wait*  
 122629 utility use a method such as the functionality provided by the *waitpid()* function.

122630 The ability to wait for multiple *pid* operands was adopted from the KornShell.

122631 This new functionality was added because it is needed to determine the exit status of any  
 122632 asynchronous AND-OR list accurately. The only compatibility problem that this change creates  
 122633 is for a script like

```

122634     while sleep 60 do
122635         job& echo Job started $(date) as $! done

```

122636 which causes the shell to monitor all of the jobs started until the script terminates or runs out of  
 122637 memory. This would not be a problem if the loop did not reference "\$!" or if the script would  
 122638 occasionally *wait* for jobs it started.

#### 122639 FUTURE DIRECTIONS

122640 A future version of this standard may add an option which takes a variable name as an option-  
 122641 argument, allowing *wait* to return information about the wait status of a process in an  
 122642 unambiguous way.

#### 122643 SEE ALSO

122644 [Chapter 2](#) (on page 2472), *kill*, *sh*  
 122645 [XBD Section 3.182](#) (on page 57), [Chapter 8](#) (on page 167)  
 122646 XSH *wait()*

#### 122647 CHANGE HISTORY

122648 First released in Issue 2.

#### 122649 Issue 8

122650 Austin Group Defect 854 is applied, adding a note to the APPLICATION USAGE section that  
 122651 this utility is required to be intrinsic.

122652 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.



122653  
122654

Austin Group Defect 1254 is applied, updating various requirements for the *jobs* utility to account for the addition of [Section 2.11](#) (on page 2518).

122655 **NAME**122656 `wc` — word, line, and byte or character count122657 **SYNOPSIS**122658 `wc [-c|-m] [-lw] [file...]`122659 **DESCRIPTION**122660 The *wc* utility shall read one or more input files and, by default, write the number of <newline>  
122661 characters, words, and bytes contained in each input file to the standard output.122662 The utility also shall write a total count for all named files, if more than one input file is  
122663 specified.122664 The *wc* utility shall consider a *word* to be a non-zero-length string of characters delimited by  
122665 white space.122666 **OPTIONS**122667 The *wc* utility shall conform to XBD [Section 12.2](#) (on page 215).

122668 The following options shall be supported:

122669 `-c` Write to the standard output the number of bytes in each input file.122670 `-l` Write to the standard output the number of <newline> characters in each input  
122671 file.122672 `-m` Write to the standard output the number of characters in each input file.122673 `-w` Write to the standard output the number of words in each input file.122674 When any option is specified, *wc* shall report only the information requested by the specified  
122675 options.122676 **OPERANDS**

122677 The following operand shall be supported:

122678 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
122679 shall be used.122680 **STDIN**122681 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
122682 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
122683 the standard input shall not be used. See the INPUT FILES section.122684 **INPUT FILES**

122685 The input files may be of any type.

122686 **ENVIRONMENT VARIABLES**122687 The following environment variables shall affect the execution of *wc*:122688 *LANG* Provide a default value for the internationalization variables that are unset or null.  
122689 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
122690 variables used to determine the values of locale categories.)122691 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
122692 internationalization variables.122693 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
122694 characters (for example, single-byte as opposed to multi-byte characters in  
122695 arguments and input files) and which characters are defined as white-space  
122696 characters.

- 122697 **LC\_MESSAGES**
- 122698 Determine the locale that should be used to affect the format and contents of
- 122699 diagnostic messages written to standard error and informative messages written to
- 122700 standard output.
- 
- 122701 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 122702 **ASYNCHRONOUS EVENTS**
- 122703 Default.
- 122704 **STDOUT**
- 122705 By default, the standard output shall contain an entry for each input file of the form:
- 122706 "%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>
- 122707 If the **-m** option is specified, the number of characters shall replace the <bytes> field in this
- 122708 format.
- 122709 If any options are specified and the **-l** option is not specified, the number of <newline>
- 122710 characters shall not be written.
- 122711 If any options are specified and the **-w** option is not specified, the number of words shall not be
- 122712 written.
- 122713 If any options are specified and neither **-c** nor **-m** is specified, the number of bytes or characters
- 122714 shall not be written.
- 122715 If no input *file* operands are specified, no name shall be written and no <blank> characters
- 122716 preceding the pathname shall be written.
- 122717 If more than one input *file* operand is specified, an additional line shall be written, of the same
- 122718 format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead
- 122719 of a pathname and the total of each column shall be written as appropriate. Such an additional
- 122720 line, if any, is written at the end of the output.
- 122721 **STDERR**
- 122722 The standard error shall be used only for diagnostic messages.
- 122723 **OUTPUT FILES**
- 122724 None.
- 122725 **EXTENDED DESCRIPTION**
- 122726 None.
- 122727 **EXIT STATUS**
- 122728 The following exit values shall be returned:
- 122729 0 Successful completion.
- 122730 >0 An error occurred.
- 122731 **CONSEQUENCES OF ERRORS**
- 122732 Default.

122733 **APPLICATION USAGE**

122734 The `-m` option is not a switch, but an option at the same level as `-c`. Thus, to produce the full  
 122735 default output with character counts instead of bytes, the command required is:

122736 `wc -mlw`

122737 **EXAMPLES**

122738 None.

122739 **RATIONALE**

122740 The output file format pseudo-*printf()* string differs from the System V version of *wc*:

122741 `"%7d%7d%7d %s\n"`

122742 which produces possibly ambiguous and unparseable results for very large files, as it assumes no  
 122743 number shall exceed six digits.

122744 Some historical implementations use only `<space>`, `<tab>`, and `<newline>` as word separators.  
 122745 The equivalent of the ISO C standard *isspace()* function is more appropriate.

122746 The `-c` option stands for “character” count, even though it counts bytes. This stems from the  
 122747 sometimes erroneous historical view that bytes and characters are the same size. Due to  
 122748 international requirements, the `-m` option (reminiscent of “multi-byte”) was added to obtain  
 122749 actual character counts.

122750 Early proposals only specified the results when input files were text files. The current  
 122751 specification more closely matches historical practice. (Bytes, words, and `<newline>` characters  
 122752 are counted separately and the results are written when an end-of-file is detected.)

122753 Historical implementations of the *wc* utility only accepted one argument to specify the options  
 122754 `-c`, `-l`, and `-w`. Some of them also had multiple occurrences of an option cause the  
 122755 corresponding count to be written multiple times and had the order of specification of the  
 122756 options affect the order of the fields on output, but did not document either of these. Because  
 122757 common usage either specifies no options or only one option, and because none of this was  
 122758 documented, the changes required by this volume of POSIX.1-2024 should not break many  
 122759 historical applications (and do not break any historical conforming applications).

122760 **FUTURE DIRECTIONS**

122761 If this utility is directed to display a pathname that contains any bytes that have the encoded  
 122762 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
 122763 format being used, implementations are encouraged to treat this as an error. A future version of  
 122764 this standard may require implementations to treat this as an error.

122765 **SEE ALSO**

122766 [\*cksum\*](#)

122767 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

122768 **CHANGE HISTORY**

122769 First released in Issue 2.

122770 **Issue 7**

122771 Austin Group Interpretation 1003.1-2001 #092 is applied.

122772 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

122773 **Issue 8**

122774 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
122775 directed to display a pathname that contains any bytes that have the encoded value of a  
122776 <newline> character when <newline> is a terminator or separator in the output format being  
122777 used.

122778 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

122779 **NAME**122780 what — identify SCCS files (**DEVELOPMENT**)122781 **SYNOPSIS**122782 XSI `what [-s] file...`122783 **DESCRIPTION**

122784 The *what* utility shall search the given files for all occurrences of the pattern that *get* (see [get](#))  
 122785 substitutes for the %Z% keyword ("@ (#) "). The *what* utility shall write to standard output the  
 122786 identification string that follows up to, but not including, the first occurrence of one of the  
 122787 following: <double-quote> ('"'), <greater-than-sign> ('>'), <newline>, <backslash> ('\\'),  
 122788 <NUL> ('\0'), or an end-of-file condition on the input file. If not at end-of-file, the *what* utility  
 122789 shall then look for the next occurrence of "@ (#) " after one of those characters.

122790 **OPTIONS**122791 The *what* utility shall conform to XBD [Section 12.2](#) (on page 215).

122792 The following option shall be supported:

122793 **-s** Write at most one identification string for each file. After locating and writing to  
 122794 standard output the identification string following the first pattern (if any) in a file,  
 122795 no further data shall be read from that file and the search shall recommence from  
 122796 the beginning of the next file, if any.

122797 **OPERANDS**

122798 The following operands shall be supported:

122799 *file* A pathname of a file to search.122800 **STDIN**

122801 Not used.

122802 **INPUT FILES**

122803 The input files shall be of any file type.

122804 **ENVIRONMENT VARIABLES**122805 The following environment variables shall affect the execution of *what*:

122806 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 122807 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 122808 variables used to determine the values of locale categories.)

122809 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 122810 internationalization variables.

122811 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 122812 characters (for example, single-byte as opposed to multi-byte characters in  
 122813 arguments and input files).

122814 **LC\_MESSAGES**

122815 Determine the locale that should be used to affect the format and contents of  
 122816 diagnostic messages written to standard error.

122817 **NLSPATH** Determine the location of messages objects and message catalogs.

122818 **ASYNCHRONOUS EVENTS**

122819 Default.

122820 **STDOUT**

122821 For each *file* operand, the standard output shall consist of:

122822 "%s:\n", <pathname>

122823 followed by zero or more of:

122824 "\t%s\n", <identification string>

122825 one for each identification string located.

122826 **STDERR**

122827 The standard error shall be used only for diagnostic messages.

122828 **OUTPUT FILES**

122829 None.

122830 **EXTENDED DESCRIPTION**

122831 None.

122832 **EXIT STATUS**

122833 The following exit values shall be returned:

122834 0 One or more matches were found and the output specified in STDOUT was successfully  
122835 written to standard output.

122836 1 No matches were found or an error occurred.

122837 **CONSEQUENCES OF ERRORS**

122838 Default.

122839 **APPLICATION USAGE**

122840 The *what* utility is intended to be used in conjunction with the SCCS command *get*, which  
122841 automatically inserts identifying information, but it can also be used where the information is  
122842 inserted by any other means.

122843 When the string "@ (#) " is included in a library routine in a shared library, it might not be found  
122844 in an **a.out** file using that library routine.

122845 **EXAMPLES**

122846 If the C-language program in file **f.c** contains:

122847 char ident[] = "@ (#) identification information";

122848 and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

122849 what f.c f.o a.out

122850 writes:

122851 f.c:  
122852 identification information

122853 ...

122854 f.o:  
122855 identification information

122856 ...

122857 a.out:  
122858 identification information

122859 ...

122860 **RATIONALE**

122861 This standard requires that when the `-s` option is used, *what* does not continue reading from the  
122862 current file after writing the first identification string. This might seem an unimportant detail,  
122863 but applications would experience different behavior if a *file* operand named a FIFO special file  
122864 and *what* waited for an end-of-file condition rather than closing the file straight away.

122865 **FUTURE DIRECTIONS**

122866 If this utility is directed to display a pathname that contains any bytes that have the encoded  
122867 value of a `<newline>` character when `<newline>` is a terminator or separator in the output  
122868 format being used, implementations are encouraged to treat this as an error. A future version of  
122869 this standard may require implementations to treat this as an error.

122870 **SEE ALSO**

122871 *get*  
122872 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

122873 **CHANGE HISTORY**

122874 First released in Issue 2.

122875 **Issue 8**

122876 Austin Group Defect 251 is applied, encouraging implementations to report an error if a utility is  
122877 directed to display a pathname that contains any bytes that have the encoded value of a  
122878 `<newline>` character when `<newline>` is a terminator or separator in the output format being  
122879 used.

122880 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

122881 Austin Group Defect 1512 is applied, changing the EXIT STATUS section.

122882 Austin Group Defect 1538 is applied, clarifying the `-s` option.

122883 Austin Group Defect 1563 is applied, clarifying the output format when the *what* utility finds  
122884 multiple identification strings in one file.



122885 **NAME**122886        **who** — display who is on the system122887 **SYNOPSIS**122888 XSI        **who** [-mTu] [-abdHlprt] [*file*]122889 XSI        **who** [-mu] -s [-bHlprt] [*file*]122890        **who** -q [*file*]122891        **who** am i122892        **who** am I122893 **DESCRIPTION**122894        The *who* utility shall list various pieces of information about accessible users. The domain of  
122895 accessibility is implementation-defined.122896 XSI        Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed  
122897 time since activity occurred on the line, and the process ID of the command interpreter for each  
122898 current system user.122899 **OPTIONS**122900        The *who* utility shall conform to XBD Section 12.2 (on page 215).122901        The following options shall be supported. The metavariables, such as *<line>*, refer to fields  
122902 described in the STDOUT section.122903 XSI        **-a**        Process the implementation-defined database or named file with the **-b**, **-d**, **-l**, **-p**,  
122904 **-r**, **-t**, **-T** and **-u** options turned on.122905 XSI        **-b**        Write the time and date of the last system reboot. The system reboot time is the  
122906 time at which the implementation is able to commence running processes.122907 XSI        **-d**        Write a list of all processes that have expired and not been respawned by the *init*  
122908 system process. The *<exit>* field shall appear for dead processes and contain the  
122909 termination and exit values of the dead process. This can be useful in determining  
122910 why a process terminated.122911 XSI        **-H**        Write column headings above the regular output.122912 XSI        **-l**        (The letter ell.) List only those lines on which the system is waiting for someone to  
122913 login. The *<name>* field shall be **LOGIN** in such cases. Other fields shall be the  
122914 same as for user entries except that the *<state>* field does not exist.122915        **-m**        Output only information about the current terminal.122916 XSI        **-p**        List any other process that is currently active and has been previously spawned by  
122917 *init*.122918 XSI        **-q**        (Quick.) List only the names and the number of users currently logged on. When  
122919 this option is used, all other options shall be ignored.122920 XSI        **-r**        Write the current *run-level* of the *init* process.122921 XSI        **-s**        List only the *<name>*, *<line>*, and *<time>* fields. This is the default case.122922 XSI        **-t**        Indicate the last change to the system clock.122923        **-T**        Show the state of each terminal, as described in the STDOUT section.

122924            **-u**            Write ``idle time'' for each displayed user in addition to any other information. The  
 122925                            idle time is the time since any activity occurred on the user's terminal. The method  
 122926 XSI                           of determining this is unspecified. This option shall list only those users who are  
 122927                            currently logged in. The *<name>* is the user's login name. The *<line>* is the name  
 122928                            of the line as found in the directory */dev*. The *<time>* is the time that the user  
 122929                            logged in. The *<activity>* is the number of hours and minutes since activity last  
 122930                            occurred on that particular line. A dot indicates that the terminal has seen activity  
 122931                            in the last minute and is therefore ``current''. If more than twenty-four hours have  
 122932                            elapsed or the line has not been used since boot time, the entry shall be marked  
 122933                            *<old>*. This field is useful when trying to determine whether a person is working at  
 122934                            the terminal or not. The *<pid>* is the process ID of the user's login process.

#### 122935 OPERANDS

122936 XSI            The following operands shall be supported:

122937            **am i, am I**    In the POSIX locale, limit the output to describing the invoking user, equivalent to  
 122938                            the **-m** option. The **am** and **i** or **I** need to be separate arguments.

122939            *file*           Specify a pathname of a file to substitute for the implementation-defined database  
 122940                            of logged-on users that *who* uses by default.

#### 122941 STDIN

122942            Not used.

#### 122943 INPUT FILES

122944            None.

#### 122945 ENVIRONMENT VARIABLES

122946            The following environment variables shall affect the execution of *who*:

122947            **LANG**            Provide a default value for the internationalization variables that are unset or null.  
 122948                            (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 122949                            variables used to determine the values of locale categories.)

122950            **LC\_ALL**           If set to a non-empty string value, override the values of all the other  
 122951                            internationalization variables.

122952            **LC\_CTYPE**       Determine the locale for the interpretation of sequences of bytes of text data as  
 122953                            characters (for example, single-byte as opposed to multi-byte characters in  
 122954                            arguments).

122955            **LC\_MESSAGES**

122956                            Determine the locale that should be used to affect the format and contents of  
 122957                            diagnostic messages written to standard error.

122958            **LC\_TIME**        Determine the locale used for the format and contents of the date and time strings.

122959 XSI            **NLSPATH**        Determine the location of messages objects and message catalogs.

122960            **TZ**             Determine the timezone used when writing date and time information. If **TZ** is  
 122961                            unset or null, an unspecified default timezone shall be used.

#### 122962 ASYNCHRONOUS EVENTS

122963            Default.

#### 122964 STDOUT

122965            The *who* utility shall write its default format to the standard output in an implementation-  
 122966                            defined format, subject only to the requirement of containing the information described above.

122967 XSI OF XSI-conformant systems shall write the default information to the standard output in the  
 122968 following general format:

122969 `<name> [<state>] <line> <time> [<activity>] [<pid>] [<comment>] [<exit>]`

122970 For the `-b` option, `<line>` shall be "system boot". The `<name>` is unspecified.

122971 The following format shall be used for the `-T` option:

122972 `"%s %c %s %s\n" <name>, <terminal state>, <terminal name>,  
 122973 <time of login>`

122974 where `<terminal state>` is one of the following characters:

122975 + The terminal allows write access to other users.

122976 - The terminal denies write access to other users.

122977 ? The terminal write-access state cannot be determined.

122978 `<space>` This entry is not associated with a terminal.

122979 In the POSIX locale, the `<time of login>` shall be equivalent in format to the output of:

122980 `date +"%b %e %H:%M"`

122981 If the `-u` option is used with `-T`, the idle time shall be added to the end of the previous format in  
 122982 an unspecified format.

#### 122983 **STDERR**

122984 The standard error shall be used only for diagnostic messages.

#### 122985 **OUTPUT FILES**

122986 None.

#### 122987 **EXTENDED DESCRIPTION**

122988 None.

#### 122989 **EXIT STATUS**

122990 The following exit values shall be returned:

122991 0 Successful completion.

122992 >0 An error occurred.

#### 122993 **CONSEQUENCES OF ERRORS**

122994 Default.

#### 122995 **APPLICATION USAGE**

122996 The name *init* used for the system process is the most commonly used on historical systems, but  
 122997 it may vary.

122998 The "domain of accessibility" referred to is a broad concept that permits interpretation either on  
 122999 a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

#### 123000 **EXAMPLES**

123001 None.

#### 123002 **RATIONALE**

123003 Due to differences between historical implementations, the base options provided were a  
 123004 compromise to allow users to work with those functions. The standard developers also  
 123005 considered removing all the options, but felt that these options offered users valuable  
 123006 functionality. Additional options to match historical systems are available on XSI-conformant

- 123007 systems.
- 123008 It is recognized that the *who* command may be of limited usefulness, especially in a multi-level  
123009 secure environment. The standard developers considered, however, that having some standard  
123010 method of determining the “accessibility” of other users would aid user portability.
- 123011 No format was specified for the default *who* output for systems not supporting the XSI option. In  
123012 such a user-oriented command, designed only for human use, this was not considered to be a  
123013 deficiency.
- 123014 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require  
123015 that they use the same format.
- 123016 It is acceptable for an implementation to produce no output for an invocation of *who mil*.
- 123017 **FUTURE DIRECTIONS**
- 123018 None.
- 123019 **SEE ALSO**
- 123020 *mesg*
- 123021 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)
- 123022 **CHANGE HISTORY**
- 123023 First released in Issue 2.
- 123024 **Issue 6**
- 123025 This utility is marked as part of the User Portability Utilities option.
- 123026 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.
- 123027 **Issue 7**
- 123028 SD5-XCU-ERN-58 is applied, clarifying the **-b** option.
- 123029 The *who* utility is moved from the User Portability Utilities option to the Base. User Portability  
123030 Utilities is now an option for interactive utilities.
- 123031 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 123032 **Issue 8**
- 123033 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

123034 **NAME**

123035 write — write to another user

123036 **SYNOPSIS**123037 write *user\_name* [*terminal*]123038 **DESCRIPTION**123039 The *write* utility shall read lines from the standard input and write them to the terminal of the  
123040 specified user. When first invoked, it shall write the message:123041 **Message from** *sender-login-id* (*sending-terminal*) [*date*]...123042 to *user\_name*. When it has successfully completed the connection, the sender's terminal shall be  
123043 alerted twice to indicate that what the sender is typing is being written to the recipient's  
123044 terminal.

123045 If the recipient wants to reply, this can be accomplished by typing:

123046 write *sender-login-id* [*sending-terminal*]123047 upon receipt of the initial message. Whenever a line of input as delimited by an NL, EOF, or  
123048 EOL special character (see XBD [Chapter 11](#), on page 199) is accumulated while in canonical  
123049 input mode, the accumulated data shall be written on the other user's terminal. Characters shall  
123050 be processed as follows:

- 123051 • Typing <alert> shall write the <alert> character to the recipient's terminal.
- 123052 • Typing the erase and kill characters shall affect the sender's terminal in the manner  
123053 described by the **termios** interface in XBD [Chapter 11](#) (on page 199).
- 123054 • Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate  
123055 message ("EOT\n" in the POSIX locale) to the recipient's terminal and exit.
- 123056 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those  
123057 characters to be sent to the recipient's terminal.
- 123058 • When and only when the *stty* **ixten** local mode is enabled, the existence and processing of  
123059 additional special control characters and multi-byte or single-byte functions is  
123060 implementation-defined.
- 123061 • Typing other non-printable characters shall cause implementation-defined sequences of  
123062 printable characters to be written to the recipient's terminal.

123063 To write to a user who is logged in more than once, the *terminal* argument can be used to  
123064 indicate which terminal to write to; otherwise, the recipient's terminal is selected in an  
123065 implementation-defined manner and an informational message is written to the sender's  
123066 standard output, indicating which terminal was chosen.

123067 Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg*  
123068 utility. However, a user's privilege may further constrain the domain of accessibility of other  
123069 users' terminals. The *write* utility shall fail when the user lacks appropriate privileges to perform  
123070 the requested action.

123071 **OPTIONS**

123072 None.

123073 **OPERANDS**

- 123074 The following operands shall be supported:
- 123075 *user\_name* Login name of the person to whom the message shall be written. The application  
123076 shall ensure that this operand is of the form returned by the *who* utility.
- 123077 *terminal* Terminal identification in the same format provided by the *who* utility.
- 123078 **STDIN**
- 123079 Lines to be copied to the recipient's terminal are read from standard input.
- 123080 **INPUT FILES**
- 123081 None.
- 123082 **ENVIRONMENT VARIABLES**
- 123083 The following environment variables shall affect the execution of *write*:
- 123084 *LANG* Provide a default value for the internationalization variables that are unset or null.  
123085 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
123086 variables used to determine the values of locale categories.)
- 123087 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
123088 internationalization variables.
- 123089 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
123090 characters (for example, single-byte as opposed to multi-byte characters in  
123091 arguments and input files). If the recipient's locale does not use an *LC\_CTYPE*  
123092 equivalent to the sender's, the results are undefined.
- 123093 *LC\_MESSAGES*
- 123094 Determine the locale that should be used to affect the format and contents of  
123095 diagnostic messages written to standard error and informative messages written to  
123096 standard output.
- 123097 XSI *NLSPATH* Determine the location of messages objects and message catalogs.
- 123098 **ASYNCHRONOUS EVENTS**
- 123099 If an interrupt signal is received, *write* shall write an appropriate message on the recipient's  
123100 terminal and exit with a status of zero. It shall take the standard action for all other signals.
- 123101 **STDOUT**
- 123102 An informational message shall be written to standard output if a recipient is logged in more  
123103 than once.
- 123104 **STDERR**
- 123105 The standard error shall be used only for diagnostic messages.
- 123106 **OUTPUT FILES**
- 123107 The recipient's terminal is used for output.
- 123108 **EXTENDED DESCRIPTION**
- 123109 None.
- 123110 **EXIT STATUS**
- 123111 The following exit values shall be returned:
- 123112 0 Successful completion.
- 123113 >0 The addressed user is not logged on or the addressed user denies permission.

123114 **CONSEQUENCES OF ERRORS**

123115 Default.

123116 **APPLICATION USAGE**123117 The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.123118 **EXAMPLES**

123119 None.

123120 **RATIONALE**

123121 The *write* utility was included in this volume of POSIX.1-2024 since it can be implemented on all  
123122 terminal types. The standard developers considered the *talk* utility, which cannot be  
123123 implemented on certain terminals, to be a “better” communications interface. Both of these  
123124 programs are in widespread use on historical implementations. Therefore, the standard  
123125 developers decided that both utilities should be specified.

123126 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*  
123127 require that they all use or accept the same format.

123128 **FUTURE DIRECTIONS**

123129 None.

123130 **SEE ALSO**123131 *mesg*, *talk*, *who*123132 XBD [Chapter 8](#) (on page 167), [Chapter 11](#) (on page 199)123133 **CHANGE HISTORY**

123134 First released in Issue 2.

123135 **Issue 5**

123136 The FUTURE DIRECTIONS section is added.

123137 **Issue 6**

123138 This utility is marked as part of the User Portability Utilities option.

123139 The normative text is reworded to avoid use of the term “must” for application requirements.

123140 **Issue 7**

123141 The *write* utility is moved from the User Portability Utilities option to the Base. User Portability  
123142 Utilities is now an option for interactive utilities.

123143 **Issue 8**123144 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

123145 **NAME**

123146 xargs — construct argument lists and invoke utility

123147 **SYNOPSIS**

```
123148 XSI xargs [-prtx] [-E eofstr|-0] [-I replstr|-L number|-n number]
123149 [-s size] [utility [argument...]]
```

123150 **DESCRIPTION**

123151 The *xargs* utility shall construct a command line consisting of the *utility* and *argument* operands  
 123152 specified followed by as many arguments read in sequence from standard input as fit in length  
 123153 and number constraints specified by the options. The *xargs* utility shall then invoke the  
 123154 constructed command line and wait for its completion. This sequence shall be repeated until one  
 123155 of the following occurs:

- 123156 • An end-of-file condition is detected on standard input.
- 123157 • An argument consisting of just the logical end-of-file string (see the `-E eofstr` option) is  
 123158 found on standard input after double-quote processing, <apostrophe> processing, and  
 123159 <backslash>-escape processing (see next paragraph). All arguments up to but not  
 123160 including the argument consisting of just the logical end-of-file string shall be used as  
 123161 arguments in constructed command lines.
- 123162 • An invocation of a constructed command line returns an exit status of 255.

123163 If the `-0` option is not specified, the application shall ensure that arguments in the standard  
 123164 input are delimited by unquoted <blank> characters, unescaped <blank> characters, or  
 123165 <newline> characters, and quoting characters shall be interpreted as follows:

- 123166 • A string of zero or more non-double-quote ( ' ' ) non-<newline> characters can be quoted  
 123167 by enclosing them in double-quotes.
- 123168 • A string of zero or more non-<apostrophe> ( ' \ ' ) non-<newline> characters can be  
 123169 quoted by enclosing them in <apostrophe> characters.
- 123170 • Any unquoted character can be escaped by preceding it with a <backslash>.

123171 Multiple adjacent delimiter characters shall be treated as a single delimiter. If the standard input  
 123172 is not empty and does not end with a <newline>, the behavior is undefined (because the  
 123173 requirement in STDIN that the input is a text file is not met in that case).

123174 If the `-0` option is specified, the application shall ensure that arguments in the standard input are  
 123175 delimited by null bytes. If multiple adjacent null bytes occur in the input, each null byte shall be  
 123176 treated as a delimiter. If the standard input is not empty and does not end with a null byte, *xargs*  
 123177 should ignore the trailing non-null bytes (as this can signal incomplete data) but may use them  
 123178 as the last argument passed to utility.

123179 The utility named by *utility* shall be executed zero or more times until the end-of-file is reached  
 123180 or the logical end-of file string is found. If no arguments are supplied on standard input, the  
 123181 utility named by *utility* shall be executed zero times if the `-r` option is specified and shall be  
 123182 executed exactly once if the `-r` option is not specified. The results are unspecified if the utility  
 123183 named by *utility* attempts to read from its standard input.

123184 The generated command line length shall be the sum of the size in bytes of the utility name and  
 123185 each argument treated as strings, including a null byte terminator for each of these strings. The  
 123186 *xargs* utility shall limit the command line length such that when the command line is invoked,  
 123187 the combined argument and environment lists (see the *exec* family of functions in the System  
 123188 Interfaces volume of POSIX.1-2024) shall not exceed {ARG\_MAX}-2 048 bytes. Within this  
 123189 constraint, if neither the `-n` nor the `-s` option is specified, the default command line length shall  
 123190 be at least {LINE\_MAX}.



123191 **OPTIONS**

123192 The *xargs* utility shall conform to XBD Section 12.2 (on page 215).

123193 The following options shall be supported:

123194 **-E *eofstr*** Use *eofstr* as the logical end-of-file string. If neither **-E** nor **-0** is specified, it is  
 123195 unspecified whether the logical end-of-file string is the `<underscore>` character  
 123196 ('\_') or the end-of-file string capability is disabled. When *eofstr* is the null string,  
 123197 the logical end-of-file string capability shall be disabled and `<underscore>`  
 123198 characters shall be taken literally.

123199 XSI **-I *replstr*** Insert mode: invoke *utility* for each argument from standard input. If **-0** is not  
 123200 specified, arguments in the standard input shall be delimited only by unescaped  
 123201 `<newline>` characters, not by `<blank>` characters, and any unquoted unescaped  
 123202 `<blank>` characters at the beginning of each line shall be ignored. The resulting  
 123203 argument shall be inserted in *arguments* in place of each occurrence of *replstr*. At  
 123204 least five arguments in *arguments* can each contain one or more instances of *replstr*.  
 123205 Each of these constructed arguments cannot grow larger than an implementation-  
 123206 defined limit greater than or equal to 255 bytes. Option **-x** shall be forced on.

123207 XSI **-L *number*** Invoke *utility* for each set of *number* arguments from standard input. The last  
 123208 invocation of *utility* shall be with fewer arguments if fewer than *number* remain. If  
 123209 the **-0** option is not specified, each line in the standard input shall be treated as  
 123210 containing one argument except that empty lines shall be ignored and a line  
 123211 ending with a trailing unescaped `<blank>` shall signal continuation to the next  
 123212 non-empty line, inclusive; such continuation shall result in removal of all trailing  
 123213 unescaped `<blank>` characters and all `<newline>` characters that immediately  
 123214 follow them from the argument.

123215 **-n *number*** Invoke *utility* using as many standard input arguments as possible, up to *number* (a  
 123216 positive decimal integer) arguments maximum. Fewer arguments shall be used if:

123217 • The command line length accumulated exceeds the size specified by the **-s**  
 123218 option (or `{LINE_MAX}` if there is no **-s** option).

123219 • The last iteration has fewer than *number*, but not zero, operands remaining.

123220 **-p** Prompt mode: the user is asked whether to execute *utility* at each invocation. Trace  
 123221 mode (**-t**) is turned on to write the command instance to be executed, followed by  
 123222 a prompt to standard error. An affirmative response read from `/dev/tty` shall  
 123223 execute the command; otherwise, that particular invocation of *utility* shall be  
 123224 skipped.

123225 **-r** Do not execute the utility named by *utility* if no arguments are supplied on  
 123226 standard input.

123227 **-s *size*** Invoke *utility* using as many standard input arguments as possible yielding a  
 123228 command line length less than *size* (a positive decimal integer) bytes. Fewer  
 123229 arguments shall be used if:

123230 • The total number of arguments exceeds that specified by the **-n** option.

123231 XSI • The total number of arguments exceeds that specified by the **-L** option.

123232 • End-of-file is encountered on standard input before *size* bytes are  
 123233 accumulated.

123234 Values of *size* up to at least `{LINE_MAX}` bytes shall be supported, provided that  
 123235 the constraints specified in the DESCRIPTION are met. It shall not be considered

123236 an error if a value larger than that supported by the implementation or exceeding  
 123237 the constraints specified in the DESCRIPTION is given; *xargs* shall use the largest  
 123238 value it supports within the constraints.

123239 **-t** Enable trace mode. Each generated command line shall be written to standard  
 123240 error just prior to invocation.

123241 **-x** Terminate if a constructed command line will not fit in the implied or specified size  
 123242 (see the **-s** option above).

123243 **-0** Use a null byte as the input argument delimiter and do not treat any other input  
 123244 bytes as special.

123245 If the mutually exclusive **-0** and **-E eofstr** options are both specified, the behavior is unspecified,  
 123246 except that if *eofstr* is the null string the behavior shall be the same as if **-0** was specified without  
 123247 **-E eofstr**.

#### 123248 OPERANDS

123249 The following operands shall be supported:

123250 *utility* The name of the utility to be invoked, found by search path using the *PATH*  
 123251 environment variable, described in XBD [Chapter 8](#) (on page 167). If *utility* is  
 123252 omitted, the default shall be the *echo* utility. If the *utility* operand names any of the  
 123253 special built-in utilities in [Section 2.15](#) (on page 2526), the results are undefined.

123254 *argument* An initial option or operand for the invocation of *utility*.

#### 123255 STDIN

123256 If the **-0** option is not specified, the standard input shall be a text file and the results are  
 123257 unspecified if an end-of-file condition is detected immediately following an escaped `<newline>`.

123258 If the **-0** option is specified, the standard input need not be a text file, and *xargs* shall process the  
 123259 input as bytes, not characters.

#### 123260 INPUT FILES

123261 The file `/dev/tty` shall be used to read responses required by the **-p** option.

#### 123262 ENVIRONMENT VARIABLES

123263 The following environment variables shall affect the execution of *xargs*:

123264 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 123265 (See XBD [Section 8.2](#) (on page 169) for the precedence of internationalization  
 123266 variables used to determine the values of locale categories.)

123267 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 123268 internationalization variables.

123269 *LC\_COLLATE*

123270 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 123271 character collating elements used in the extended regular expression defined for  
 123272 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.

123273 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 123274 characters (for example, single-byte as opposed to multi-byte characters in  
 123275 arguments and input files) and the behavior of character classes used in the  
 123276 extended regular expression defined for the **yesexpr** locale keyword in the  
 123277 *LC\_MESSAGES* category.

- 123278 **LC\_MESSAGES**
- 123279 Determine the locale used to process affirmative responses, and the locale used to
- 123280 affect the format and contents of diagnostic messages and prompts written to
- 123281 standard error.
- 123282 XSI **NLSPATH** Determine the location of messages objects and message catalogs.
- 123283 **PATH** Determine the location of *utility*, as described in XBD [Chapter 8](#) (on page 167).
- 123284 **ASYNCHRONOUS EVENTS**
- 123285 Default.
- 123286 **STDOUT**
- 123287 Not used.
- 123288 **STDERR**
- 123289 The standard error shall be used for diagnostic messages and the **-t** and **-p** options. If the **-t**
- 123290 option is specified, the *utility* and its constructed argument list shall be written to standard error,
- 123291 as it will be invoked, prior to invocation. If **-p** is specified, a prompt of the following format
- 123292 shall be written (in the POSIX locale):
- 123293 " ? . . . "
- 123294 at the end of the line of the output from **-t**.
- 123295 **OUTPUT FILES**
- 123296 None.
- 123297 **EXTENDED DESCRIPTION**
- 123298 None.
- 123299 **EXIT STATUS**
- 123300 The following exit values shall be returned:
- 123301 0 Successful completion.
- 123302 1-125 A command line meeting the specified requirements could not be assembled, one or
- 123303 more of the invocations of *utility* returned a non-zero exit status, or some other error
- 123304 occurred.
- 123305 126 The utility specified by *utility* was found but could not be invoked.
- 123306 127 The utility specified by *utility* could not be found.
- 123307 **CONSEQUENCES OF ERRORS**
- 123308 If a command line meeting the specified requirements cannot be assembled, the utility cannot be
- 123309 invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits
- 123310 with exit status 255, the *xargs* utility shall write a diagnostic message and exit without
- 123311 processing any remaining input.
- 123312 **APPLICATION USAGE**
- 123313 The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no
- 123314 further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit*
- 123315 with an appropriate value to avoid accidentally returning with 255.
- 123316 Note that since input is parsed as lines (if **-0** is not specified), with <blank> characters
- 123317 separating arguments and <backslash>, <apostrophe>, and double-quote characters used for
- 123318 quoting, if *xargs* is used to bundle the output of commands like *find dir -print* or *ls* into
- 123319 commands to be executed, unexpected results are likely if any filenames contain <blank>,
- 123320 <newline>, or quoting characters. This can be solved by using the **-print0** primary of *find*

123321 together with the *xargs* `-0` option, or by using *find* to call a script that converts each file found  
 123322 into a quoted string that is then piped to *xargs*, but in most cases it is preferable just to have *find*  
 123323 do the argument aggregation itself by using `-exec` with a '+' terminator instead of ';' . Note  
 123324 that the quoting rules used by *xargs* are not the same as in the shell. They were not made  
 123325 consistent here because existing applications depend on the current rules. An easy (but  
 123326 inefficient) method that can be used to transform input consisting of one argument per line into  
 123327 a quoted form that *xargs* interprets correctly is to precede each non-`<newline>` character with a  
 123328 `<backslash>`. More efficient alternatives are shown in Example 2 and Example 5 below.

123329 On implementations with a large value for `{ARG_MAX}`, *xargs* may produce command lines  
 123330 longer than `{LINE_MAX}`. For invocation of utilities, this is not a problem. If *xargs* is being used  
 123331 to create a text file, users should explicitly set the maximum command line length with the `-s`  
 123332 option.

123333 The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit  
 123334 code 127 if a utility to be invoked cannot be found, so that applications can distinguish "failure  
 123335 to find a utility" from "invoked utility exited with an error indication". The value 127 was  
 123336 chosen because it is not commonly used for other meanings; most utilities use small values for  
 123337 "normal error conditions" and the values above 128 can be confused with termination due to  
 123338 receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could  
 123339 be found, but not invoked. Some scripts produce meaningful error messages differentiating the  
 123340 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice  
 123341 that uses 127 when all attempts to *exec* the utility fail with `[ENOENT]`, and uses 126 when any  
 123342 attempt to *exec* the utility fails for any other reason.

#### 123343 EXAMPLES

123344 1. The following command combines the output of the parenthesized commands (minus the  
 123345 `<apostrophe>` characters) onto one line, which is then appended to the file `log`. It assumes  
 123346 that the expansion of `"$0 $*"` does not include any `<apostrophe>` or `<newline>`  
 123347 characters.

```
123348 (logname; date; printf "'%s'\n" "$0 $*") | xargs -E "" >>log
```

123349 2. The following command invokes *diff* with successive pairs of arguments originally typed  
 123350 as command line arguments.

```
123351 printf "%s\n" "$@" | xargs -0 -n 2 -x diff --
```

123352 3. In the following command, the user is asked which regular files below the current  
 123353 directory are to be archived.

```
123354 find . -type f -print0 | xargs -0 -p -L 1 ar -r arch
```

123355 4. The following command invokes *command1* one or more times with multiple arguments,  
 123356 stopping if an invocation of *command1* has a non-zero exit status.

```
123357 xargs -E "" sh -c 'command1 "$@" || exit 255' sh < xargs_input
```

#### 123358 RATIONALE

123359 The *xargs* utility was usually found only in System V-based systems; BSD systems included an  
 123360 *apply* utility that provided functionality similar to *xargs* `-n number`. The SVID lists *xargs* as a  
 123361 software development extension. This volume of POSIX.1-2024 does not share the view that it is  
 123362 used only for development, and therefore it is not optional.

123363 The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the  
 123364 number of processes launched by a simplistic use of the *find* `-exec` combination. The *xargs* utility  
 123365 is also used to enforce an upper limit on memory required to launch a process. With this basis in

123366 mind, this volume of POSIX.1-2024 selected only the minimal features required.

123367 Although the 255 exit status is mostly an accident of historical implementations, it allows a  
 123368 utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the  
 123369 current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125  
 123370 range when *xargs* exits. There is no statement of how the various non-zero utility exit status  
 123371 codes are accumulated by *xargs*. The value could be the addition of all codes, their highest  
 123372 value, the last one received, or a single value such as 1. Since no algorithm is arguably better  
 123373 than the others, and since many of the standard utilities say little more (portably) than  
 123374 “pass/fail”, no new algorithm was invented.

123375 Several other *xargs* options were removed because simple alternatives already exist within this  
 123376 volume of POSIX.1-2024. For example, the `-i replstr` option can be just as efficiently performed  
 123377 using a shell `for` loop. Since *xargs* calls an *exec* function with each input line, the `-i` option does  
 123378 not usually exploit the grouping capabilities of *xargs*.

123379 The requirement that *xargs* never produces command lines such that invocation of *utility* is  
 123380 within 2048 bytes of hitting the POSIX *exec* {ARG\_MAX} limitations is intended to guarantee  
 123381 that the invoked utility has room to modify its environment variables and command line  
 123382 arguments and still be able to invoke another utility. Note that the minimum {ARG\_MAX}  
 123383 allowed by the System Interfaces volume of POSIX.1-2024 is 4096 bytes and the minimum value  
 123384 allowed by this volume of POSIX.1-2024 is 2048 bytes; therefore, the 2048 bytes difference seems  
 123385 reasonable. Note, however, that *xargs* may never be able to invoke a utility if the environment  
 123386 passed in to *xargs* comes close to using {ARG\_MAX} bytes.

123387 The version of *xargs* required by this volume of POSIX.1-2024 is required to wait for the  
 123388 completion of the invoked command before invoking another command. This was done because  
 123389 historical scripts using *xargs* assumed sequential execution. Implementations wanting to provide  
 123390 parallel operation of the invoked utilities are encouraged to add an option enabling parallel  
 123391 invocation, but should still wait for termination of all of the children before *xargs* terminates  
 123392 normally.

123393 The `-e` option was omitted from the ISO POSIX-2:1993 standard in the belief that the *eofstr*  
 123394 option-argument was recognized only when it was on a line by itself and before quote and  
 123395 escape processing were performed, and that the logical end-of-file processing was only enabled  
 123396 if a `-e` option was specified. In that case, a simple *sed* script could be used to duplicate the `-e`  
 123397 functionality. Further investigation revealed that:

- 123398 • The logical end-of-file string was checked for after quote and escape processing, making a  
 123399 *sed* script that provided equivalent functionality much more difficult to write.
- 123400 • The default was to perform logical end-of-file processing with an `<underscore>` as the  
 123401 logical end-of-file string.

123402 To correct this misunderstanding, the `-E eofstr` option was adopted from the X/Open Portability  
 123403 Guide. Users should note that the description of the `-E` option matches historical documentation  
 123404 of the `-e` option (which was not adopted because it did not support the Utility Syntax  
 123405 Guidelines), by saying that if *eofstr* is the null string, logical end-of-file processing is disabled.  
 123406 Historical implementations of *xargs* actually did not disable logical end-of-file processing; they  
 123407 treated a null argument found in the input as a logical end-of-file string. (A null *string* argument  
 123408 could be generated using single or double-quotes ( ' ' or " " ). Since this behavior was not  
 123409 documented historically, it is considered to be a bug.

123410 The `-I`, `-L`, and `-n` options are mutually-exclusive. Some implementations use the last one  
 123411 specified if more than one is given on a command line; other implementations treat  
 123412 combinations of the options in different ways.

123413 **FUTURE DIRECTIONS**

123414 A future version of this standard may require that, when the `-0` option is specified, if the  
123415 standard input is not empty and does not end with a null byte, *xargs* ignores the trailing non-  
123416 null bytes.

123417 **SEE ALSO**

123418 [Chapter 2](#) (on page 2472), *diff*, *echo*, *find*

123419 [XBD Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

123420 [XSH \*exec\*](#)

123421 **CHANGE HISTORY**

123422 First released in Issue 2.

123423 **Issue 5**

123424 A second FUTURE DIRECTION is added.

123425 **Issue 6**

123426 The obsolescent `-e`, `-i`, and `-l` options are removed.

123427 The following new requirements on POSIX implementations derive from alignment with the  
123428 Single UNIX Specification:

- 123429 • The `-p` option is added.
- 123430 • In the INPUT FILES section, the file `/dev/tty` is used to read responses required by the `-p`  
123431 option.
- 123432 • The STDERR section is updated to describe the `-p` option.

123433 The description of the `-E` option is aligned with the ISO POSIX-2: 1993 standard.

123434 The normative text is reworded to avoid use of the term “must” for application requirements.

123435 **Issue 7**

123436 Austin Group Interpretation 1003.1-2001 #123 is applied, changing the description of the *xargs* `-I`  
123437 option.

123438 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the  
123439 `LC_MESSAGES` environment variable.

123440 SD5-XCU-ERN-68 is applied.

123441 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

123442 SD5-XCU-ERN-128 is applied, clarifying the DESCRIPTION of the logical end-of-file string.

123443 SD5-XCU-ERN-132 is applied, updating the EXAMPLES section.

123444 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0149 [342] is applied.

123445 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0203 [499] is applied.

123446 **Issue 8**

123447 Austin Group Defect 243 is applied, adding the `-r` and `-0` options.

123448 Austin Group Defect 248 is applied, changing the EXAMPLES section.

123449 Austin Group Defect 1122 is applied, changing the description of `NLSPATH`.

123450 Austin Group Defect 1586 is applied, adding the *timeout* utility.

123451

Austin Group Defect 1594 is applied, changing the APPLICATION USAGE section.

## 123452 NAME

123453 *xgettext* — extract *gettext* call strings from C-language source files (DEVELOPMENT)

## 123454 SYNOPSIS

```
123455 CD xgettext [-j] [-n] [-d default-domain] [-K keyword-spec]...
123456 [-p pathname] file...
123457 xgettext -a [-n] [-d default-domain] [-p pathname]
123458 [-x exclude-file] file...
```

## 123459 DESCRIPTION

123460 The *xgettext* utility shall automate the creation of portable messages object source files (dot-po  
123461 files). A dot-po file shall contain copies of string literals that are found in C-language source  
123462 code in files specified by *file* operands. The dot-po file can be used as input to the *msgfmt* utility,  
123463 to produce a messages object file that can be used by applications.

123464 The *xgettext* utility shall write *msgid* argument strings that are passed as string literals in  
123465 *gettext()*, *gettext\_l()*, *ngettext()*, and *ngettext\_l()* calls in C-language source code to the default  
123466 output file; this file shall be named **messages.po** unless it is changed by the **-d** option. The  
123467 *xgettext* utility shall also write *msgid* argument strings that are passed as string literals in  
123468 *dcgettext()*, *dcgettext\_l()*, *dcngettext()*, *dcngettext\_l()*, *dgettext()*, *dgettext\_l()*, *dngettext()*, and  
123469 *dngettext\_l()* calls either to the default output file or to the output file *domainname.po* where  
123470 *domainname* is the first parameter to the call; it is implementation-defined which of those output  
123471 files is used. A **msgid** directive shall precede each *msgid* argument string. For the functions that  
123472 have a *msgid\_plural* argument, a **msgid\_plural** directive followed by that argument string shall  
123473 also be written directly after the corresponding **msgid** directive. A **msgstr** directive or  
123474 **msgstr[index]** directives with an empty string shall be written after the corresponding **msgid** or  
123475 **msgid\_plural** directive, respectively. The function names that *xgettext* searches for can be  
123476 changed using the **-K** option.

123477 The first directive in each created dot-po file shall be a **domain** directive giving the associated  
123478 domain name, except that this directive is optional in the default output file.

123479 If the **-p pathname** option is specified, *xgettext* shall create the dot-po files in the *pathname*  
123480 directory. Otherwise, the dot-po files shall be created in the current working directory.

123481 The **msgid** values shall be in the same order that the strings are extracted from each *file* and  
123482 subsections with duplicate **msgid** values shall be written to the dot-po files as comment lines.

## 123483 OPTIONS

123484 The *xgettext* utility shall conform to XBD [Section 12.2](#) (on page 215).

123485 The following options shall be supported:

123486 **-a** Extract all strings, not just those found in calls to *gettext* family functions. Only one  
123487 dot-po file shall be created.

123488 **-d default-domain**  
123489 Name the default output file *default-domain.po* instead of **messages.po**.

123490 **-j** Join messages from C-language source files with existing dot-po files. For each  
123491 dot-po file that *xgettext* writes messages to, if the file does not exist, it shall be  
123492 created. New messages shall be appended but any subsections with duplicate  
123493 **msgid** values except the first (including **msgid** values found in an existing dot-po  
123494 file) shall either be commented out or omitted in the resulting dot-po file; if  
123495 omitted, a warning message may be written to standard error. Domain directives  
123496 in the existing dot-po files shall be ignored; the assumption is that all previous



123497 **msgid** values belong to the same domain. The behavior is unspecified if an existing  
123498 dot-po file was not created by *xgettext* or has been modified by another application.

123499 **-K** *keyword-spec*

123500 Specify an additional keyword to be looked for:

123501 • If *keyword-spec* is an empty string, this shall disable the use of default  
123502 keywords for the *gettext* family of functions.

123503 • If *keyword-spec* is a C identifier, *xgettext* shall look for strings in the first  
123504 argument of each call to the function or macro *keyword-spec*.

123505 • If *keyword-spec* is of the form *id:argnum* then *xgettext* shall treat the *argnum*-th  
123506 argument of a call to the function or macro *id* as the *msgid* argument, where  
123507 *argnum* 1 is the first argument.

123508 • If *keyword-spec* is of the form *id:argnum1,argnum2* then *xgettext* shall treat  
123509 strings in the *argnum1*-th argument and in the *argnum2*-th argument of a call  
123510 to the function or macro *id* as the *msgid* and *msgid\_plural* arguments,  
123511 respectively.

123512 For all mentioned forms, the application shall ensure that if *argnum2* is given, it is  
123513 not equal to *argnum1*. All numeric values shall be converted as specified in item 6  
123514 in XBD [Section 12.1](#) (on page 213).

123515 **-n** Add comment lines to the output file indicating pathnames and line numbers in  
123516 the source files where each extracted string is encountered. These lines shall  
123517 appear before each **msgid** directive. Such comments should have the format:

123518 `#: pathname1:linenumber1 [pathname2:linenumber2...]`

123519 **-p** *pathname*

123520 Create output files in the directory specified by *pathname* instead of in the current  
123521 working directory.

123522 **-x** *exclude-file*

123523 Specify a file containing strings that shall not be extracted from the input files. The  
123524 format of *exclude-file* is identical to that of a dot-po file. However, only statements  
123525 containing **msgid** directives in *exclude-file* shall be used. All other statements shall  
123526 be ignored.

## 123527 OPERANDS

123528 The following operand shall be supported:

123529 *file* A pathname of an input file containing C-language source code. If '-' is specified  
123530 for an instance of *file*, the standard input shall be used.

## 123531 STDIN

123532 The standard input shall not be used unless a *file* operand is specified as '- '.

## 123533 INPUT FILES

123534 The input files specified as *file* operands shall be C-language source files. The input file specified  
123535 as the option-argument for the **-x** option shall be a dot-po file in the format specified as input for  
123536 the *msgfmt* utility.

## 123537 ENVIRONMENT VARIABLES

123538 The following environment variables shall affect the execution of *xgettext*:

|        |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 123539 | <i>LANG</i>                   | Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 169) for the precedence of internationalization variables used to determine the values of locale categories.)                                                                                                                                                                                  |
| 123540 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123541 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123542 | XSI <i>LANGUAGE</i>           | Determine the location of messages objects if <i>NLSPATH</i> is not set or the evaluation of <i>NLSPATH</i> did not lead to a suitable messages object being found.                                                                                                                                                                                                                                                 |
| 123543 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123544 | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                                                                                                                            |
| 123545 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123546 | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).                                                                                                                                                                                                                           |
| 123547 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123548 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123549 | <i>LC_MESSAGES</i>            |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123550 |                               | Determine the locale name used to locate messages objects, and the locale that should be used to affect the format and contents of diagnostic messages written to standard error.                                                                                                                                                                                                                                   |
| 123551 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123552 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123553 | XSI <i>NLSPATH</i>            | Determine the location of messages objects and message catalogs.                                                                                                                                                                                                                                                                                                                                                    |
| 123554 | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123555 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 123556 | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123557 |                               | The standard output shall not be used.                                                                                                                                                                                                                                                                                                                                                                              |
| 123558 | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123559 |                               | The standard error shall be used for diagnostic messages and may be used for warning messages.                                                                                                                                                                                                                                                                                                                      |
| 123560 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123561 | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123562 |                               | The output files shall be dot-po files in the format specified as input for the <i>msgfmt</i> utility. It is unspecified whether each output file includes a header ( <b>msgid ""</b> ) before the content derived from the input C-language source files.                                                                                                                                                          |
| 123563 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123564 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123565 | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123566 |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123567 | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123568 |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                        |
| 123569 |                               | 0 Successful completion.                                                                                                                                                                                                                                                                                                                                                                                            |
| 123570 |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                                                                               |
| 123571 | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123572 |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 123573 | <b>APPLICATION USAGE</b>      |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123574 |                               | Implementations differ as to whether they write all output to the default output file or split the output into separate per-domain files. Portable applications can either ensure that each C-language source file contains calls to <i>gettext</i> family functions for only a single domain, or force all output to be to the default output file by using the <b>-K</b> option to override the default keywords. |
| 123575 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123576 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123577 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123578 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123579 |                               | Some implementations of <i>xgettext</i> are not able to extract cast strings (unless <b>-a</b> is used), for example casts of literal strings to ( <b>const char *</b> ). Use of a cast is unnecessary anyway, since the prototypes in <b>&lt;libintl.h&gt;</b> already specify this type.                                                                                                                          |
| 123580 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 123581 |                               |                                                                                                                                                                                                                                                                                                                                                                                                                     |

123582 The *xgettext* utility is not required to handle C preprocessor directives. Therefore if, for example,  
 123583 calls to *gettext* family functions are wrapped by macros, they might not be found unless the **-K**  
 123584 option is used to tell *xgettext* to look for the macro calls.

## 123585 EXAMPLES

### 123586 Example 1

123587 The following example shows how **-K** can be used to force all output to be to the default output  
 123588 file:

```
123589 xgettext -K "" -K gettext:1 -K dgettext:2 -K dcgettext:2 \  

  123590 -K ngettext:1,2 -K dngettext:2,3 -K dcngettext:2,3 source.c
```

123591 By overriding the default keywords using the **-K** option as above, the *xgettext* utility is directed  
 123592 to ignore the *domainname* arguments to the *dgettext()*, *dcgettext()*, *dngettext()*, and *dcngettext()*  
 123593 functions. Thus, the utility treats the functions as their respective equivalent without the *d* prefix,  
 123594 ignoring the *domainname* argument and writing generated output to the default output file,  
 123595 **messages.po**. Additional **-K** options would be needed for the variants of the functions with an  
 123596 *\_l* suffix if they are used.

### 123597 Example 2

123598 If the source uses a macro definition such as:

```
123599 #define i18n gettext
```

123600 the use of:

```
123601 xgettext -K i18n:1 source.c
```

123602 will pick up **msgid** values from a line such as:

```
123603 fprintf(stdout, i18n("The value is %s"), value1);
```

## 123604 RATIONALE

123605 The **-K** option is based on the **-k** option of GNU *xgettext*; the only difference is that GNU's **-k**  
 123606 takes an optional option-argument whereas **-K** in this standard has a mandatory option-  
 123607 argument in order to comply with the syntax guidelines.

123608 The standard developers considered including functionality equivalent to the **-c**, **-m**, and **-M**  
 123609 options in existing implementations. However, those letters could not be used as the syntax  
 123610 differed between implementations. The usual solution of adding an uppercase equivalent of  
 123611 lowercase options with the standard syntax instead was not possible, for obvious reasons for **-m**  
 123612 and **-M**, and as **-C** was already in use for another purpose in one implementation.

123613 The **-s** option is not included as it has been deprecated in at least one implementation because it  
 123614 has been found to deprive translators of valuable context.

## 123615 FUTURE DIRECTIONS

123616 If this utility is directed to create a new directory entry that contains any bytes that have the  
 123617 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 123618 error. A future version of this standard may require implementations to treat this as an error.

123619 A future version of this standard may change the description of the **-n** option to mandate the  
 123620 given comment format (by using ``shall'' instead of ``should'').

## 123621 SEE ALSO

123622 *gettext*, *msgfmt*

123623 XBD Chapter 8 (on page 167), Section 12.2 (on page 215)

123624 XSH *xgettext*  
123625 **CHANGE HISTORY**  
123626 First released in Issue 8.

123627 **NAME**123628 yacc — yet another compiler compiler (**DEVELOPMENT**)123629 **SYNOPSIS**123630 CD `yacc [-dltv] [-b file_prefix] [-p sym_prefix] grammar`123631 **DESCRIPTION**

123632 The *yacc* utility shall read a description of a context-free grammar in *grammar* and write C source  
 123633 code, conforming to the ISO C standard, to a code file, and optionally header information into a  
 123634 header file, in the current directory. The generated source code shall not depend on any  
 123635 undefined, unspecified, or implementation-defined behavior, except in cases where it is copied  
 123636 directly from the supplied grammar, or in cases that are documented by the implementation.  
 123637 The C code shall define a function and related routines and macros for an automaton that  
 123638 executes a parsing algorithm meeting the requirements in [Algorithms](#) (on page 3625).

123639 The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

123640 The C source code and header file shall be produced in a form suitable as input for the C  
 123641 compiler (see [c17](#)).

123642 **OPTIONS**

123643 The *yacc* utility shall conform to XBD [Section 12.2](#) (on page 215), except for Guideline 9.

123644 The following options shall be supported:

123645 **-b *file\_prefix*** Use *file\_prefix* instead of **y** as the prefix for all output filenames. The code file  
 123646 **y.tab.c**, the header file **y.tab.h** (created when **-d** is specified), and the description  
 123647 file **y.output** (created when **-v** is specified), shall be changed to *file\_prefix.tab.c*,  
 123648 *file\_prefix.tab.h*, and *file\_prefix.output*, respectively.

123649 **-d** Write the header file; by default only the code file is written. See the OUTPUT  
 123650 FILES section.

123651 **-l** Produce a code file that does not contain any **#line** constructs. If this option is not  
 123652 present, it is unspecified whether the code file or header file contains **#line**  
 123653 directives. This should only be used after the grammar and the associated actions  
 123654 are fully debugged.

123655 **-p *sym\_prefix***

123656 Use *sym\_prefix* instead of **yy** as the prefix for all external names produced by *yacc*.  
 123657 The names affected shall include the functions *yyparse()*, *yylex()*, and *yyerror()*, and  
 123658 the variables *yyval*, *yychar*, and *yydebug*. (In the remainder of this section, the six  
 123659 symbols cited are referenced using their default names only as a notational  
 123660 convenience.) Local names may also be affected by the **-p** option; however, the **-p**  
 123661 option shall not affect **#define** symbols generated by *yacc*.

123662 **-t** Modify conditional compilation directives to permit compilation of debugging  
 123663 code in the code file. Runtime debugging statements shall always be contained in  
 123664 the code file, but by default conditional compilation directives prevent their  
 123665 compilation.

123666 **-v** Write a file containing a description of the parser and a report of conflicts  
 123667 generated by ambiguities in the grammar.

123668 **OPERANDS**

123669 The following operand is required:

123670 *grammar* A pathname of a file containing instructions, hereafter called *grammar*, for which a  
 123671 parser is to be created. The format for the grammar is described in the EXTENDED  
 123672 DESCRIPTION section.

123673 **STDIN**

123674 Not used.

123675 **INPUT FILES**

123676 The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION  
 123677 section.

123678 **ENVIRONMENT VARIABLES**123679 The following environment variables shall affect the execution of *yacc*:

123680 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 123681 (See XBD Section 8.2 (on page 169) for the precedence of internationalization  
 123682 variables used to determine the values of locale categories.)

123683 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 123684 internationalization variables.

123685 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 123686 characters (for example, single-byte as opposed to multi-byte characters in  
 123687 arguments and input files).

123688 *LC\_MESSAGES*

123689 Determine the locale that should be used to affect the format and contents of  
 123690 diagnostic messages written to standard error.

123691 *XSHELL* *NLSPATH* Determine the location of messages objects and message catalogs.

123692 The *LANG* and *LC\_\** variables affect the execution of the *yacc* utility as stated. The *main()*  
 123693 function defined in [Yacc Library](#) (on page 3624) shall call:

123694 `setlocale(LC_ALL, "")`

123695 and thus the program generated by *yacc* shall also be affected by the contents of these variables  
 123696 at runtime.

123697 **ASYNCHRONOUS EVENTS**

123698 Default.

123699 **STDOUT**

123700 Not used.

123701 **STDERR**

123702 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* shall write a report of  
 123703 those conflicts to the standard error in an unspecified format.

123704 Standard error shall also be used for diagnostic messages.

123705 **OUTPUT FILES**

123706 The code file, the header file, and the description file shall be text files. All are described in the  
 123707 following sections.

123708 **Code File**

123709 This file shall contain the C source code for the *yyparse()* function. It shall contain code for the  
 123710 various semantic actions with macro substitution performed on them as described in the  
 123711 EXTENDED DESCRIPTION section. Preceding this code it shall contain an `extern int`  
 123712 `yychar` declaration or `int yychar` definition, and **#define** statements for the following  
 123713 macros:

123714 **YYEMPTY** Token number indicating there is no lookahead token. This macro shall expand to  
 123715 an integer constant with a value less than zero, protected by parentheses.

123716 **YYEOF** Token number indicating the end of input. This macro shall expand to the value 0.

123717 It also shall contain a copy of the **#define** statements in the header file, prior to any code copied  
 123718 from semantic actions in *grammar*, and the following function prototypes for the *yyerror()*,  
 123719 *yylex()*, and *yyparse()* functions, after any code copied from within `{` and `}` in the declarations  
 123720 section in *grammar* and before any code copied from semantic actions in *grammar*:

```
123721 void yyerror(const char *);
123722 int yylex(void);
123723 int yyparse(void);
```

123724 The declarations of *yyerror()* and *yylex()* shall be protected by **#ifndef** or **#if** preprocessor  
 123725 statements such that each is only visible if a preprocessor macro with the name *yyerror* or *yylex*,  
 123726 respectively, is not already defined, where the *yy* in the macro names is replaced by *sym\_prefix* if  
 123727 the `-p sym_prefix` option is used.

123728 If a **%union** declaration is used, the declaration for **YYSTYPE** and an `extern YYSTYPE`  
 123729 `yylval` declaration or **YYSTYPE** `yylval` definition shall also be included in this file.

123730 The code file shall not contain a declaration of the *main()* function, unless one is present within  
 123731 `{` and `}` in the declarations section in *grammar*.

123732 **Header File**

123733 The header file shall contain **#define** statements that associate the token numbers with the token  
 123734 names. This allows source files other than the code file to access the token codes. If a **%union**  
 123735 declaration is used, the declaration for **YYSTYPE** and an `extern YYSTYPE` `yylval`  
 123736 declaration shall also be included in this file. The header file may also declare the *yyparse()*  
 123737 function, using a function prototype. It shall not declare the *yyerror()* and *yylex()* functions.

123738 **Description File**

123739 The description file shall be a text file containing a description of the state machine  
 123740 corresponding to the parser, using an unspecified format. Limits for internal tables (see [Limits](#),  
 123741 on page 3625) shall also be reported, in an implementation-defined manner. (Some  
 123742 implementations may use dynamic allocation techniques and have no specific limit values to  
 123743 report.)

123744 **EXTENDED DESCRIPTION**

123745 The *yacc* command accepts a language that is used to define a grammar for a target language to  
 123746 be parsed by the tables and code generated by *yacc*. The language accepted by *yacc* as a  
 123747 grammar for the target language is described below using the *yacc* input language itself.

123748 The input *grammar* includes rules describing the input structure of the target language and code  
 123749 to be invoked when these rules are recognized to provide the associated semantic action. The  
 123750 code to be executed shall appear as bodies of text that are intended to be C-language code. These  
 123751 bodies of text shall not contain C-language trigraphs. The C-language inclusions are presumed

123752 to form a correct function when processed by *yacc* into its output files. The code included in this  
123753 way shall be executed during the recognition of the target language.

123754 Given a grammar, the *yacc* utility generates the files described in the OUTPUT FILES section.  
123755 The code file can be compiled and linked using *c17*. If the declaration and programs sections of  
123756 the grammar file did not include definitions of *main()*, *yylex()*, and *yyerror()*, the compiled  
123757 output requires linking with externally supplied versions of those functions. Default versions of  
123758 *main()* and *yyerror()* are supplied in the *yacc* library and can be linked in by using the `-I y`  
123759 operand to *c17*. The *yacc* library interfaces need not support interfaces with other than the  
123760 default **yy** symbol prefix. The application provides the lexical analyzer function, *yylex()*; the *lex*  
123761 utility is specifically designed to generate such a routine.

### 123762 **Input Language**

123763 The application shall ensure that every specification file consists of three sections in order:  
123764 *declarations*, *grammar rules*, and *programs*, separated by double <percent-sign> characters ("%").  
123765 The declarations and programs sections can be empty. If the latter is empty, the preceding "%"  
123766 mark separating it from the rules section can be omitted.

123767 The input is free form text following the structure of the grammar defined below.

### 123768 **Lexical Structure of the Grammar**

123769 The <blank>, <newline>, and <form-feed> character shall be ignored, except that the application  
123770 shall ensure that they do not appear in names or multi-character reserved symbols. Comments  
123771 shall be enclosed in `"/ * . . . */`, and can appear wherever a name is valid.

123772 Names are of arbitrary length, made up of letters, periods ('.'), underscores ('\_'), and non-  
123773 initial digits. Uppercase and lowercase letters are distinct. Conforming applications shall not  
123774 use names beginning in **yy** or **YY** since the *yacc* parser uses such names. Many of the names  
123775 appear in the final output of *yacc*, and thus they should be chosen to conform with any  
123776 additional rules created by the C compiler to be used. In particular they appear in **#define**  
123777 statements.

123778 A literal shall consist of a single character enclosed in single-quote characters. All of the escape  
123779 sequences supported for character constants by the ISO C standard shall be supported by *yacc*.

123780 The relationship with the lexical analyzer is discussed in detail below.

123781 The application shall ensure that the NUL character is not used in grammar rules or literals.

### 123782 **Declarations Section**

123783 The declarations section is used to define the symbols used to define the target language and  
123784 their relationship with each other. In particular, much of the additional information required to  
123785 resolve ambiguities in the context-free grammar for the target language is provided here.

123786 Usually *yacc* assigns the relationship between the symbolic names it generates and their  
123787 underlying numeric value. The declarations section makes it possible to control the assignment  
123788 of these values.

123789 It is also possible to keep semantic information associated with the tokens currently on the parse  
123790 stack in a user-defined C-language **union**, if the members of the union are associated with the  
123791 various names in the grammar. The declarations section provides for this as well.

123792 The first group of declarators below all take a list of names as arguments. That list can optionally  
123793 be preceded by the name of a C union member (called a *tag* below) appearing within '<' and  
123794 '>'. (As an exception to the typographical conventions of the rest of this volume of



123795 POSIX.1-2024, in this case `<tag>` does not represent a metavariable, but the literal angle bracket  
 123796 characters surrounding a symbol.) The use of `tag` specifies that the tokens named on this line  
 123797 shall be of the same C type as the union member referenced by `tag`. This is discussed in more  
 123798 detail below.

123799 For lists used to define tokens, the first appearance of a given token can be followed by a  
 123800 positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it  
 123801 for lexical purposes shall be taken to be that number.

123802 The following declares `name` to be a token:

```
123803 %token [<tag>] name [number] [name [number]]...
```

123804 If `tag` is present, the C type for all tokens on this line shall be declared to be the type referenced  
 123805 by `tag`. If a positive integer, `number`, follows a `name`, that value shall be assigned to the token.

123806 The following declares `name` to be a token, and assigns precedence to it:

```
123807 %left [<tag>] name [number] [name [number]]...
123808 %right [<tag>] name [number] [name [number]]...
```

123809 One or more lines, each beginning with one of these symbols, can appear in this section. All  
 123810 tokens on the same line have the same precedence level and associativity; the lines are in order  
 123811 of increasing precedence or binding strength. `%left` denotes that the operators on that line are  
 123812 left associative, and `%right` similarly denotes right associative operators. If `tag` is present, it shall  
 123813 declare a C type for `names` as described for `%token`.

123814 The following declares `name` to be a token, and indicates that this cannot be used associatively:

```
123815 %nonassoc [<tag>] name [number] [name [number]]...
```

123816 If the parser encounters associative use of this token it reports an error. If `tag` is present, it shall  
 123817 declare a C type for `names` as described for `%token`.

123818 The following declares that union member `names` are non-terminals, and thus it is required to  
 123819 have a `tag` field at its beginning:

```
123820 %type <tag> name...
```

123821 Because it deals with non-terminals only, assigning a token number or using a literal is also  
 123822 prohibited. If this construct is present, `yacc` shall perform type checking; if this construct is not  
 123823 present, the parse stack shall hold only the `int` type.

123824 Every name used in `grammar` not defined by a `%token`, `%left`, `%right`, or `%nonassoc` declaration  
 123825 is assumed to represent a non-terminal symbol. The `yacc` utility shall report an error for any non-  
 123826 terminal symbol that does not appear on the left side of at least one grammar rule.

123827 Once the type, precedence, or token number of a name is specified, it shall not be changed. If the  
 123828 first declaration of a token does not assign a token number, `yacc` shall assign a token number.  
 123829 Once this assignment is made, the token number shall not be changed by explicit assignment.

123830 The following declarators do not follow the previous pattern.

123831 The following declares the non-terminal `name` to be the *start symbol*, which represents the largest,  
 123832 most general structure described by the grammar rules:

```
123833 %start name
```

123834 By default, it is the left-hand side of the first grammar rule; this default can be overridden with  
 123835 this declaration.

123836 The following declares the `yacc` value stack to be a union of the various types of values desired.

123837       %union { *body of union (in C)* }

123838       The body of the union shall not contain unbalanced curly brace preprocessing tokens.

123839       By default, the values returned by actions (see below) and the lexical analyzer shall be of type  
123840       **int**. The *yacc* utility keeps track of types, and it shall insert corresponding union member names  
123841       in order to perform strict type checking of the resulting parser.

123842       Alternatively, given that at least one *<tag>* construct is used, the union can be declared in a  
123843       header file (which shall be included in the declarations section by using a **#include** construct  
123844       within *%{* and *%}*), and a **typedef** used to define the symbol *YYSTYPE* to represent this union.  
123845       The effect of **%union** is to provide the declaration of *YYSTYPE* directly from the *yacc* input.

123846       C-language declarations and definitions can appear in the declarations section, enclosed by the  
123847       following marks:

123848       %{ . . . % }

123849       These statements shall be copied into the code file, and have global scope within it so that they  
123850       can be used in the rules and program sections. The statements shall not contain *"%}"* outside a  
123851       comment, string literal, or multi-character constant.

123852       The application shall ensure that the declarations section is terminated by the token *%%*.

### 123853       **Grammar Rules in yacc**

123854       The rules section defines the context-free grammar to be accepted by the function *yacc* generates,  
123855       and associates with those rules C-language actions and additional precedence information. The  
123856       grammar is described below, and a formal definition follows.

123857       The rules section is comprised of one or more grammar rules. A grammar rule has the form:

123858       A : BODY ;

123859       The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or  
123860       more *names*, *literals*, and *semantic actions* that can then be followed by optional *precedence rules*.  
123861       Only the names and literals participate in the formation of the grammar; the semantic actions  
123862       and precedence rules are used in other ways. The *<colon>* and the *<semicolon>* are *yacc*  
123863       punctuation. If there are several successive grammar rules with the same left-hand side, the  
123864       *<vertical-line>* (*'|'*) can be used to avoid rewriting the left-hand side; in this case the  
123865       *<semicolon>* appears only after the last rule. The *BODY* part can be empty (or empty of names  
123866       and literals) to indicate that the non-terminal symbol matches the empty string.

123867       The *yacc* utility assigns a unique number to each rule. Rules using the vertical bar notation are  
123868       distinct rules. The number assigned to the rule appears in the description file.

123869       The elements comprising a *BODY* are:

123870       *name, literal*   These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*  
123871       stands for itself (less the lexically required quotation marks).

123872       *semantic action*

123873       With each grammar rule, the user can associate actions to be performed each time  
123874       the rule is recognized in the input process. (Note that the word *"action"* can also  
123875       refer to the actions of the parser—shift, reduce, and so on.)

123876       These actions can return values and can obtain the values returned by previous  
123877       actions. These values are kept in objects of type *YYSTYPE* (see **%union**). The  
123878       result value of the action shall be kept on the parse stack with the left-hand side of  
123879       the rule, to be accessed by other reductions as part of their right-hand side. By

123880 using the `<tag>` information provided in the declarations section, the code  
 123881 generated by *yacc* can be strictly type checked and contain arbitrary information. In  
 123882 addition, the lexical analyzer can provide the same kinds of values for tokens, if  
 123883 desired.

123884 An action is an arbitrary C statement and as such can do input or output, call  
 123885 subprograms, and alter external variables. An action is one or more C statements  
 123886 enclosed in curly braces '{' and '}'. The statements shall not contain  
 123887 unbalanced curly brace preprocessing tokens.

123888 Certain pseudo-variables can be used in the action. These are macros for access to  
 123889 data structures known internally to *yacc*.

123890 **\$\$** The value of the action can be set by assigning it to **\$\$**. If type  
 123891 checking is enabled and the type of the value to be assigned cannot  
 123892 be determined, a diagnostic message may be generated.

123893 **\$number** This refers to the value returned by the component specified by the  
 123894 token *number* in the right side of a rule, reading from left to right;  
 123895 *number* can be zero or negative. If *number* is zero or negative, it refers  
 123896 to the data associated with the name on the parser's stack preceding  
 123897 the leftmost symbol of the current rule. (That is, "\$0" refers to the  
 123898 name immediately preceding the leftmost name in the current rule to  
 123899 be found on the parser's stack and "\$-1" refers to the symbol to *its*  
 123900 left.) If *number* refers to an element past the current point in the rule,  
 123901 or beyond the bottom of the stack, the result is undefined. If type  
 123902 checking is enabled and the type of the value to be assigned cannot  
 123903 be determined, a diagnostic message may be generated.

123904 **\$<tag>number**  
 123905 These correspond exactly to the corresponding symbols without the  
 123906 *tag* inclusion, but allow for strict type checking (and preclude  
 123907 unwanted type conversions). The effect is that the macro is expanded  
 123908 to use *tag* to select an element from the YYSTYPE union (using  
 123909 *dataname.tag*). This is particularly useful if *number* is not positive.

123910 **\$<tag>\$** This imposes on the reference the type of the union member  
 123911 referenced by *tag*. This construction is applicable when a reference to  
 123912 a left context value occurs in the grammar, and provides *yacc* with a  
 123913 means for selecting a type.

123914 Actions can occur anywhere in a rule (not just at the end); an action can access  
 123915 values returned by actions to its left, and in turn the value it returns can be  
 123916 accessed by actions to its right. An action appearing in the middle of a rule shall be  
 123917 equivalent to replacing the action with a new non-terminal symbol and adding an  
 123918 empty rule with that non-terminal symbol on the left-hand side. The semantic  
 123919 action associated with the new rule shall be equivalent to the original action. The  
 123920 use of actions within rules might introduce conflicts that would not otherwise  
 123921 exist.

123922 By default, the value of a rule shall be the value of the first element in it. If the first  
 123923 element does not have a type (particularly in the case of a literal) and type  
 123924 checking is turned on by **%type**, an error message shall result.

123925 *precedence* The keyword **%prec** can be used to change the precedence level associated with a  
 123926 particular grammar rule. Examples of this are in cases where a unary and binary  
 123927 operator have the same symbolic representation, but need to be given different  
 123928 precedences, or where the handling of an ambiguous if-else construction is  
 123929 necessary. The reserved symbol **%prec** can appear immediately after the body of  
 123930 the grammar rule and can be followed by a token name or a literal. It shall cause  
 123931 the precedence of the grammar rule to become that of the following token name or  
 123932 literal. The action for the rule as a whole can follow **%prec**.

123933 If a program section follows, the application shall ensure that the grammar rules are terminated  
 123934 by **%%**.

### 123935 **Programs Section**

123936 The *programs* section can include the definition of the lexical analyzer *yylex()*, and any other  
 123937 functions; for example, those used in the actions specified in the grammar rules. It is unspecified  
 123938 whether the programs section precedes or follows the semantic actions in the output file;  
 123939 therefore, if the application contains any macro definitions and declarations intended to apply to  
 123940 the code in the semantic actions, it shall place them within "**%{ . . . %}**" in the declarations  
 123941 section.

### 123942 **Input Grammar**

123943 The following input to *yacc* yields a parser for the input to *yacc*. This formal syntax takes  
 123944 precedence over the preceding text syntax description.

123945 The lexical structure is defined less precisely; [Lexical Structure of the Grammar](#) (on page 3616)  
 123946 defines most terms. The correspondence between the previous terms and the tokens below is as  
 123947 follows.

123948 **IDENTIFIER** This corresponds to the concept of *name*, given previously. It also includes  
 123949 literals as defined previously.

123950 **C\_IDENTIFIER** This is a name, and additionally it is known to be followed by a <colon>. A  
 123951 literal cannot yield this token.

123952 **NUMBER** A string of digits (a non-negative decimal integer).

123953 **TYPE, LEFT, MARK, LCURL, RCURL**

123954 These correspond directly to **%type**, **%left**, **%%**, **%{**, and **%}**.

123955 **{...}** This indicates C-language source code, with the possible inclusion of '\$'  
 123956 macros as discussed previously.

```

123957 /* Grammar for the input to yacc. */
123958 /* Basic entries. */
123959 /* The following are recognized by the lexical analyzer. */

123960 %token IDENTIFIER /* Includes identifiers and literals */
123961 %token C_IDENTIFIER /* identifier (but not literal)
123962 followed by a :. */
123963 %token NUMBER /* [0-9][0-9]* */

123964 /* Reserved words : %type=>TYPE %left=>LEFT, and so on */

123965 %token LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

123966 %token MARK /* The %% mark. */
123967 %token LCURL /* The %{ mark. */

```

```

123968      %token      RCURL          /* The %} mark. */
123969      /* 8-bit character literals stand for themselves; */
123970      /* tokens have to be defined for multi-byte characters. */
123971      %start      spec
123972      %%
123973      spec : defs MARK rules tail
123974          ;
123975      tail : MARK
123976          {
123977          /* In this action, set up the rest of the file. */
123978          }
123979          | /* Empty; the second MARK is optional. */
123980          ;
123981      defs : /* Empty. */
123982          | defs def
123983          ;
123984      def  : START IDENTIFIER
123985          | UNION
123986          {
123987          /* Copy union definition to output. */
123988          }
123989          | LCURL
123990          {
123991          /* Copy C code to output file. */
123992          }
123993          | RCURL
123994          | rword tag nlist
123995          ;
123996      rword : TOKEN
123997          | LEFT
123998          | RIGHT
123999          | NONASSOC
124000          | TYPE
124001          ;
124002      tag  : /* Empty: union tag ID optional. */
124003          | '<' IDENTIFIER '>'
124004          ;
124005      nlist : nmno
124006          | nlist nmno
124007          ;
124008      nmno : IDENTIFIER          /* Note: literal invalid with % type. */
124009          | IDENTIFIER NUMBER /* Note: invalid with % type. */
124010          ;
124011      /* Rule section */
124012      rules : C_IDENTIFIER rbody prec
124013          | rules rule
124014          ;
124015      rule  : C_IDENTIFIER rbody prec
124016          | '|' rbody prec

```

```

124017         ;
124018 rbody : /* empty */
124019         | rbody IDENTIFIER
124020         | rbody act
124021         ;
124022 act   : '{'
124023         {
124024         /* Copy action, translate $$, and so on. */
124025         }
124026         '}'
124027         ;
124028 prec  : /* Empty */
124029         | PREC IDENTIFIER
124030         | PREC IDENTIFIER act
124031         | prec ';'
124032         ;

```

### 124033 Conflicts

124034 The parser produced for an input grammar may contain states in which conflicts occur. The  
 124035 conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at  
 124036 least one LALR(1) conflict. The *yacc* utility shall resolve all conflicts, using either default rules or  
 124037 user-specified precedence rules.

124038 Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is  
 124039 where, for a given state and lookahead symbol, both a shift action and a reduce action are  
 124040 possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions  
 124041 by two different rules are possible.

124042 The rules below describe how to specify what actions to take when a conflict occurs. Not all  
 124043 shift/reduce conflicts can be successfully resolved this way because the conflict may be due to  
 124044 something other than ambiguity, so incautious use of these facilities can cause the language  
 124045 accepted by the parser to be much different from that which was intended. The description file  
 124046 shall contain sufficient information to understand the cause of the conflict. Where ambiguity is  
 124047 the reason either the default or explicit rules should be adequate to produce a working parser.

124048 The declared precedences and associativities (see [Declarations Section](#), on page 3616) are used to  
 124049 resolve parsing conflicts as follows:

- 124050 1. A precedence and associativity is associated with each grammar rule; it is the precedence  
 124051 and associativity of the last token or literal in the body of the rule. If the `%prec` keyword  
 124052 is used, it overrides this default. Some grammar rules might not have both precedence  
 124053 and associativity.
- 124054 2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have  
 124055 precedence and associativity associated with them, then the conflict is resolved in favor of  
 124056 the action (shift or reduce) associated with the higher precedence. If the precedences are  
 124057 the same, then the associativity is used; left associative implies reduce, right associative  
 124058 implies shift, and non-associative implies an error in the string being parsed.
- 124059 3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done.  
 124060 Conflicts resolved this way are counted in the diagnostic output described in [Error  
 124061 Handling](#) (on page 3623).

124062 4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that  
124063 occurs earlier in the input sequence. Conflicts resolved this way are counted in the  
124064 diagnostic output described in [Error Handling](#).

124065 Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and  
124066 reduce/reduce conflicts reported by *yacc* on either standard error or in the description file.

## 124067 **Error Handling**

124068 The token **error** shall be reserved for error handling. The name **error** can be used in grammar  
124069 rules. It indicates places where the parser can recover from a syntax error. The default value of  
124070 **error** shall be 256. Its value can be changed using a **%token** declaration. The lexical analyzer  
124071 should not return the value of **error**.

124072 The parser shall detect a syntax error when it is in a state where the action associated with the  
124073 lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by  
124074 executing the macro YYERROR. When YYERROR is executed, the semantic action passes control  
124075 back to the parser. YYERROR cannot be used outside of semantic actions.

124076 When the parser detects a syntax error, it normally calls *yyerror()* with the character string  
124077 "syntax error" as its argument. The call shall not be made if the parser is still recovering  
124078 from a previous error when the error is detected. The parser is considered to be recovering from  
124079 a previous error until the parser has shifted over at least three normal input symbols since the  
124080 last error was detected or a semantic action has executed the macro *yyerrok*. The parser shall not  
124081 call *yyerror()* when YYERROR is executed.

124082 The macro function YYRECOVERING shall return 1 if a syntax error has been detected and the  
124083 parser has not yet fully recovered from it. Otherwise, zero shall be returned.

124084 When a syntax error is detected by the parser, the parser shall check if a previous syntax error  
124085 has been detected. If a previous error was detected, and if no normal input symbols have been  
124086 shifted since the preceding error was detected, the parser checks if the lookahead symbol is an  
124087 endmarker (see [Interface to the Lexical Analyzer](#), on page 3624). If it is, the parser shall return  
124088 with a non-zero value. Otherwise, the lookahead symbol shall be discarded and normal parsing  
124089 shall resume.

124090 When YYERROR is executed or when the parser detects a syntax error and no previous error has  
124091 been detected, or at least one normal input symbol has been shifted since the previous error was  
124092 detected, the parser shall pop back one state at a time until the parse stack is empty or the  
124093 current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a  
124094 non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser  
124095 reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead  
124096 symbol when parsing is resumed.

124097 The macro *yyerrok* in a semantic action shall cause the parser to act as if it has fully recovered  
124098 from any previous errors. The macro *yyclearin* shall cause the parser to discard the current  
124099 lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no  
124100 effect.

124101 The macro YYACCEPT shall cause the parser to return with the value zero. The macro  
124102 YYABORT shall cause the parser to return with a non-zero value.

124103 **Interface to the Lexical Analyzer**

124104 The application shall ensure that the *yylex()* function is an integer-valued function that returns a  
 124105 *token number* greater than zero representing the kind of token read, or a value less than or equal  
 124106 to zero when the end of input is reached. When the parser generated by *yacc* calls *yylex()*, it shall  
 124107 assign the returned value, if greater than zero, to the external variable *yychar*. If there is a value  
 124108 associated with the returned token (see the discussion of *tag* above), it shall be assigned to the  
 124109 external variable *yyval*. If the return value from *yylex()* is less than or equal to zero, the parser  
 124110 shall assign the value *YYEOF* to *yychar*.

124111 If the parser and *yylex()* do not agree on these token numbers, reliable communication between  
 124112 them cannot occur. For (single-byte character) literals, the token is simply the numeric value of  
 124113 the character in the current character set. The numbers for other tokens can either be chosen by  
 124114 *yacc*, or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex()* to  
 124115 return these numbers symbolically. The **#define** statements are put into the code file, and the  
 124116 header file if that file is requested. The set of characters permitted by *yacc* in an identifier is  
 124117 larger than that permitted by C. Token names found to contain such characters shall not be  
 124118 included in the **#define** declarations.

124119 If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers  
 124120 greater than 256, although no order is implied. A token can be explicitly assigned a number by  
 124121 following its first appearance in the declarations section with a number. Names and literals not  
 124122 defined this way retain their default definition. All token numbers assigned by *yacc* shall be  
 124123 unique and distinct from the token numbers used for literals and user-assigned tokens. If  
 124124 duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error;  
 124125 otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

124126 When a parser action is executed, *yychar* shall hold either the token number of the lookahead  
 124127 token, or *YYEMPTY* indicating that there is no lookahead token, or *YYEOF* indicating the end of  
 124128 input. If *yychar* holds the token number of the lookahead token, *yyval* shall hold the value  
 124129 associated with that token, if any.

124130 The application shall ensure that when the end of input is reached, the *yylex()* function returns a  
 124131 value that is zero or negative. The parser shall treat this as the token number *YYEOF* for a  
 124132 special token called the *endmarker*. If the tokens up to, but excluding, the endmarker form a  
 124133 structure that matches the start symbol, the parser shall accept the input. If the endmarker is  
 124134 seen in any other context, it shall be considered an error.

124135 **Completing the Program**

124136 In addition to *yparse()* and *yylex()*, the functions *yyerror()* and *main()* are required to make a  
 124137 complete program. The application can supply *main()* and *yyerror()*, or those routines can be  
 124138 obtained from the *yacc* library.

124139 **Yacc Library**

124140 The following functions shall appear only in the *yacc* library accessible through the **-I y** operand  
 124141 to *c17*; they can therefore be redefined by a conforming application:

124142 **int main(void)**

124143 This function shall call *yparse()* and exit with an unspecified value. Other actions within  
 124144 this function are unspecified.

124145 **void yyerror(const char \*s)**

124146 This function shall write the NUL-terminated argument to standard error, followed by a  
 124147 <newline>.



124148 The order of the `-ly` and `-ll` operands given to `c17` is significant; the application shall either  
124149 provide its own `main()` function or ensure that `-ly` precedes `-ll`.

### 124150 **Debugging the Parser**

124151 The parser generated by `yacc` shall have diagnostic facilities in it that can be optionally enabled  
124152 at either compile time or at runtime (if enabled at compile time). The compilation of the runtime  
124153 debugging code is under the control of `YYDEBUG`, a preprocessor symbol. If `YYDEBUG` has a  
124154 non-zero value, the debugging code shall be included. If its value is zero, the code shall not be  
124155 included.

124156 In parsers where the debugging code has been included, the external `int yydebug` can be used to  
124157 turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of  
124158 `yydebug` shall be zero.

124159 When `-t` is specified, the code file shall be built such that, if `YYDEBUG` is not already defined at  
124160 compilation time (using the `c17 -D YYDEBUG` option, for example), `YYDEBUG` shall be set  
124161 explicitly to 1. When `-t` is not specified, the code file shall be built such that, if `YYDEBUG` is not  
124162 already defined, it shall be set explicitly to zero.

124163 The format of the debugging output is unspecified but includes at least enough information to  
124164 determine the shift and reduce actions, and the input symbols. It also provides information  
124165 about error recovery.

### 124166 **Algorithms**

124167 The parser constructed by `yacc` implements an LALR(1) parsing algorithm as documented in the  
124168 literature. It is unspecified whether the parser is table-driven or direct-coded.

124169 A parser generated by `yacc` shall never request an input symbol from `yylex()` while in a state  
124170 where the only actions other than the error action are reductions by a single rule.

124171 The literature of parsing theory defines these concepts.

### 124172 **Limits**

124173 The `yacc` utility may have several internal tables. The minimum maximums for these tables are  
124174 shown in the following table. The exact meaning of these values is implementation-defined. The  
124175 implementation shall define the relationship between these values and between them and any  
124176 error messages that the implementation may generate should it run out of space for any internal  
124177 structure. An implementation may combine groups of these resources into a single pool as long  
124178 as the total available to the user does not fall below the sum of the sizes specified by this section.

124179

Table 3-23 Internal Limits in *yacc*

124180

124181

124182

124183

124184

124185

124186

124187

124188

124189

124190

124191

124192

124193

124194

124195

124196

| Limit      | Minimum<br>Maximum | Description                                                                                                                                                                                                                                                        |
|------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {NTERMS}   | 126                | Number of tokens.                                                                                                                                                                                                                                                  |
| {NNONTERM} | 200                | Number of non-terminals.                                                                                                                                                                                                                                           |
| {NPROD}    | 300                | Number of rules.                                                                                                                                                                                                                                                   |
| {NSTATES}  | 600                | Number of states.                                                                                                                                                                                                                                                  |
| {MEMSIZE}  | 5 200              | Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in <a href="#">Grammar Rules in yacc</a> (on page 3618). |
| {ACTSIZE}  | 4 000              | Number of actions. “Actions” here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in <a href="#">Grammar Rules in yacc</a> (on page 3618).                                                        |

124197 **EXIT STATUS**

124198 The following exit values shall be returned:

124199 0 Successful completion.

124200 &gt;0 An error occurred.

124201 **CONSEQUENCES OF ERRORS**

124202 If any errors are encountered, the run is aborted and *yacc* exits with a non-zero status. Partial  
 124203 code files and header files may be produced. The summary information in the description file  
 124204 shall always be produced if the `-v` flag is present.

124205 **APPLICATION USAGE**

124206 Historical implementations experience name conflicts on the names `yacc.tmp`, `yacc.acts`,  
 124207 `yacc.debug`, `y.tab.c`, `y.tab.h`, and `y.output` if more than one copy of *yacc* is running in a single  
 124208 directory at one time. The `-b` option was added to overcome this problem. The related problem  
 124209 of allowing multiple *yacc* parsers to be placed in the same file was addressed by adding a `-p`  
 124210 option to override the previously hard-coded `yy` variable prefix.

124211 The description of the `-p` option specifies the minimal set of function and variable names that  
 124212 cause conflict when multiple parsers are linked together. `YYSTYPE` does not need to be changed.  
 124213 Instead, the programmer can use `-b` to give the header files for different parsers different names,  
 124214 and then the file with the `yylex()` for a given parser can include the header for that parser.  
 124215 Names such as `yyclearerr` do not need to be changed because they are used only in the actions;  
 124216 they do not have linkage. It is possible that an implementation has other names, either internal  
 124217 ones for implementing things such as `yyclearerr`, or providing non-standard features that it wants  
 124218 to change with `-p`.

124219 Unary operators that are the same token as a binary operator in general need their precedence  
 124220 adjusted. This is handled by the `%prec` advisory symbol associated with the particular grammar  
 124221 rule defining that unary operator. (See [Grammar Rules in yacc](#) (on page 3618).) Applications are  
 124222 not required to use this operator for unary operators, but the grammars that do not require it are  
 124223 rare.

124224 If `yyerror()` and `yylex()` are not defined within `%{` and `%}` in the declarations section as functions  
 124225 or macros, nor in the programs section as functions, recommended practice is to declare them as

124226 functions in a separate header file and include that file in the declarations section, followed by  
 124227 `#define yyerror yyerror` and `#define yylex yylex`. This lets the separate header file  
 124228 be the definitive API for all code defining or using these functions.

#### 124229 EXAMPLES

124230 Access to the *yacc* library is obtained with library search operands to *c17*. To use the *yacc* library  
 124231 *main()*:

```
124232 c17 y.tab.c -l y
```

124233 Both the *lex* library and the *yacc* library contain *main()*. To access the *yacc main()*:

```
124234 c17 y.tab.c lex.yy.c -l y -l l
```

124235 This ensures that the *yacc* library is searched first, so that its *main()* is used.

124236 The historical *yacc* libraries have contained two simple functions that are normally coded by the  
 124237 application programmer. These functions are similar to the following code:

```
124238 #include <locale.h>
124239 int main(void)
124240 {
124241     extern int yyparse();
124242     setlocale(LC_ALL, "");
124243     /* If the following parser is one created by lex, the
124244        application must be careful to ensure that LC_CTYPE
124245        and LC_COLLATE are set to the POSIX locale. */
124246     (void) yyparse();
124247     return (0);
124248 }
124249 #include <stdio.h>
124250 void yyerror(const char *msg)
124251 {
124252     (void) fprintf(stderr, "%s\n", msg);
124253     return (0);
124254 }
```

#### 124255 RATIONALE

124256 The references in **Referenced Documents** may be helpful in constructing the parser generator.  
 124257 The referenced DeRemer and Pennello article (along with the works it references) describes a  
 124258 technique to generate parsers that conform to this volume of POSIX.1-2024. Work in this area  
 124259 continues to be done, so implementors should consult current literature before doing any new  
 124260 implementations. The original Knuth article is the theoretical basis for this kind of parser, but the  
 124261 tables it generates are impractically large for reasonable grammars and should not be used. The  
 124262 “equivalent to” wording is intentional to assure that the best tables that are LALR(1) can be  
 124263 generated.

124264 There has been confusion between the class of grammars, the algorithms needed to generate  
 124265 parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal.  
 124266 In particular, a parser generator that accepts the full range of LR(1) grammars need not generate  
 124267 a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars)  
 124268 for a grammar that happens to be SLR(1). Such an implementation need not recognize the case,  
 124269 either; table compression can yield the SLR(1) table (or one even smaller than that) without  
 124270 recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent

124271 more upon the table representation and compression (or the code generation if a direct parser is  
 124272 generated) than upon the class of grammar that the table generator handles.

124273 The speed of the parser generator is somewhat dependent upon the class of grammar it handles.  
 124274 However, the original Knuth article algorithms for constructing LR parsers were judged by its  
 124275 author to be impractically slow at that time. Although full LR is more complex than LALR(1), as  
 124276 computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock  
 124277 execution time) is becoming less significant.

124278 Potential authors are cautioned that the referenced DeRemer and Pennello article previously  
 124279 cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in  
 124280 some of the LALR(1) algorithm statements that preceded it to publication. They should take the  
 124281 time to seek out that paper, as well as current relevant work, particularly Aho's.

124282 The **-b** option was added to provide a portable method for permitting *yacc* to work on multiple  
 124283 separate parsers in the same directory. If a directory contains more than one *yacc* grammar, and  
 124284 both grammars are constructed at the same time (by, for example, a parallel *make* program),  
 124285 conflict results. While the solution is not historical practice, it corrects a known deficiency in  
 124286 historical implementations. Corresponding changes were made to all sections that referenced  
 124287 the filenames **y.tab.c** (now ``the code file''), **y.tab.h** (now ``the header file''), and **y.output** (now  
 124288 ``the description file'').

124289 The grammar for *yacc* input is based on System V documentation. The textual description shows  
 124290 there that the **' ; '** is required at the end of the rule. The grammar and the implementation do  
 124291 not require this. (The use of **C\_IDENTIFIER** causes a reduce to occur in the right place.)

124292 Also, in that implementation, the constructs such as **%token** can be terminated by a  
 124293 <semicolon>, but this is not permitted by the grammar. The keywords such as **%token** can also  
 124294 appear in uppercase, which is again not discussed. In most places where **'%'** is used,  
 124295 <backslash> can be substituted, and there are alternate spellings for some of the symbols (for  
 124296 example, **%LEFT** can be **"%<"** or even **"\<"**).

124297 Historically, <tag> can contain any characters except **'>'**, including white space, in the  
 124298 implementation. However, since the *tag* must reference an ISO C standard union member, in  
 124299 practice conforming implementations need to support only the set of characters for ISO C  
 124300 standard identifiers in this context.

124301 Some historical implementations are known to accept actions that are terminated by a period.  
 124302 Historical implementations often allow **'\$'** in names. A conforming implementation does not  
 124303 need to support either of these behaviors.

124304 Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of *yacc*. There  
 124305 may be situations in which the *grammar* is not, strictly speaking, in error, and yet *yacc* cannot  
 124306 interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances  
 124307 be resolved by providing additional information, such as using **%type** or **%union** declarations.  
 124308 It is often easier and it usually yields a smaller parser to take this alternative when it is  
 124309 appropriate.

124310 The size and execution time of a program produced without the runtime debugging code is  
 124311 usually smaller and slightly faster in historical implementations.

124312 Statistics messages from several historical implementations include the following types of  
 124313 information:

124314 *n*/512 terminals, *n*/300 non-terminals  
 124315 *n*/600 grammar rules, *n*/1500 states  
 124316 *n* shift/reduce, *n* reduce/reduce conflicts reported

124317  $n/350$  working sets used  
 124318 Memory: states, etc.  $n/15000$ , parser  $n/15000$   
 124319  $n/600$  distinct lookahead sets  
 124320  $n$  extra closures  
 124321  $n$  shift entries,  $n$  exceptions  
 124322  $n$  goto entries  
 124323  $n$  entries saved by goto default  
 124324 Optimizer space used: input  $n/15000$ , output  $n/15000$   
 124325  $n$  table entries,  $n$  zero  
 124326 Maximum spread:  $n$ , Maximum offset:  $n$

124327 The report of internal tables in the description file is left implementation-defined because all  
 124328 aspects of these limits are also implementation-defined. Some implementations may use  
 124329 dynamic allocation techniques and have no specific limit values to report.

124330 The format of the **y.output** file is not given because specification of the format was not seen to  
 124331 enhance applications portability. The listing is primarily intended to help human users  
 124332 understand and debug the parser; use of **y.output** by a conforming application script would be  
 124333 unusual. Furthermore, implementations have not produced consistent output and no popular  
 124334 format was apparent. The format selected by the implementation should be human-readable, in  
 124335 addition to the requirement that it be a text file.

124336 Standard error reports are not specifically described because they are seldom of use to  
 124337 conforming applications and there was no reason to restrict implementations.

124338 Some implementations recognize "`={"`" as equivalent to "`{`" because it appears in historical  
 124339 documentation. This construction was recognized and documented as obsolete as long ago as  
 124340 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of POSIX.1-2024 chose to  
 124341 leave it as obsolete and omit it.

124342 Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They  
 124343 should not be returned as multi-byte character literals. The token **error** that is used for error  
 124344 recovery is normally assigned the value 256 in the historical implementation. Thus, the token  
 124345 value 256, which is used in many multi-byte character sets, is not available for use as the value  
 124346 of a user-defined token.

124347 Earlier versions of this standard did not require the code file created by *yacc* to contain  
 124348 declarations of *yyerror()*, *yylex()*, and *yyparse()*. This meant that portable applications that did  
 124349 not define them had to declare them in the *grammar* file, to ensure they would not be diagnosed  
 124350 by the compiler as being called without being declared, but this was not stated in those versions  
 124351 of the standard either. The standard developers decided it was preferable for *yacc* to include the  
 124352 declarations in the code file and this is now a requirement. However, the declarations of  
 124353 *yyerror()* and *yylex()* are only visible if a macro of the same name is not defined, which provides  
 124354 application writers with a way to suppress the declaration if desired (for example, in order to  
 124355 provide their own declaration that would conflict with the one written by *yacc*). These functions  
 124356 are not declared in the header file because a macro definition in the declaration section would  
 124357 not be able to suppress them there.

124358 Earlier versions of this standard were also silent about a declaration of *main()*. However, the  
 124359 equivalent solution was not adopted because a declaration of *main()* would only be needed if it  
 124360 is called recursively by an application. Although in theory an application could call the *yacc*  
 124361 library version of *main()* from code in a *grammar* file, it is questionable why any application  
 124362 (other than a test suite) would do so, in particular because that version of *main()* does not accept  
 124363 any arguments and it calls *exit()*—it does not return—and therefore is of little use recursively.  
 124364 An application that includes its own definition of *main()* could call it recursively, but can

124365 reasonably be expected to ensure it does not call *main()* without previously defining or declaring  
 124366 it. An additional complication is that *main()* has multiple different allowed prototypes. The  
 124367 standard developers decided the simplest solution was to disallow *yacc* from providing a  
 124368 declaration of *main()* in the code file.

#### 124369 **FUTURE DIRECTIONS**

124370 If this utility is directed to create a new directory entry that contains any bytes that have the  
 124371 encoded value of a <newline> character, implementations are encouraged to treat this as an  
 124372 error. A future version of this standard may require implementations to treat this as an error.

#### 124373 **SEE ALSO**

124374 *c17, lex*

124375 XBD [Chapter 8](#) (on page 167), [Section 12.2](#) (on page 215)

#### 124376 **CHANGE HISTORY**

124377 First released in Issue 2.

#### 124378 **Issue 5**

124379 The FUTURE DIRECTIONS section is added.

#### 124380 **Issue 6**

124381 This utility is marked as part of the C-Language Development Utilities option.

124382 Minor changes have been added to align with the IEEE P1003.2b draft standard.

124383 The normative text is reworded to avoid use of the term “must” for application requirements.

124384 IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the }%  
 124385 token to the %}.

#### 124386 **Issue 7**

124387 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for  
 124388 generated code to conform to the ISO C standard.

124389 Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language  
 124390 trigraphs and curly brace preprocessing tokens.

124391 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
 124392 apply.

124393 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

124394 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0204 [977] is applied.

#### 124395 **Issue 8**

124396 Austin Group Defect 251 is applied, encouraging implementations to disallow the creation of  
 124397 filenames containing any bytes that have the encoded value of a <newline> character.

124398 Austin Group Defect 1122 is applied, changing the description of *NLSPATH*.

124399 Austin Group Defect 1269 is applied, changing the required contents of the code file (including  
 124400 **#define** statements for **YYEMPTY** and **YYEOF**) and adding new requirements for the interface to  
 124401 the lexical analyzer.

124402 Austin Group Defect 1388 is applied, changing the requirements relating to declarations of  
 124403 *yyerror()*, *yylex()*, *yyparse()*, and *main()*.

124404 **NAME**

124405 zcat — expand and concatenate data

124406 **SYNOPSIS**124407 XSI `zcat [file...]`124408 **DESCRIPTION**124409 Refer to *compress*.





124410

 *The Open Group Standard*

124411

**Vol. 4:**

124412

**Rationale (Informative), Issue 8**

124413


*The Open Group*

124414

*The Institute of Electrical and Electronics Engineers, Inc.*



124415

 *Rationale (Informative)*

124416

**Part A:**

124417

**Base Definitions**

124418

*The Open Group*

124419

*The Institute of Electrical and Electronics Engineers, Inc.*



# Rationale for Base Definitions

## 124422 A.1 Introduction

### 124423 A.1.1 Scope

124424 POSIX.1-2024 is one of a family of standards known as POSIX. The family of standards extends  
124425 to many topics; POSIX.1 consists of both operating system interfaces and shell and utilities.  
124426 POSIX.1-2024 is technically identical to The Open Group Base Specifications, Issue 8.

#### 124427 Scope of POSIX.1-2024

124428 The (paraphrased) goals of this development were to revise the single document that is ISO/IEC  
124429 9945:2009 Parts 1 through 4 as amended by ISO/IEC 9945:2009/Cor.1:2013 and ISO/IEC  
124430 9945:2009/Cor.2:2017, IEEE Std 1003.1-2017, and the appropriate parts of The Open Group  
124431 Single UNIX Specification, Version 5. This work has been undertaken by the Austin Group, a  
124432 joint working group of IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.

124433 The following are the base documents in this version:

- 124434 • IEEE Std 1003.1-2017
- 124435 • IEEE Std 1003.26-2003
- 124436 • ISO/IEC 9899: 2018, Programming Languages — C

124437 This version has addressed the following areas:

- 124438 • Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1,  
124439 and ISO/IEC defect reports against ISO/IEC 9945

124440 The repository of interpretations can be accessed at [www.opengroup.org/austin/interps](http://www.opengroup.org/austin/interps).

- 124441 • Issues raised in corrigenda for The Open Group Standards and working group resolutions  
124442 from The Open Group

- 124443 • Changes to make the text self-consistent with the additional material merged

124444 A list of the new interfaces is included in [Section B.1.1](#) (on page 3731).

- 124445 • Features, marked obsolescent in the base documents, have been considered for removal in  
124446 this version

124447 See [Section B.1.1](#) (on page 3731) and [Section C.1.1](#) (on page 3855).

- 124448 • Alignment with the ISO/IEC 9899: 2018 standard

124449 The following were requirements on POSIX.1-2024:

- 124450 • Backward-compatibility

124451 For interfaces carried forward, it was agreed that there should be no breakage of  
124452 functionality in the existing base documents. All strictly conforming applications will be

124453 conforming but not necessarily strictly conforming to the revised standard. The goal is for  
 124454 system implementations to be able to support the existing and revised standards  
 124455 simultaneously.

124456 • Architecture and *n*-bit-neutral

124457 The common standard should not make any implicit assumptions about the system  
 124458 architecture or size of data types; for example, previously some 32-bit implicit assumptions  
 124459 had crept into the standards.

124460 • Extensibility

124461 It should be possible to extend the common standard without breaking backwards-  
 124462 compatibility; for example, the name space should be reserved and structured to avoid  
 124463 duplication of names between the standard and extensions to it.

#### 124464 **POSIX.1 and the ISO C Standard**

124465 The standard developers believed it essential for a programmer to have a single complete  
 124466 reference place, but recognized that deference to the formal standard has to be addressed for the  
 124467 duplicate interface definitions between the ISO C standard and POSIX.1-2024.

124468 Where an interface has a version in the ISO C standard, the DESCRIPTION section describes the  
 124469 relationship to the ISO C standard and markings are included as appropriate to show where the  
 124470 ISO C standard has been extended in the text.

124471 A block of text is included at the start of each affected reference page stating whether the page is  
 124472 aligned with the ISO C standard or extended. Each page has been parsed for additions beyond  
 124473 the ISO C standard (that is, including both POSIX and UNIX extensions), and these extensions  
 124474 are marked as CX extensions (for C extensions).

#### 124475 **FIPS Requirements**

124476 The Federal Information Processing Standards (FIPS) are a series of US government  
 124477 procurement standards managed and maintained on behalf of the US Department of Commerce  
 124478 by the National Institute of Standards and Technology (NIST).

124479 The following restrictions were integrated into IEEE Std 1003.1-2001. They originally came from  
 124480 FIPS 151-2 which was withdrawn by NIST on February 25 2000.

124481 • The implementation supports `_POSIX_CHOWN_RESTRICTED`.

124482 • The limit `{NGROUPS_MAX}` is greater than or equal to 8.

124483 • The implementation supports the setting of the group ID of a file (when it is created) to  
 124484 that of the parent directory.

124485 • The implementation supports `_POSIX_SAVED_IDS`.

124486 • The implementation supports `_POSIX_VDISABLE`.

124487 • The implementation supports `_POSIX_JOB_CONTROL`.

124488 • The implementation supports `_POSIX_NO_TRUNC`.

124489 • The `read()` function returns the number of bytes read when interrupted by a signal and  
 124490 does not return `-1`.

124491 • The `write()` function returns the number of bytes written when interrupted by a signal and  
 124492 does not return `-1`.

- 124493 • In the environment for the login shell, the environment variables *LOGNAME* and *HOME*
- 124494 are defined and have the properties described in POSIX.1-2024.
- 124495 • The value of {CHILD\_MAX} is greater than or equal to 25.
- 124496 • The value of {OPEN\_MAX} is greater than or equal to 20.
- 124497 • The implementation supports the functionality associated with the symbols CS7, CS8,
- 124498 CSTOPB, PARODD, and PARENB defined in `<termios.h>`.

### 124499 **A.1.2 Word Usage**

124500 The content of this section is mandated by IEEE and consequently it cannot be combined with  
124501 the “Terminology” section.

124502 Note that where the footnotes state that “must” is used only to describe unavoidable situations  
124503 and “will” is only used in statements of fact, they are referring to uses of these words in  
124504 normative text. In informative text, they are used in other ways with their usual dictionary  
124505 meanings.

### 124506 **A.1.3 Conformance**

124507 See [Section A.2](#) (on page 3643).

### 124508 **A.1.4 Normative References**

124509 There is no additional rationale provided for this section.

### 124510 **A.1.5 Change History**

124511 For Issue 7 onwards, in references to Technical Corrigenda, the original Austin Group defect  
124512 report numbers that gave rise to the change are included in square brackets after the change  
124513 number from the Technical Corrigendum. For more information on Austin Group defect reports  
124514 see [www.opengroup.org/austin/defectform.html](http://www.opengroup.org/austin/defectform.html).

### 124515 **A.1.6 Terminology**

124516 The meanings specified in POSIX.1-2024 for the words *shall*, *should*, and *may* are mandated by  
124517 ISO/IEC directives.

124518 In the Rationale (Informative) volume of POSIX.1-2024, the words *shall*, *should*, and *may* are  
124519 sometimes used to illustrate similar usages in POSIX.1-2024. However, the rationale itself does  
124520 not specify anything regarding implementations or applications.

#### 124521 **conformance document**

124522 As a practical matter, the conformance document is effectively part of the system  
124523 documentation. Conformance documents are distinguished by POSIX.1-2024 so that they  
124524 can be referred to distinctly.

#### 124525 **implementation-defined**

124526 This definition is analogous to that of the ISO C standard and, together with “undefined”  
124527 and “unspecified”, provides a range of specification of freedom allowed to the interface  
124528 implementor.

- 124529       **may**
- 124530       The use of *may* has been limited as much as possible, due both to confusion stemming from
- 124531       its ordinary English meaning and to objections regarding the desirability of having as few
- 124532       options as possible and those as clearly specified as possible.
- 124533       The usage of *can* and *may* were selected to contrast optional application behavior (can)
- 124534       against optional implementation behavior (may).
- 124535       **shall**
- 124536       Declarative sentences are sometimes used in POSIX.1-2024 as if they included the word
- 124537       *shall*, and facilities thus specified are no less required. For example, the two statements:
- 124538           1. The *foo()* function shall return zero.
- 124539           2. The *foo()* function returns zero.
- 124540       are meant to be exactly equivalent.
- 124541       **should**
- 124542       In POSIX.1-2024, the word *should* does not usually apply to the implementation, but rather
- 124543       to the application. Thus, the important words regarding implementations are *shall*, which
- 124544       indicates requirements, and *may*, which indicates options.
- 124545       **obsolescent**
- 124546       The term “obsolescent” means “do not use this feature in new applications”. A feature
- 124547       noted as obsolescent is supported by all implementations, but may be removed in a future
- 124548       version; new applications should not use these features. The obsolescence concept is not an
- 124549       ideal solution, but was used as a method of increasing consensus: many more objections
- 124550       would be heard from the user community if some of these historical features were suddenly
- 124551       removed without the grace period obsolescence implies. The phrase “may be removed in a
- 124552       future version” implies that the result of that consideration might in fact keep those features
- 124553       indefinitely if the predominance of applications do not migrate away from them quickly.
- 124554       **legacy**
- 124555       The term “legacy” was included in earlier versions of this standard but is no longer used in
- 124556       the current version.
- 124557       **system documentation**
- 124558       The system documentation should normally describe the whole of the implementation,
- 124559       including any extensions provided by the implementation. Such documents normally
- 124560       contain information at least as detailed as the specifications in POSIX.1-2024. Few
- 124561       requirements are made on the system documentation, but the term is needed to avoid a
- 124562       dangling pointer where the conformance document is permitted to point to the system
- 124563       documentation.
- 124564       **undefined**
- 124565       See *implementation-defined*.
- 124566       **unspecified**
- 124567       See *implementation-defined*.
- 124568       The definitions for “unspecified” and “undefined” appear nearly identical at first
- 124569       examination, but are not. The term “unspecified” means that a conforming application may
- 124570       deal with the unspecified behavior, and it should not care what the outcome is. The term
- 124571       “undefined” says that a conforming application should not do it because no definition is
- 124572       provided for what it does (and implicitly it would care what the outcome was if it tried it).
- 124573       It is important to remember, however, that if the syntax permits the statement at all, it must
- 124574       have some outcome in a real implementation.



124575 Thus, the terms “undefined” and “unspecified” apply to the way the application should  
124576 think about the feature. In terms of the implementation, it is always “defined”—there is  
124577 always some result, even if it is an error. The implementation is free to choose the behavior  
124578 it prefers.

124579 This also implies that an implementation, or another standard, could specify or define the  
124580 result in a useful fashion. The terms apply to POSIX.1-2024 specifically.

124581 The term “implementation-defined” implies requirements for documentation that are not  
124582 required for “undefined” (or “unspecified”). Where there is no need for a conforming  
124583 program to know the definition, the term “undefined” is used, even though  
124584 “implementation-defined” could also have been used in this context. There could be a  
124585 fourth term, specifying “this standard does not say what this does; it is acceptable to define  
124586 it in an implementation, but it does not need to be documented”, and undefined would  
124587 then be used very rarely for the few things for which any definition is not useful. In  
124588 particular, implementation-defined is used where it is believed that certain classes of  
124589 application will need to know such details to determine whether the application can be  
124590 successfully ported to the implementation. Such applications are not always strictly  
124591 portable, but nevertheless are common and useful; often the requirements met by the  
124592 application cannot be met without dealing with the issues implied by “implementation-  
124593 defined”. In some places the text refers to facilities supplied by the implementation that are  
124594 outside the standard as implementation-supplied or implementation-provided. This is not  
124595 intended to imply a requirement for documentation. If it were, the term “implementation-  
124596 defined” would have been used.

124597 In many places POSIX.1-2024 is silent about the behavior of some possible construct. For  
124598 example, a variable may be defined for a specified range of values and behaviors are  
124599 described for those values; nothing is said about what happens if the variable has any other  
124600 value. That kind of silence can imply an error in the standard, but it may also imply that the  
124601 standard was intentionally silent and that any behavior is permitted. There is a natural  
124602 tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent.  
124603 Silence is intended to be equivalent to the term “unspecified”.

124604 Three terms used within POSIX.1-2024 overlap in meaning: “macro”, “symbolic name”, and  
124605 “symbolic constant”.

#### 124606 **macro**

124607 This usually describes a C preprocessor symbol, the result of the **#define** operator, with or  
124608 without an argument. It may also be used to describe similar mechanisms in editors and  
124609 text processors.

#### 124610 **symbolic name**

124611 In earlier versions of this standard this was also sometimes used to refer to a C preprocessor  
124612 symbol (without arguments), but the intention is for all such uses to have been removed. It  
124613 is now mainly used to refer to the names for characters in character sets, but is sometimes  
124614 used to refer to host names and even filenames.

#### 124615 **symbolic constant**

124616 This also refers to a C preprocessor symbol, with specific associated requirements. See the  
124617 definition in [Section 3.363](#) (on page 84).

124618 **A.1.7 Definitions and Concepts**

124619 There is no additional rationale provided for this section.

124620 **A.1.8 Portability**

124621 To aid the identification of options within POSIX.1-2024, a notation consisting of margin codes  
 124622 and shading is used. This is based on the notation used in earlier versions of The Open Group  
 124623 Base specifications.

124624 The benefit of this approach is a reduction in the number of *if* statements within the running  
 124625 text, that makes the text easier to read, and also an identification to the programmer that they  
 124626 need to ensure that their target platforms support the underlying options. For example, if  
 124627 functionality is marked with RPP in the margin, it will be available on all systems supporting  
 124628 the Robust Mutex Priority Protection option, but may not be available on some others.

124629 *A.1.8.1 Codes*

124630 This section includes codes for options defined in XBD [Section 2.1.6](#) (on page 25), and the  
 124631 following additional codes for other purposes:

124632 **CX** This margin code is used to denote extensions beyond and, in exceptional cases,  
 124633 deviations from the ISO C standard. For interfaces that are duplicated between  
 124634 POSIX.1-2024 and the ISO C standard, a CX introduction block describes the nature of  
 124635 the duplication, with any extensions or deviations appropriately CX marked and  
 124636 shaded. Where deviations exist, the reasons for them are explained in the RATIONALE  
 124637 section of the affected interface. Deviations have become necessary because there is no  
 124638 longer any formal way for ISO to acknowledge defects in the ISO C standard. For the  
 124639 original C90 standard and the C99 revision, defect reports (DRs) were issued, but there  
 124640 is no equivalent mechanism for the current revision. Even if the defect is corrected in a  
 124641 later revision, without stating deviations POSIX.1-2024 would continue to require the  
 124642 incorrect behavior described in the version of the ISO C standard that it references.

124643 Where an interface is added to an ISO C standard header, within the header the  
 124644 interface has an appropriate margin marker and shading (for example, CX, XSI, TSE,  
 124645 and so on) and the same marking appears on the reference page in the SYNOPSIS  
 124646 section. This enables a programmer to easily identify that the interface is extending an  
 124647 ISO C standard header.

124648 Austin Group Defect 1755 is applied, changing the CX code description to include  
 124649 intentional conflicts (deviations).

124650 **MX and MXX**

124651 These two margin codes both relate to the IEC 60559 Floating-Point option. The MX  
 124652 code denotes functionality that is mandated by the ISO C standard for IEC 60559  
 124653 implementations; the MXX code denotes IEC 60559 functionality that is an extension to  
 124654 the ISO C standard.

124655 **MXC** This margin code is used to denote functionality related to the IEC 60559 Complex  
 124656 Floating-Point option.

124657 **OB** This margin code is used to denote obsolescent behavior and thus flag a possible future  
 124658 applications portability warning.

124659 OH The Single UNIX Specification has historically tried to reduce the number of headers an  
 124660 application has had to include when using a particular interface. Sometimes this was  
 124661 fewer than the base standard, and hence a notation is used to flag which headers are  
 124662 optional if you are using a system supporting the XSI option.

124663 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0001 [591] is applied.

#### 124664 A.1.8.2 Margin Code Notation

124665 Since some features may depend on one or more options, or require more than one option, a  
 124666 notation is used. Where a feature requires support of a single option, a single margin code will  
 124667 occur in the margin. If it depends on two options and both are required, then the codes will  
 124668 appear with a <space> separator. If either of two options are required, then a logical OR is  
 124669 denoted using the ' | ' symbol. If more than two codes are used, a special notation is used.

## 124670 A.2 Conformance

124671 The terms “profile” and “profiling” are used throughout this section.

124672 A profile of a standard or standards is a codified set of option selections, such that by being  
 124673 conformant to a profile, particular classes of users are specifically supported.

### 124674 A.2.1 Implementation Conformance

124675 These definitions allow application developers to know what to depend on in an  
 124676 implementation.

124677 There is no definition of a “strictly conforming implementation”; that would be an  
 124678 implementation that provides *only* those facilities specified by POSIX.1 with no extensions  
 124679 whatsoever. This is because no actual operating system implementation can exist without  
 124680 system administration and initialization facilities that are beyond the scope of POSIX.1.

#### 124681 A.2.1.1 Requirements

124682 The word “support” is used in certain instances, rather than “provide”, in order to allow an  
 124683 implementation that has no resident software development facilities, but that supports the  
 124684 execution of a *Strictly Conforming POSIX.1 Application*, to be a *conforming implementation*.

124685 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0002 [810] is applied.

#### 124686 A.2.1.2 Documentation

124687 The conformance documentation is required to use the same numbering scheme as POSIX.1 for  
 124688 purposes of cross-referencing. All options that an implementation chooses are reflected in  
 124689 <limits.h> and <unistd.h>.

124690 Note that the use of “may” in terms of where conformance documents record where  
 124691 implementations may vary, implies that it is not required to describe those features identified as  
 124692 undefined or unspecified.

124693 Other aspects of systems must be evaluated by purchasers for suitability. Many systems  
 124694 incorporate buffering facilities, maintaining updated data in volatile storage and transferring  
 124695 such updates to non-volatile storage asynchronously. Various exception conditions, such as a  
 124696 power failure or a system crash, can cause this data to be lost. The data may be associated with a

124697 file that is still open, with one that has been closed, with a directory, or with any other internal  
124698 system data structures associated with permanent storage. This data can be lost, in whole or  
124699 part, so that only careful inspection of file contents could determine that an update did not  
124700 occur.

124701 Also, interrelated file activities, where multiple files and/or directories are updated, or where  
124702 space is allocated or released in the file system structures, can leave inconsistencies in the  
124703 relationship between data in the various files and directories, or in the file system itself. Such  
124704 inconsistencies can break applications that expect updates to occur in a specific sequence, so that  
124705 updates in one place correspond with related updates in another place.

124706 For example, if a user creates a file, places information in the file, and then records this action in  
124707 another file, a system or power failure at this point followed by restart may result in a state in  
124708 which the record of the action is permanently recorded, but the file created (or some of its  
124709 information) has been lost. The consequences of this to the user may be undesirable. For a user  
124710 on such a system, the only safe action may be to require the system administrator to have a  
124711 policy that requires, after any system or power failure, that the entire file system must be  
124712 restored from the most recent backup copy (causing all intervening work to be lost).

124713 The characteristics of each implementation will vary in this respect and may or may not meet  
124714 the requirements of a given application or user. Enforcement of such requirements is beyond the  
124715 scope of POSIX.1. It is up to the purchaser to determine what facilities are provided in an  
124716 implementation that affect the exposure to possible data or sequence loss, and also what  
124717 underlying implementation techniques and/or facilities are provided that reduce or limit such  
124718 loss or its consequences.

#### 124719 A.2.1.3 *POSIX Conformance*

124720 This really means conformance to the base standard; however, since this document includes the  
124721 core material of the Single UNIX Specification, the standard developers decided that it was  
124722 appropriate to segment the conformance requirements into two, the former for the base  
124723 standard, and the latter for the Single UNIX Specification (denoted XSI Conformance).

124724 Within POSIX.1 there are some symbolic constants that, if defined to a certain value or range of  
124725 values, indicate that a certain option is enabled. Other symbolic constants exist in POSIX.1 for  
124726 other reasons.

124727 In this version, some features that were previously optional have been made mandatory. For  
124728 backwards compatibility, the symbolic constants associated with the option are still required  
124729 now with fixed allowable ranges or values. The following options from previous versions of this  
124730 standard are now mandatory:

124731            \_POSIX\_ASYNCHRONOUS\_IO  
 124732            \_POSIX\_BARRIERS  
 124733            \_POSIX\_CLOCK\_SELECTION  
 124734            \_POSIX\_MAPPED\_FILES  
 124735            \_POSIX\_MEMORY\_PROTECTION  
 124736            \_POSIX\_MONOTONIC\_CLOCK  
 124737            \_POSIX\_READER\_WRITER\_LOCKS  
 124738            \_POSIX\_REALTIME\_SIGNALS  
 124739            \_POSIX\_SEMAPHORES  
 124740            \_POSIX\_SPIN\_LOCKS  
 124741            \_POSIX\_THREAD\_SAFE\_FUNCTIONS  
 124742            \_POSIX\_THREADS  
 124743            \_POSIX\_TIMEOUTS  
 124744            \_POSIX\_TIMERS

124745            A POSIX-conformant system may support the XSI option required by the Single UNIX  
 124746            Specification. This was intentional since the standard developers intend them to be upwards-  
 124747            compatible, so that a system conforming to the Single UNIX Specification can also conform to  
 124748            the base standard at the same time.

124749            POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0003 [637] is applied.

124750            Austin Group Defect 729 is applied, adding \_POSIX\_DEVICE\_CONTROL.

124751            Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

#### 124752 A.2.1.4 XSI Conformance

124753            This section is included to describe the conformance requirements for the base volumes of the  
 124754            Single UNIX Specification.

124755            XSI conformance can be thought of as a profile, selecting certain options from POSIX.1-2024.

#### 124756 A.2.1.5 Option Groups

124757            The concept of “Option Groups” is included to allow collections of related functions or options  
 124758            to be grouped together. This has been used as follows: the “XSI Option Groups” have been  
 124759            created to allow super-options, collections of underlying options and related functions, to be  
 124760            collectively supported by XSI-conforming systems.

124761            The standard developers considered the matter of subprofiling and decided it was better to  
 124762            include an enabling mechanism rather than detailed normative requirements. A set of  
 124763            subprofiling options was developed and included later in this volume of POSIX.1-2024 as an  
 124764            informative illustration.

#### 124765 Subprofiling Considerations

124766            The goal of not simultaneously fixing maximums and minimums was to allow implementations  
 124767            of the base standard or standards to support multiple profiles without conflict.

124768 The following summarizes the rules for the limit types:

| Limit Type       | Fixed Value                   | Minimum Acceptable Value                   | Maximum Acceptable Value                   |
|------------------|-------------------------------|--------------------------------------------|--------------------------------------------|
| Standard Profile | Xs<br>Xp == Xs<br>(No change) | Ys<br>Yp >= Ys<br>(May increase the limit) | Zs<br>Zp <= Zs<br>(May decrease the limit) |

124774 The intent is that ranges specified by limits in profiles be entirely contained within the  
 124775 corresponding ranges of the base standard or standards being profiled, and that the unlimited  
 124776 end of a range in a base standard must remain unlimited in any profile of that standard.

124777 Thus, the fixed `_POSIX_*` limits are constants and must not be changed by a profile. The variable  
 124778 counterparts (typically without the leading `_POSIX_`) can be changed but still remain  
 124779 semantically the same; that is, they still allow implementation values to vary as long as they  
 124780 meet the requirements for that value (be it a minimum or maximum).

124781 Where a profile does not provide a feature upon which a limit is based, the limit is not relevant.  
 124782 Applications written to that profile should be written to operate independently of the value of  
 124783 the limit.

124784 An example which has previously allowed implementations to support both the base standard  
 124785 and two other profiles in a compatible manner follows:

```
124786 Base standard (POSIX.1-1996): _POSIX_CHILD_MAX 6
124787 Base standard: CHILD_MAX minimum maximum _POSIX_CHILD_MAX
124788 FIPS profile/SUSv2 CHILD_MAX 25 (minimum maximum)
```

124789 Another example:

```
124790 Base standard (POSIX.1-1996): _POSIX_NGROUPS_MAX 0
124791 Base standard: NGROUPS_MAX minimum maximum _POSIX_NGROUP_MAX
124792 FIPS profile/SUSv2 NGROUPS_MAX 8
```

124793 A profile may lower a minimum maximum below the equivalent `_POSIX` value:

```
124794 Base standard: _POSIX_foo_MAX Z
124795 Base standard: foo_MAX _POSIX_foo_MAX
124796 profile standard : foo_MAX X (X can be less than, equal to,
124797 or greater than _POSIX_foo_MAX)
```

124798 In this case an implementation conforming to the profile may not conform to the base standard,  
 124799 but an implementation to the base standard will conform to the profile.

124800 **XSI Option Groups**

124801 Austin Group Defect 1192 is applied, marking the `encrypt()` and `setkey()` functions as  
 124802 obsolescent.

124803 Austin Group Defect 1346 is applied, removing `_POSIX_MONOTONIC_CLOCK` from the  
 124804 Advanced Realtime option group.

124805 *A.2.1.6 Options*

124806 The final subsections within *Implementation Conformance* list the core options within  
 124807 POSIX.1-2024. This includes both options for the System Interfaces volume of POSIX.1-2024 and  
 124808 the Shell and Utilities volume of POSIX.1-2024.

124809 Austin Group Defect 190 is applied, adding `man` to the list of utilities in the User Portability

124810 Utilities option.

## 124811 A.2.2 Application Conformance

124812 These definitions guide users or adapters of applications in determining on which  
124813 implementations an application will run and how much adaptation would be required to make  
124814 it run on others. These definitions are modeled after related ones in the ISO C standard.

124815 POSIX.1 occasionally uses the expressions “portable application” or “conforming application”.  
124816 As they are used, these are synonyms for any of these terms. The differences between the classes  
124817 of application conformance relate to the requirements for other standards, the options supported  
124818 (such as the XSI option) or, in the case of the Conforming POSIX.1 Application Using Extensions,  
124819 to implementation extensions. When one of the less explicit expressions is used, it should be  
124820 apparent from the context of the discussion which of the more explicit names is appropriate

### 124821 A.2.2.1 Strictly Conforming POSIX Application

124822 This definition is analogous to that of an ISO C standard “conforming program”.

124823 The major difference between a Strictly Conforming POSIX Application and an ISO C standard  
124824 strictly conforming program is that the latter is not allowed to use features of POSIX that are not  
124825 in the ISO C standard.

### 124826 A.2.2.2 Conforming POSIX Application

124827 Examples of <National Bodies> include ANSI, BSI, and AFNOR.

### 124828 A.2.2.3 Conforming POSIX Application Using Extensions

124829 Due to possible requirements for configuration or implementation characteristics in excess of the  
124830 specifications in <limits.h> or related to the hardware (such as array size or file space), not  
124831 every Conforming POSIX Application Using Extensions will run on every conforming  
124832 implementation.

### 124833 A.2.2.4 Strictly Conforming XSI Application

124834 This is intended to be upwards-compatible with the definition of a Strictly Conforming POSIX  
124835 Application, with the addition of the facilities and functionality included in the XSI option.

### 124836 A.2.2.5 Conforming XSI Application Using Extensions

124837 Such applications may use extensions beyond the facilities defined by POSIX.1-2024 including  
124838 the XSI option, but need to document the additional requirements.

### 124839 **A.2.3 Language-Dependent Services for the C Programming Language**

124840 POSIX.1 is, for historical reasons, both a specification of an operating system interface, shell and  
 124841 utilities, and a C binding for that specification. Efforts had been previously undertaken to  
 124842 generate a language-independent specification; however, that had failed, and the fact that the  
 124843 ISO C standard is the *de facto* primary language on POSIX and the UNIX system makes this a  
 124844 necessary and workable situation.

### 124845 **A.2.4 Other Language-Related Specifications**

124846 There is no additional rationale provided for this section.

## 124847 **A.3 Definitions**

124848 The definitions in this section are stated so that they can be used as exact substitutes for the  
 124849 terms in text. They should not contain requirements or cross-references to sections within  
 124850 POSIX.1-2024; that is accomplished by using an informative note. In addition, the term should  
 124851 not be included in its own definition. Where requirements or descriptions need to be addressed  
 124852 but cannot be included in the definitions, due to not meeting the above criteria, these occur in  
 124853 the General Concepts chapter.

124854 In this version, the definitions have been reworked extensively to meet style requirements and to  
 124855 include terms from the base documents (see the Scope).

124856 Many of these definitions are necessarily circular, and some of the terms (such as “process”) are  
 124857 variants of basic computing science terms that are inherently hard to define. Where some  
 124858 definitions are more conceptual and contain requirements, these appear in the General Concepts  
 124859 chapter. Those listed in this section appear in an alphabetical glossary format of terms.

124860 Some definitions must allow extension to cover terms or facilities that are not explicitly  
 124861 mentioned in POSIX.1-2024. For example, the definition of “Extended Security Controls”  
 124862 permits implementations beyond those defined in POSIX.1-2024.

124863 Some terms in the following list of notes do not appear in POSIX.1-2024; these are marked  
 124864 suffixed with an asterisk (\*). Many of them have been specifically excluded from POSIX.1-2024  
 124865 because they concern system administration, implementation, or other issues that are not  
 124866 specific to the programming interface. Those are marked with a reason, such as  
 124867 “implementation-defined”.

#### 124868 **Alias Name**

124869 Austin Group Defect 1050 is applied, adding '-' to the characters that can be used in an alias  
 124870 name.

#### 124871 **Anonymous Memory Object**

124872 Austin Group Defect 850 is applied, adding anonymous memory objects.



124873 **Application**

124874 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0004 [937] is applied.

124875 **Appropriate Privileges**

124876 One of the fundamental security problems with many historical UNIX systems has been that the  
 124877 privilege mechanism is monolithic—a user has either no privileges or *all* privileges. Thus, a  
 124878 successful “trojan horse” attack on a privileged process defeats all security provisions.  
 124879 Therefore, POSIX.1 allows more granular privilege mechanisms to be defined. For many  
 124880 historical implementations of the UNIX system, the presence of the term “appropriate  
 124881 privileges” in POSIX.1 may be understood as a synonym for “superuser” (UID 0). However,  
 124882 other systems have emerged where this is not the case and each discrete controllable action has  
 124883 *appropriate privileges* associated with it. Because this mechanism is implementation-defined, it  
 124884 must be described in the conformance document. Although that description affects several parts  
 124885 of POSIX.1 where the term “appropriate privilege” is used, because the term “implementation-  
 124886 defined” only appears here, the description of the entire mechanism and its effects on these  
 124887 other sections belongs in this equivalent section of the conformance document. This is especially  
 124888 convenient for implementations with a single mechanism that applies in all areas, since it only  
 124889 needs to be described once.

124890 **Async-Signal-Safe Function**

124891 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0005 [516] is applied.

124892 **Background Job**

124893 Austin Group Defect 1254 is applied, changing this definition.

124894 **Base Character\***

124895 The term “Base Character” has been removed, as it was felt that the use of this term within  
 124896 POSIX.1-2024 was common usage English.

124897 **Basename**

124898 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0006 [653] is applied.

124899 **Built-In Utility**

124900 Austin Group Defect 854 is applied, changing text relating to regular built-in utilities.

124901 **Byte**

124902 The restriction that a byte is now exactly eight bits was a conscious decision by the standard  
 124903 developers. It came about due to a combination of factors, primarily the use of the type **int8\_t**  
 124904 within the networking functions and the alignment with the ISO/IEC 9899:1999 standard,  
 124905 where the **intN\_t** types were first defined.

124906 According to the ISO/IEC 9899:1999 standard:

- 124907 • The **[u]intN\_t** types must be two’s complement with no padding bits and no illegal values.
- 124908 • All types (apart from bit fields, which are not relevant here) must occupy an integral  
 124909 number of bytes.
- 124910 • If a type with width  $W$  occupies  $B$  bytes with  $C$  bits per byte ( $C$  is the value of  
 124911  $\{\text{CHAR\_BIT}\}$ ), then it has  $P$  padding bits where  $P+W=B*C$ .

124912 • Therefore, for `int8_t`  $P=0$ ,  $W=8$ . Since  $B \geq 1$ ,  $C \geq 8$ , the only solution is  $B=1$ ,  $C=8$ .

124913 The standard developers also felt that this was not an undue restriction for the current state-of-  
124914 the-art for this version of the standard, but recognize that if industry trends continue, a wider  
124915 character type may be required in the future.

#### 124916 **Character**

124917 The term “character” is used to mean a sequence of one or more bytes representing a member of  
124918 a character set. The deviation in the exact text of the ISO C standard definition for “byte” meets  
124919 the intent of the rationale of the ISO C standard also clears up the ambiguity raised by the term  
124920 “basic execution character set”. The octet-minimum requirement is a reflection of the  
124921 {CHAR\_BIT} value.

124922 Austin Group Defect 1356 is applied, changing the definition of “character” to match the  
124923 definition of the term “multi-byte character” in the ISO C standard.

#### 124924 **Child Process**

124925 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/3 is applied, adding the `vfork()` function  
124926 to those listed.

#### 124927 **Clock Tick**

124928 The ISO C standard defines a similar interval for use by the `clock()` function. There is no  
124929 requirement that these intervals be the same. In historical implementations these intervals are  
124930 different.

#### 124931 **Code Block**

124932 Austin Group Defect 613 is applied, adding this definition.

#### 124933 **Command**

124934 The terms “command” and “utility” are related but have distinct meanings. Command is  
124935 defined as “a directive to a shell to perform a specific task”. The directive can be in the form of a  
124936 single utility name (for example, `ls`), or the directive can take the form of a compound command  
124937 (for example, `"ls | grep name | pr"`). A utility is a program that can be called by name  
124938 from a shell. Issuing only the name of the utility to a shell is the equivalent of a one-word  
124939 command. A utility may be invoked as a separate program that executes in a different process  
124940 than the command language interpreter, or it may be implemented as a part of the command  
124941 language interpreter. For example, the `echo` command (the directive to perform a specific task)  
124942 may be implemented such that the `echo` utility (the logic that performs the task of echoing) is in a  
124943 separate program; therefore, it is executed in a process that is different from the command  
124944 language interpreter. Conversely, the logic that performs the `echo` utility could be built into the  
124945 command language interpreter; therefore, it could execute in the same process as the command  
124946 language interpreter.

124947 The terms “tool” and “application” can be thought of as being synonymous with “utility” from  
124948 the perspective of the operating system kernel. Tools, applications, and utilities historically have  
124949 run, typically, in processes above the kernel level. Tools and utilities historically have been a part  
124950 of the operating system non-kernel code and have performed system-related functions, such as  
124951 listing directory contents, checking file systems, repairing file systems, or extracting system  
124952 status information. Applications have not generally been a part of the operating system, and  
124953 they perform non-system-related functions, such as word processing, architectural design,  
124954 mechanical design, workstation publishing, or financial analysis. Utilities have most frequently  
124955 been provided by the operating system distributor, applications by third-party software

124956 distributors, or by the users themselves. Nevertheless, POSIX.1-2024 does not differentiate  
 124957 between tools, utilities, and applications when it comes to receiving services from the system, a  
 124958 shell, or the standard utilities. (For example, the *xargs* utility invokes another utility; it would be  
 124959 of fairly limited usefulness if the users could not run their own applications in place of the  
 124960 standard utilities.) Utilities are not applications in the sense that they are not themselves subject  
 124961 to the restrictions of POSIX.1-2024 or any other standard—there is no requirement for *grep*, *stty*,  
 124962 or any of the utilities defined here to be any of the classes of conforming applications.

### 124963 **Column Positions**

124964 In most 1-byte character sets, such as ASCII, the concept of column positions is identical to  
 124965 character positions and to bytes. Therefore, it has been historically acceptable for some  
 124966 implementations to describe line folding or tab stops or table column alignment in terms of  
 124967 bytes or character positions. Other character sets pose complications, as they can have internal  
 124968 representations longer than one octet and they can have display characters that have different  
 124969 widths on the terminal screen or printer.

124970 In POSIX.1-2024 the term “column positions” has been defined to mean character—not byte—  
 124971 positions in input files. Output files describe the column position in terms of the display width  
 124972 of the narrowest printable character in the character set, adjusted to fit the characteristics of the  
 124973 output device. It is very possible that *n* column positions will not be able to hold *n* characters in  
 124974 some character sets, unless all of those characters are of the narrowest width. It is assumed that  
 124975 the implementation is aware of the width of the various characters, deriving this information  
 124976 from the value of *LC\_CTYPE*, and thus can determine how many column positions to allot for  
 124977 each character in those utilities where it is important.

124978 The term “column position” was used instead of the more natural “column” because the latter is  
 124979 frequently used in the different contexts of columns of figures, columns of table values, and so  
 124980 on. Wherever confusion might result, these latter types of columns are referred to as “text  
 124981 columns”.

### 124982 **Condition Variable**

124983 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

### 124984 **Control Operator**

124985 Austin Group Defect 449 is applied, adding ; & to the list of control operators.

### 124986 **Controlling Terminal**

124987 The question of which of possibly several special files referring to the terminal is meant is not  
 124988 addressed in POSIX.1. The pathname */dev/tty* is a synonym for the controlling terminal  
 124989 associated with a process.

### 124990 **Core Image**

124991 Austin Group Defect 1141 is applied, replacing the core file definition with a core image  
 124992 definition.

- 124993           **CPU Time (Execution Time)**
- 124994           Austin Group Defect 1116 is applied, removing a reference to the Threads option that existed in  
124995           earlier versions of this standard.
- 124996           **Decimal-Point Character**
- 124997           Austin Group Defect 1449 is applied, adding this definition.
- 124998           **Declaration Utility**
- 124999           Austin Group Defect 351 is applied, adding this definition.
- 125000           **Device Number\***
- 125001           The concept is handled in *stat()* as *ID of device*.
- 125002           **Direct I/O**
- 125003           Historically, direct I/O refers to the system bypassing intermediate buffering, but may be  
125004           extended to cover implementation-defined optimizations.
- 125005           **Directory**
- 125006           The format of the directory file is implementation-defined and differs radically between  
125007           System V and 4.3 BSD. However, routines (derived from 4.3 BSD) for accessing directories and  
125008           certain constraints on the format of the information returned by those routines are described in  
125009           the **<dirent.h>** header.
- 125010           **Directory Entry (or Hard Link)**
- 125011           Austin Group Defect 1380 is applied, changing “link” to “hard link”.
- 125012           **Display**
- 125013           The Shell and Utilities volume of POSIX.1-2024 assigns precise requirements for the terms  
125014           “display” and “write”. Some historical systems have chosen to implement certain utilities  
125015           without using the traditional file descriptor model. For example, the *vi* editor might employ  
125016           direct screen memory updates on a personal computer, rather than a *write()* system call. An  
125017           instance of user prompting might appear in a dialog box, rather than with standard error. When  
125018           the Shell and Utilities volume of POSIX.1-2024 uses the term “display”, the method of  
125019           outputting to the terminal is unspecified; many historical implementations use *termcap* or  
125020           *terminfo*, but this is not a requirement. The term “write” is used when the Shell and Utilities  
125021           volume of POSIX.1-2024 mandates that a file descriptor be used and that the output can be  
125022           redirected. However, it is assumed that when the writing is directly to the terminal (it has not  
125023           been redirected elsewhere), there is no practical way for a user or test suite to determine whether  
125024           a file descriptor is being used. Therefore, the use of a file descriptor is mandated only for the  
125025           redirection case and the implementation is free to use any method when the output is not  
125026           redirected. The verb *write* is used almost exclusively, with the very few exceptions of those  
125027           utilities where output redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.

- 125028           **Dot**
- 125029           The symbolic name *dot* is carefully used in POSIX.1 to distinguish the working directory  
125030           filename from a period or a decimal point.
- 125031           **Dot-Dot**
- 125032           Historical implementations permit the use of these filenames without their special meanings.  
125033           Such use precludes any meaningful use of these filenames by a Conforming POSIX.1  
125034           Application. Therefore, such use is considered an extension, the use of which makes an  
125035           implementation non-conforming; see also [Section A.4.16](#) (on page 3683).
- 125036           **Dot-Po File**
- 125037           Austin Group Defect 1122 is applied, adding this definition.
- 125038           **Empty Directory**
- 125039           Austin Group Defect 1380 is applied, changing “link” to “hard link”.
- 125040           **Epoch**
- 125041           Historically, the origin of UNIX system time was referred to as “00:00:00 GMT, January 1, 1970”.  
125042           Greenwich Mean Time is actually not a term acknowledged by the international standards  
125043           community; therefore, this term, “Epoch”, is used to abbreviate the reference to the actual  
125044           standard, Coordinated Universal Time.
- 125045           **FIFO Special File**
- 125046           See [Pipe](#) (on page 3663).
- 125047           **File**
- 125048           It is permissible for an implementation-defined file type to be non-readable or non-writable.
- 125049           **File Classes**
- 125050           These classes correspond to the historical sets of permission bits. The classes are general to  
125051           allow implementations flexibility in expanding the access mechanism for more stringent security  
125052           environments. Note that a process is in one and only one class, so there is no ambiguity.
- 125053           **File Descriptor**
- 125054           Austin Group Defect 1493 is applied, moving some information from XCU [Section 2.7](#) (on page  
125055           2493) to this definition.
- 125056           **File Lock**
- 125057           Austin Group Defect 768 is applied, changing this definition.

- 125058           **File Mode**
- 125059           POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0007 [834] is applied.
- 125060           **Filename**
- 125061           Filenames are sequences of bytes, not sequences of characters. The only bytes that this standard  
125062           says cannot appear in any filename are the slash byte and the null byte. This is a side-effect of  
125063           the fact that no conforming implementations of the standard currently provide a way to pass  
125064           information specifying the locale associated with strings passed between user-level applications  
125065           and the kernel. This decision could be revisited if implementations develop a way to associate a  
125066           locale with the strings passed between kernel space and user space.
- 125067           Implementations may add other restrictions to the byte sequences allowed in filenames except  
125068           that any filename consisting of no more than {NAME\_MAX} bytes from the set of characters in  
125069           the portable filename character set must be allowed.
- 125070           See [Pathname](#) (on page 3663).
- 125071           **File System**
- 125072           Historically, the meaning of this term has been overloaded with two meanings: that of the  
125073           complete file hierarchy, and that of a mountable subset of that hierarchy; that is, a mounted file  
125074           system. POSIX.1 uses the term “file system” in the second sense, except that it is limited to the  
125075           scope of a process (and root directory of a process). This usage also clarifies the domain in which  
125076           a file serial number is unique.
- 125077           **Foreground Job**
- 125078           Austin Group Defect 1254 is applied, changing this definition.
- 125079           **Graphic Character**
- 125080           This definition is made available for those definitions (in particular, *TZ*) which must exclude  
125081           control characters.
- 125082           **Group Database**
- 125083           IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/4 is applied, removing the words “of  
125084           implementation-defined format”. See [User Database](#) (on page 3675).
- 125085           **Group File\***
- 125086           Implementation-defined; see [User Database](#) (on page 3675).
- 125087           **Group ID**
- 125088           POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0008 [511] is applied.
- 125089           **Group Name**
- 125090           POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0009 [584] is applied.

- 125091           **Hard Link**
- 125092           Austin Group Defect 1380 is applied, changing this definition.
- 125093           **Historical Implementations\***
- 125094           This refers to previously existing implementations of programming interfaces and operating  
125095           systems that are related to the interface specified by POSIX.1.
- 125096           **Hole**
- 125097           Austin Group Defect 415 is applied, adding this definition.
- 125098           **Hosted Implementation\***
- 125099           This refers to a POSIX.1 implementation that is accomplished through interfaces from the  
125100           POSIX.1 services to some alternate form of operating system kernel services. Note that the line  
125101           between a hosted implementation and a native implementation is blurred, since most  
125102           implementations will provide some services directly from the kernel and others through some  
125103           indirect path. (For example, *fopen()* might use *open()*; or *mkfifo()* might use *mknod()*.) There is  
125104           no necessary relationship between the type of implementation and its correctness, performance,  
125105           and/or reliability.
- 125106           **Implementation\***
- 125107           This term is generally used instead of its synonym, “system”, to emphasize the consequences of  
125108           decisions to be made by system implementors. Perhaps if no options or extensions to POSIX.1  
125109           were allowed, this usage would not have occurred.
- 125110           The term “specific implementation” is sometimes used as a synonym for “implementation”.  
125111           This should not be interpreted too narrowly; both terms can represent a relatively broad group  
125112           of systems. For example, a hardware vendor could market a very wide selection of systems that  
125113           all used the same instruction set, with some systems desktop models and others large multi-user  
125114           minicomputers. This wide range would probably share a common POSIX.1 operating system,  
125115           allowing an application compiled for one to be used on any of the others; this is a [*specific*]  
125116           *implementation*. However, such a wide range of machines probably has some differences  
125117           between the models. Some may have different clock rates, different file systems, different  
125118           resource limits, different network connections, and so on, depending on their sizes or intended  
125119           usages. Even on two identical machines, the system administrators may configure them  
125120           differently. Each of these different systems is known by the term “a specific instance of a specific  
125121           implementation”. This term is only used in the portions of POSIX.1 dealing with runtime  
125122           queries: *sysconf()* and *pathconf()*.
- 125123           **Incomplete Pathname\***
- 125124           Absolute pathname has been adequately defined.
- 125125           **Interactive Device**
- 125126           Austin Group Defect 1347 is applied, adding a definition of interactive device.

**125127 Intrinsic Utility**

125128 Austin Group Defect 854 is applied, adding intrinsic utilities.

**125129 Job**

125130 Austin Group Defect 1254 is applied, changing this definition.

**125131 Job Control**

125132 In order to understand the job control facilities in POSIX.1 it is useful to understand how they  
125133 are used by a job control-cognizant shell to create the user interface effect of job control.

125134 While the job control facilities supplied by POSIX.1 can, in theory, support different types of  
125135 interactive job control interfaces supplied by different types of shells, there was historically one  
125136 particular interface that was most common when the standard was originally developed  
125137 (provided by BSD C Shell).

125138 This discussion describes that interface as a means of illustrating how the POSIX.1 job control  
125139 facilities can be used.

125140 Job control allows users to selectively stop (suspend) the execution of processes and continue  
125141 (resume) their execution at a later point. The user typically employs this facility via the  
125142 interactive interface jointly supplied by the terminal I/O driver and a command interpreter  
125143 (shell).

125144 The user can launch jobs (command pipelines) in either the foreground or background. When  
125145 launched in the foreground, the shell waits for the job to complete before prompting for  
125146 additional commands. When launched in the background, the shell does not wait, but  
125147 immediately prompts for new commands.

125148 If the user launches a job in the foreground and subsequently regrets this, the user can type the  
125149 suspend character (typically set to <control>-Z), which causes the foreground process group to  
125150 stop, and the shell to convert the corresponding foreground job to a suspended job and begin  
125151 prompting for new commands. The suspended job can be continued by the user (via special  
125152 shell commands) either as a foreground job or as a background job. Background jobs can also be  
125153 moved into the foreground via shell commands.

125154 If a background process group attempts to access the login terminal (controlling terminal), it is  
125155 stopped by the terminal driver and the shell detects this and, in turn, suspends the  
125156 corresponding background job and notifies the user. (Terminal access includes *read()* and certain  
125157 terminal control functions, and conditionally includes *write()*.) The user can continue the  
125158 suspended job in the foreground, thus allowing the terminal access to succeed in an orderly  
125159 fashion. After the terminal access succeeds, the user can optionally move the job into the  
125160 background via the suspend character and shell commands.

**125161 Implementing Job Control Shells**

125162 The job control features of the POSIX shell (described in [Section 2.11](#), on page 2518) and of other  
125163 shells can be implemented using the job control facilities of the System Interfaces volume of  
125164 POSIX.1-2024 in the following way.

125165 The key feature necessary to provide job control is a way to group processes into jobs. This  
125166 grouping is necessary in order to direct signals to a single job and also to identify which job is in  
125167 the foreground. (There is at most one job that is in the foreground on any controlling terminal at  
125168 a time.)

125169 The concept of process groups is used to provide this grouping. The shell places the process(es)  
125170 it creates for each job in a separate process group via the *setpgid()* function. To do this, the



125171 *setpgid()* function is invoked by the shell for each process in the job. It is actually useful to  
125172 invoke *setpgid()* twice for each process: once in the child process, after calling *fork()* to create the  
125173 process, but before calling one of the *exec* family of functions to begin execution of the program,  
125174 and once in the parent shell process, after calling *fork()* to create the child. The redundant  
125175 invocation avoids a race condition by ensuring that the child process is placed into the new  
125176 process group before either the parent or the child relies on this being the case. The process  
125177 group ID for the job is selected by the shell to be equal to the process ID of one of the processes  
125178 in the job. Some shells choose to make one process in the job be the parent of the other processes  
125179 in the job (if any). Other shells (for example, the C Shell) choose to make themselves the parent  
125180 of all processes in the job. In order to support this latter case, the *setpgid()* function accepts a  
125181 process group ID parameter since the correct process group ID cannot be inherited from the  
125182 shell.

125183 The shell also controls which job is currently in the foreground. A foreground and background  
125184 job differ in two ways: the shell waits for a foreground command to complete (or stop) before  
125185 continuing to read new commands, and the terminal I/O driver inhibits terminal access by  
125186 background jobs (causing the processes to stop). Thus, the shell must work cooperatively with  
125187 the terminal I/O driver and have a common understanding of which job is currently in the  
125188 foreground. It is the user who decides which command should be currently in the foreground,  
125189 and the user informs the shell via shell commands. The shell, in turn, informs the terminal I/O  
125190 driver via the *tcsetpgrp()* function. This indicates to the terminal I/O driver the process group ID  
125191 of the foreground process group. When the current foreground job is either suspended or  
125192 terminated, the shell places its own process group in the foreground via *tcsetpgrp()* before  
125193 prompting for additional commands. Note that when a job is created the new process group  
125194 begins as a background process group. It requires an explicit act of the shell via *tcsetpgrp()*  
125195 to move a process group into the foreground.

125196 When a process in a job stops or terminates, its parent (for example, the shell) receives  
125197 synchronous notification by calling the *waitpid()* function with the WUNTRACED flag set.  
125198 Asynchronous notification is also provided when the parent establishes a signal handler for  
125199 SIGCHLD and does not specify the SA\_NOCLDSTOP flag. Usually all processes in a job stop as  
125200 a unit since the terminal I/O driver always sends job control stop signals to all processes in the  
125201 process group.

125202 To continue a suspended job, the shell sends a SIGCONT signal to the corresponding process  
125203 group. In addition, if the job is being continued in the foreground, the shell invokes *tcsetpgrp()*  
125204 to place the process group in the foreground before sending SIGCONT. Otherwise, the shell  
125205 leaves itself in the foreground and reads additional commands.

125206 There is additional flexibility in the POSIX.1 job control facilities that allows deviations from the  
125207 typical interface. Clearing the TOSTOP terminal flag allows background jobs to perform *write()*  
125208 functions without stopping. The same effect can be achieved on a per-process basis by having a  
125209 process set the signal action for SIGTTOU to SIG\_IGN.

125210 A login session that is not using the job control facilities can be thought of as a large collection of  
125211 processes that are all in the same job. Such a login session may have a partial distinction  
125212 between foreground and background processes; that is, the shell waits for some processes before  
125213 continuing to read new commands and does not wait for other processes. However, the terminal  
125214 I/O driver considers all these processes to be in the foreground since they are all members of the  
125215 same process group.

125216 In addition to the basic job control operations already mentioned, a job control-cognizant shell  
125217 needs to perform the following actions.

125218 When a foreground (not background) job is suspended, the shell needs to sample and remember  
125219 the current terminal settings so that it can restore them later when it continues the suspended

- 125220 job in the foreground (via the *tcgetattr()* and *tcsetattr()* functions).
- 125221 Because a shell itself can be spawned from a shell, it must take special action to ensure that child  
125222 shells interact well with their parent shells. A child shell can be spawned to perform an  
125223 interactive function (prompting the terminal for commands) or a non-interactive function  
125224 (reading commands from a file). When operating non-interactively, the job control shell will by  
125225 default refrain from performing the job control-specific actions described above. It will behave  
125226 as a shell that does not support job control. For example, all jobs will be left in the same process  
125227 group as the shell, which itself remains in the process group established for it by its parent. This  
125228 allows the shell and its children to be treated as a single job by a parent shell, and they can be  
125229 affected as a unit by terminal keyboard signals.
- 125230 An interactive child shell can be spawned from another job control-cognizant shell in either the  
125231 foreground or background. (For example, the user can execute an interactive shell in the  
125232 background by means of the command "sh &".) Before the child shell activates job control by  
125233 calling *setpgid()* to place itself in its own process group and *tcsetpgrp()* to place its new process  
125234 group in the foreground, it needs to ensure that it has already been placed in the foreground by  
125235 its parent. (Otherwise, there could be multiple job control shells that simultaneously attempt to  
125236 control mediation of the terminal.) To determine this, the shell retrieves its own process group  
125237 via *getpgrp()* and the process group of the current foreground job via *tcgetpgrp()*. If these are not  
125238 equal, the shell sends SIGTTIN to its own process group, causing itself to stop. When continued  
125239 later by its parent, the shell repeats the process group check. When the process groups finally  
125240 match, the shell is in the foreground and it can proceed to take control. After this point, the shell  
125241 ignores all the job control stop signals so that it does not inadvertently stop itself.
- 125242 *Implementing Job Control Applications*
- 125243 Most applications do not need to be aware of job control signals and operations; the intuitively  
125244 correct behavior happens by default. However, sometimes an application can inadvertently  
125245 interfere with normal job control processing, or an application may choose to overtly effect job  
125246 control in cooperation with normal shell procedures.
- 125247 An application can inadvertently subvert job control processing by "blindly" altering the  
125248 handling of signals. A common application error is to learn how many signals the system  
125249 supports and to ignore or catch them all. Such an application makes the assumption that it does  
125250 not know what this signal is, but knows the right handling action for it. The system may  
125251 initialize the handling of job control stop signals so that they are being ignored. This allows  
125252 shells that do not support job control to inherit and propagate these settings and hence to be  
125253 immune to stop signals. A job control shell will set the handling to the default action and  
125254 propagate this, allowing processes to stop. In doing so, the job control shell is taking  
125255 responsibility for restarting the stopped applications. If an application wishes to catch the stop  
125256 signals itself, it should first determine their inherited handling states. If a stop signal is being  
125257 ignored, the application should continue to ignore it. This is directly analogous to the  
125258 recommended handling of SIGINT described in the referenced UNIX Programmer's Manual.
- 125259 If an application is reading the terminal and has disabled the interpretation of special characters  
125260 (by clearing the ISIG flag), the terminal I/O driver will not send SIGTSTP when the suspend  
125261 character is typed. Such an application can simulate the effect of the suspend character by  
125262 recognizing it and sending SIGTSTP to its process group as the terminal driver would have  
125263 done. Note that the signal is sent to the process group, not just to the application itself; this  
125264 ensures that other processes in the job also stop. (Note also that other processes in the job could  
125265 be children, siblings, or even ancestors.) Applications should not assume that the suspend  
125266 character is <control>-Z (or any particular value); they should retrieve the current setting at  
125267 startup.

- 125268 *Implementing Job Control Systems*
- 125269 The intent in adding 4.2 BSD-style job control functionality was to adopt the necessary 4.2 BSD  
125270 programmatic interface with only minimal changes to resolve syntactic or semantic conflicts  
125271 with System V or to close recognized security holes. The goal was to maximize the ease of  
125272 providing both conforming implementations and Conforming POSIX.1 Applications.
- 125273 It is only useful for a process to be affected by job control signals if it is the descendant of a job  
125274 control shell. Otherwise, there will be nothing that continues the stopped process.
- 125275 POSIX.1 does not specify how controlling terminal access is affected by a user logging out (that  
125276 is, by a controlling process terminating). 4.2 BSD uses the *vhangup()* function to prevent any  
125277 access to the controlling terminal through file descriptors opened prior to logout. System V does  
125278 not prevent controlling terminal access through file descriptors opened prior to logout (except  
125279 for the case of the special file, */dev/tty*). Some implementations choose to make processes  
125280 immune from job control after logout (that is, such processes are always treated as if in the  
125281 foreground); other implementations continue to enforce foreground/background checks after  
125282 logout. Therefore, a Conforming POSIX.1 Application should not attempt to access the  
125283 controlling terminal after logout since such access is unreliable. If an implementation chooses to  
125284 deny access to a controlling terminal after its controlling process exits, POSIX.1 requires a certain  
125285 type of behavior (see [Controlling Terminal](#), on page 3651).
- 125286 Austin Group Defect 1254 is applied, changing this definition.
- 125287 **Job ID**
- 125288 Austin Group Defect 1254 is applied, changing ``job control job ID'' to ``job ID''.
- 125289 **Joinable Thread**
- 125290 Austin Group Defect 792 is applied, adding this definition.
- 125291 **Kernel\***
- 125292 See [System Call\\*](#) (on page 3673).
- 125293 **Library Routine\***
- 125294 See [System Call\\*](#) (on page 3673).
- 125295 **Link**
- 125296 Austin Group Defect 1380 is applied, changing this definition.
- 125297 **Live Process**
- 125298 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0010 [690] is applied.
- 125299 **Live Thread**
- 125300 Austin Group Defect 792 is applied, adding this definition.

- 125301        **Logical Device\***
- 125302        Implementation-defined.
- 125303        **Map**
- 125304        The definition of map is included to clarify the usage of mapped pages in the description of the  
125305        behavior of process memory locking.
- 125306        **Memory-Resident**
- 125307        The term “memory-resident” is historically understood to mean that the so-called resident pages  
125308        are actually present in the physical memory of the computer system and are immune from  
125309        swapping, paging, copy-on-write faults, and so on. This is the actual intent of POSIX.1-2024 in  
125310        the process memory locking section for implementations where this is logical. But for some  
125311        implementations—primarily mainframes—actually locking pages into primary storage is not  
125312        advantageous to other system objectives, such as maximizing throughput. For such  
125313        implementations, memory locking is a “hint” to the implementation that the application wishes  
125314        to avoid situations that would cause long latencies in accessing memory. Furthermore, there are  
125315        other implementation-defined issues with minimizing memory access latencies that “memory  
125316        residency” does not address—such as MMU reload faults. The definition attempts to  
125317        accommodate various implementations while allowing conforming applications to specify to the  
125318        implementation that they want or need the best memory access times that the implementation  
125319        can provide.
- 125320        **Memory Object\***
- 125321        The term “memory object” usually implies shared memory. If the object is the same as a  
125322        filename in the file system name space of the implementation, it is expected that the data written  
125323        into the memory object be preserved on disk. A memory object may also apply to a physical  
125324        device on an implementation. In this case, writes to the memory object are sent to the controller  
125325        for the device and reads result in control registers being returned.
- 125326        **Messages Object**
- 125327        Austin Group Defect 1122 is applied, adding this definition.
- 125328        **Mounted File System\***
- 125329        See [File System](#) (on page 3654).
- 125330        **Multi-Threaded Library**
- 125331        POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.
- 125332        **Multi-Threaded Process**
- 125333        POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

**125334 Multi-Threaded Program**

125335 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

125336 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

**125337 Mutex**

125338 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

**125339 Name**

125340 There are no explicit limits in POSIX.1-2024 on the sizes of names, words (see the definition of  
125341 word in the Base Definitions volume of POSIX.1-2024), lines, or other objects. However, other  
125342 implicit limits do apply: shell script lines produced by many of the standard utilities cannot  
125343 exceed {LINE\_MAX} and the sum of exported variables comes under the {ARG\_MAX} limit.  
125344 Historical shells dynamically allocate memory for names and words and parse incoming lines a  
125345 character at a time. Lines cannot have an arbitrary {LINE\_MAX} limit because of historical  
125346 practice, such as *makefiles*, where *make* removes the <newline> characters associated with the  
125347 commands for a target and presents the shell with one very long line. The text on INPUT FILES  
125348 in XCU [Section 1.4](#) (on page 2462) does allow a shell to run out of memory, but it cannot have  
125349 arbitrary programming limits.

**125350 Native Implementation\***

125351 This refers to an implementation of POSIX.1 that interfaces directly to an operating system  
125352 kernel; see also *hosted implementation*. A similar concept is a native UNIX system, which would  
125353 be a kernel derived from one of the original UNIX system products.

**125354 Negative**

125355 Austin Group Defect 1428 is applied, adding this definition.

**125356 Nice Value**

125357 This definition is not intended to suggest that all processes in a system have priorities that are  
125358 comparable. Scheduling policy extensions, such as adding realtime priorities, make the notion of  
125359 a single underlying priority for all scheduling policies problematic. Some implementations may  
125360 implement the features related to *nice* to affect all processes on the system, others to affect just  
125361 the general time-sharing activities implied by POSIX.1-2024, and others may have no effect at all.  
125362 Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
125363 implementation strategies is possible.

**125364 Null Pointer**

125365 Austin Group Defect 940 is applied, adding a statement that any pointer object whose  
125366 representation has all bits set to zero will be interpreted as a null pointer.

**125367 Null Terminator**

125368 Austin Group Defect 1621 is applied, adding this definition.

**125369 OFD-Owned File Lock**

125370 Austin Group Defect 768 is applied, adding this definition.

**125371 Open File Description**

125372 An “open file description”, as it is currently named, describes how a file is being accessed. What  
125373 is currently called a “file descriptor” is actually just an identifier or “handle”; it does not  
125374 actually describe anything.

125375 The following alternate names were discussed:

- 125376 • For “open file description”:  
125377 “open instance”, “file access description”, “open file information”, and “file access  
125378 information”.
- 125379 • For “file descriptor”:  
125380 “file handle”, “file number” (cf., *fileno()*). Some historical implementations use the term  
125381 “file table entry”.

**125382 Option-Argument**

125383 Austin Group Defect 1784 is applied, changing this definition.

**125384 Orphaned Process Group**

125385 Historical implementations have a concept of an orphaned process, which is a process whose  
125386 parent process has exited. When job control is in use, it is necessary to prevent processes from  
125387 being stopped in response to interactions with the terminal after they no longer are controlled by  
125388 a job control-cognizant program. Because signals generated by the terminal are sent to a process  
125389 group and not to individual processes, and because a signal may be provoked by a process that  
125390 is not orphaned, but sent to another process that is orphaned, it is necessary to define an  
125391 orphaned process group. The definition assumes that a process group will be manipulated as a  
125392 group and that the job control-cognizant process controlling the group is outside of the group  
125393 and is the parent of at least one process in the group (so that state changes may be reported via  
125394 *waitpid()*). Therefore, a group is considered to be controlled as long as at least one process in the  
125395 group has a parent that is outside of the process group, but within the session.

125396 This definition of orphaned process groups ensures that a session leader’s process group is  
125397 always considered to be orphaned, and thus it is prevented from stopping in response to  
125398 terminal signals.

**125399 Page**

125400 The term “page” is defined to support the description of the behavior of memory mapping for  
125401 shared memory and memory mapped files, and the description of the behavior of process  
125402 memory locking. It is not intended to imply that shared memory/file mapping and memory  
125403 locking are applicable only to “paged” architectures. For the purposes of POSIX.1-2024,  
125404 whatever the granularity on which an architecture supports mapping or locking, this is  
125405 considered to be a “page”. If an architecture cannot support the memory mapping or locking  
125406 functions specified by POSIX.1-2024 on any granularity, then these options will not be  
125407 implemented on the architecture.

125408 **Pathname**

125409 Pathnames historically allowed all bytes except for the <slash> and <NUL> characters. For  
 125410 compatibility with existing file systems, this usage is maintained throughout the standard by  
 125411 noting that a pathname need not be a valid character string in all locales. However, the  
 125412 properties of the portable filename character set are such that a pathname using only those  
 125413 characters and the <slash> is portable in all locales as a character string.

125414 Austin Group Defect 1073 is applied, making it implementation-defined whether the case of  
 125415 exactly two leading <slash> characters is treated specially.

125416 **Passwd File\***

125417 Implementation-defined; see [User Database](#) (on page 3675).

125418 **Parent Directory**

125419 There may be more than one directory entry pointing to a given directory in some  
 125420 implementations. The wording here identifies that exactly one of those is the parent directory. In  
 125421 pathname resolution, dot-dot is identified as the way that the unique directory is identified.  
 125422 (That is, the parent directory is the one to which dot-dot points.) In the case of a remote file  
 125423 system, if the same file system is mounted several times, it would appear as if they were distinct  
 125424 file systems (with interesting synchronization properties).

125425 **Pattern**

125426 Austin Group Defect 1443 is applied, changing this definition to be inclusive of all uses of shell  
 125427 pattern matching notation.

125428 **Pipe**

125429 It proved convenient to define a pipe as a special case of a FIFO, even though historically the  
 125430 latter was not introduced until System III and does not exist at all in 4.3 BSD.

125431 **Portable Filename**

125432 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0012 [584] is applied.

125433 **Portable Filename Character Set**

125434 The encoding of this character set is not specified—specifically, ASCII is not required. But the  
 125435 implementation must provide a unique character code for each of the printable graphics  
 125436 specified by POSIX.1; see also [Section A.4.9](#) (on page 3678).

125437 Situations where characters beyond the portable filename character set (or historically ASCII or  
 125438 the ISO/IEC 646:1991 standard) would be used (in a context where the portable filename  
 125439 character set or the ISO/IEC 646:1991 standard is required by POSIX.1) are expected to be  
 125440 common. Although such a situation renders the use technically non-compliant, mutual  
 125441 agreement among the users of an extended character set will make such use portable between  
 125442 those users. Such a mutual agreement could be formalized as an optional extension to POSIX.1.  
 125443 (Making it required would eliminate too many possible systems, as even those systems using the  
 125444 ISO/IEC 646:1991 standard as a base character set extend their character sets for Western  
 125445 Europe and the rest of the world in different ways.)

125446 Nothing in POSIX.1 is intended to preclude the use of extended characters where interchange is  
 125447 not required or where mutual agreement is obtained. It has been suggested that in several places  
 125448 “should” be used instead of “shall”. Because (in the worst case) use of any character beyond the  
 125449 portable filename character set would render the program or data not portable to all possible

- 125450 systems, no extensions are permitted in this context.
- 125451 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0013 [584] is applied.
- 125452 **Portable Messages Object Source File (or Dot-Po File)**
- 125453 Austin Group Defect 1122 is applied, adding this definition.
- 125454 **Positional Parameter**
- 125455 Austin Group Defect 1514 is applied, changing this definition in line with earlier changes to the  
125456 cross-reference to which it refers.
- 125457 **Positive**
- 125458 Austin Group Defect 1428 is applied, adding this definition.
- 125459 **Process**
- 125460 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.
- 125461 **Process Lifetime**
- 125462 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/5 is applied, adding *fork()*, *posix\_spawn()*,  
125463 *posix\_spawnp()*, and *vfork()* to the list of functions.
- 125464 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.
- 125465 **Process Termination**
- 125466 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/6 is applied, rewording the definition to  
125467 address the “passive exit” on termination of the last thread or the *\_Exit()* function.
- 125468 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.
- 125469 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.
- 125470 **Process-Owned File Lock**
- 125471 Austin Group Defect 768 is applied, adding this definition.
- 125472 **Pseudo-Terminal**
- 125473 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal  
125474 devices.
- 125475 **Radix Character (or Decimal-Point Character)**
- 125476 Austin Group Defect 1449 is applied, adding “(or Decimal-Point Character)”.
- 125477 **Record Lock**
- 125478 Austin Group Defect 768 is applied, adding this definition.



- 125479           **Regular Built-In Utility (or Regular Built-In)**
- 125480           Austin Group Defect 850 is applied, adding this entry as a pointer to the *Built-In Utility*
- 125481           definition.
- 125482           **Regular File**
- 125483           POSIX.1 does not intend to preclude the addition of structuring data (for example, record
- 125484           lengths) in the file, as long as such data is not visible to an application that uses the features
- 125485           described in POSIX.1.
- 125486           **Root Directory**
- 125487           This definition permits the operation of *chroot()*, even though that function is not in POSIX.1; see
- 125488           also [Section A.4.8](#) (on page 3678).
- 125489           **Root File System\***
- 125490           Implementation-defined.
- 125491           **Root of a File System\***
- 125492           Commonly used to refer to a mount point; this standard uses the latter.
- 125493           **Signal**
- 125494           The definition implies a double meaning for the term. Although a signal is an event, common
- 125495           usage implies that a signal is an identifier of the class of event.
- 125496           **Single-Threaded Process**
- 125497           POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.
- 125498           **Single-Threaded Program**
- 125499           POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.
- 125500           Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.
- 125501           **Source Code**
- 125502           POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0015 [896] is applied.
- 125503           **Sparse File**
- 125504           Austin Group Defect 415 is applied, adding this definition.
- 125505           **Special Built-In Utility (or Special Built-In)**
- 125506           Austin Group Defect 1583 is applied, clarifying that “special built-in utility” and “special built-
- 125507           in” are equivalent terms.

- 125508           **Standard Error**
- 125509           Austin Group Defect 1493 is applied, expanding this definition to cover uses of the term outside  
125510           the XSH volume.
- 125511           **Standard Input**
- 125512           Austin Group Defect 1493 is applied, expanding this definition to cover uses of the term outside  
125513           the XSH volume.
- 125514           **Standard Output**
- 125515           Austin Group Defect 1493 is applied, expanding this definition to cover uses of the term outside  
125516           the XSH volume.
- 125517           **Stream**
- 125518           Austin Group Defect 1371 is applied, updating the stream definition so that it applies to the shell  
125519           command language as well as the C language.
- 125520           **Superuser\***
- 125521           This concept, with great historical significance to UNIX system users, has been replaced with the  
125522           notion of appropriate privileges.
- 125523           **Supplementary Group ID**
- 125524           The POSIX.1-1990 standard is inconsistent in its treatment of supplementary groups. The  
125525           definition of supplementary group ID explicitly permits the effective group ID to be included in  
125526           the set, but wording in the description of the *setuid()* and *setgid()* functions states: “Any  
125527           supplementary group IDs of the calling process remain unchanged by these function calls”.  
125528           In the case of *setgid()* this contradicts that definition. In addition, some felt that the unspecified  
125529           behavior in the definition of supplementary group IDs adds unnecessary portability problems.  
125530           The standard developers considered several solutions to this problem:
- 125531           1.   Reword the description of *setgid()* to permit it to change the supplementary group IDs to  
125532           reflect the new effective group ID. A problem with this is that it adds more “may”s to the  
125533           wording and does not address the portability problems of this optional behavior.
  - 125534           2.   Mandate the inclusion of the effective group ID in the supplementary set (giving  
125535           {NGROUPS\_MAX} a minimum value of 1). This is the behavior of 4.4 BSD. In that  
125536           system, the effective group ID is the first element of the array of supplementary group  
125537           IDs (there is no separate copy stored, and changes to the effective group ID are made only  
125538           in the supplementary group set). By convention, the initial value of the effective group ID  
125539           is duplicated elsewhere in the array so that the initial value is not lost when executing a  
125540           set-group-ID program.
  - 125541           3.   Change the definition of supplementary group ID to exclude the effective group ID and  
125542           specify that the effective group ID does not change the set of supplementary group IDs.  
125543           This is the behavior of 4.2 BSD, 4.3 BSD, and System V Release 4.
  - 125544           4.   Change the definition of supplementary group ID to exclude the effective group ID, and  
125545           require that *getgroups()* return the union of the effective group ID and the supplementary  
125546           group IDs.
  - 125547           5.   Change the definition of {NGROUPS\_MAX} to be one more than the number of  
125548           supplementary group IDs, so it continues to be the number of values returned by  
125549           *getgroups()* and existing applications continue to work. This alternative is effectively the

125550 same as the second (and might actually have the same implementation).

125551 The standard developers decided to permit either 2 or 3. The effective group ID is orthogonal to  
 125552 the set of supplementary group IDs, and it is implementation-defined whether *getgroups()*  
 125553 returns this. If the effective group ID is returned with the set of supplementary group IDs, then  
 125554 all changes to the effective group ID affect the supplementary group set returned by *getgroups()*.  
 125555 It is permissible to eliminate duplicates from the list returned by *getgroups()*. However, if a  
 125556 group ID is contained in the set of supplementary group IDs, setting the group ID to that value  
 125557 and then to a different value should not remove that value from the supplementary group IDs.

125558 The definition of supplementary group IDs has been changed to not include the effective group  
 125559 ID. This simplifies permanent rationale and makes the relevant functions easier to understand.  
 125560 The *getgroups()* function has been modified so that it can, on an implementation-defined basis,  
 125561 return the effective group ID. By making this change, functions that modify the effective group  
 125562 ID do not need to discuss adding to the supplementary group list; the only view into the  
 125563 supplementary group list that the application developer has is through the *getgroups()* function.

### 125564 **Suspended Job**

125565 Austin Group Defect 1254 is applied, changing this definition.

### 125566 **Symbolic Constant**

125567 Earlier versions of this standard used a variety of terms other than “macro” for many of the  
 125568 constants defined in headers, and it was not clear in which of these cases they were required to  
 125569 be macros or not, or to be pre-processor constants (i.e., usable in *#if*) or not. In cases where the  
 125570 symbols had a reserved prefix or suffix, there was often inconsistency between whether the  
 125571 prefix/suffix was reserved only for macros or for any use, and whether the term “macro” or a  
 125572 different term was used in the descriptions of the symbols. There were also some unintentional  
 125573 differences from the ISO C standard.

125574 One of the most commonly used terms was “symbolic constant”. This has now been designated  
 125575 as the default term to be used wherever appropriate, and a formal definition of the term has  
 125576 been added giving the exact requirements for symbols that are described as symbolic constants.

125577 The standard developers have performed a major rationalization of the header descriptions of  
 125578 symbols with constant values according to the following policy:

125579 • Where symbols are from the ISO C standard, the wording from the ISO C standard (or  
 125580 equivalent, in cases where the exact wording is not appropriate) is used to describe them.

125581 • For all other constants, the first choice is to use “symbolic constant” when the  
 125582 requirements for the symbol are a reasonably close fit with those of the term.

125583 The description of the symbol can override individual requirements for symbolic  
 125584 constants; e.g., to specify a non-integer type, or to add a requirement that the symbol is  
 125585 usable in *#if* preprocessor directives.

125586 • When neither of the above apply, the exact requirements are stated in the description.  
 125587 (Note that macros are not required to be usable in *#if*, or even to expand to constant  
 125588 expressions, unless explicitly stated.)

125589 • In cases where there is a reserved prefix or suffix, if the symbol(s) with that prefix/suffix  
 125590 are from the ISO C standard and are required to be macros, or if the symbol is required to  
 125591 be usable in *#if*, then the prefix/suffix is reserved for use only as macros. If the symbol(s)  
 125592 are “symbolic constants” and not required to be usable in *#if*, the prefix/suffix is reserved  
 125593 for any use except in a few special cases.

125594 Where a constant is required to be a macro but is also allowed to be another type of constant

125595 such as an enumeration constant, on implementations which do define it as another type of  
125596 constant the macro is typically defined as follows:

```
125597 #define macro_name macro_name
```

125598 This allows applications to use `#ifdef`, etc. to determine whether the macro is defined, but the  
125599 macro is not usable in `#if` preprocessor directives because the preprocessor will treat the  
125600 unexpanded word `macro_name` as having the value zero.

## 125601 Symbolic Link

125602 Earlier versions of this standard did not require symbolic links to have attributes such as  
125603 ownership and a file serial number. This was because the 4.4 BSD implementation did not have  
125604 them, and it was expected that other implementations may wish to do the same. However,  
125605 experience with 4.4 BSD has shown that symbolic links implemented in this way cause problems  
125606 for users and application developers, and later BSD systems have reverted to using `inodes` to  
125607 implement symbolic links. Allowing `no-inode` symbolic links also caused problems in the  
125608 standard. For example, leaving the `st_ino` value for symbolic links unspecified meant that the  
125609 common technique of comparing the `st_dev` and `st_ino` values for two pathnames to see if they  
125610 refer to the same file could only be used with `stat()` in conforming applications and not with  
125611 `lstat()`. The standard now requires symbolic links to have meaningful values for the same **struct**  
125612 **stat** fields as regular files, except for the file mode bits in `st_mode`. Historically, the file mode bits  
125613 were unused (the contents of a symbolic link could always be read), but implementations  
125614 differed as to whether the file mode bits (as returned in `st_mode` or reported by `ls -l`) were set  
125615 according to the `umask` or just to a fixed value such as `0777`. Accordingly, the standard requires  
125616 the file mode bits to be ignored by `readlink()` and when a symbolic link is followed during  
125617 pathname resolution, but leaves the corresponding part of the value returned in `st_mode`  
125618 unspecified.

125619 Historical implementations were followed when determining which interfaces should apply to  
125620 symbolic links. Interfaces that historically followed symbolic links include `chmod()`, `stat()`, and  
125621 `utime()`. Interfaces that historically did not follow symbolic links include `lstat()`, `rename()`,  
125622 `remove()`, `rmdir()`, and `unlink()`. For `chown()` and `link()`, historical implementations differed.  
125623 POSIX.1-2024 inherited the `lchown()` function from the Single UNIX Specification, Version 2, and  
125624 therefore requires `chown()` to follow symbolic links. Earlier versions of this standard required  
125625 `link()` to follow symbolic links, but with the addition of the `linkat()` function (which has a flag to  
125626 indicate whether to follow symbolic links), both behaviors are now allowed for `link()`.

125627 When the final component of a pathname is a symbolic link, the standard requires that a trailing  
125628 `<slash>` causes the link to be followed. This is the behavior of historical implementations. For  
125629 example, for `/a/b` and `/a/b/`, if `/a/b` is a symbolic link to a directory, then `/a/b` refers to the  
125630 symbolic link, and `/a/b/` refers to the directory to which the symbolic link points.

125631 Because a symbolic link and its referenced object coexist in the file system name space, confusion  
125632 can arise in distinguishing between the link itself and the referenced object. Historically, utilities  
125633 and system calls have adopted their own link following conventions in a somewhat *ad hoc*  
125634 fashion. Rules for a uniform approach are outlined here, although historical practice has been  
125635 adhered to as much as was possible. To promote consistent system use, user-written utilities are  
125636 encouraged to follow these same rules.

125637 Symbolic links are handled either by operating on the link itself, or by operating on the object  
125638 referenced by the link. In the latter case, an application or system call is said to “follow” the link.  
125639 Symbolic links may reference other symbolic links, in which case links are dereferenced until an  
125640 object that is not a symbolic link is found, a symbolic link that references a file that does not exist  
125641 is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on  
125642 the number of symbolic links that they will dereference before declaring it an error.)

125643 There are four domains for which default symbolic link policy is established in a system. In  
 125644 almost all cases, there are utility options that override this default behavior. The four domains  
 125645 are as follows:

- 125646 1. Symbolic links specified to system calls that take pathname arguments
- 125647 2. Symbolic links specified as command line pathname arguments to utilities that are not  
 125648 performing a traversal of a file hierarchy
- 125649 3. Symbolic links referencing files not of type directory, specified to utilities that are  
 125650 performing a traversal of a file hierarchy
- 125651 4. Symbolic links referencing files of type directory, specified to utilities that are performing  
 125652 a traversal of a file hierarchy

#### 125653 *First Domain*

125654 The first domain is considered in earlier rationale.

#### 125655 *Second Domain*

125656 The reason this category is restricted to utilities that are not traversing the file hierarchy is that  
 125657 some standard utilities take an option that specifies a hierarchical traversal, but by default  
 125658 operate on the arguments themselves. Generally, users specifying the option for a file hierarchy  
 125659 traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which  
 125660 may reference files outside of the hierarchy, are ignored. For example, *chown owner file* is a  
 125661 different operation from the same command with the **-R** option specified. In this example, the  
 125662 behavior of the command *chown owner file* is described here, while the behavior of the command  
 125663 *chown -R owner file* is described in the third and fourth domains.

125664 The general rule is that the utilities in this category follow symbolic links named as arguments.

125665 Exceptions in the second domain are:

- 125666 • The *mv* and *rm* utilities do not follow symbolic links named as arguments, but respectively  
 125667 attempt to rename or delete them.
- 125668 • The *ls* utility is also an exception to this rule. For compatibility with historical systems,  
 125669 when the **-R** option is not specified, the *ls* utility follows symbolic links named as  
 125670 arguments if the **-L** option is specified or if the **-F**, **-d**, or **-l** options are not specified. (If  
 125671 the **-L** option is specified, *ls* always follows symbolic links; it is the only utility where the  
 125672 **-L** option affects its behavior even though a tree walk is not being performed.)

125673 All other standard utilities, when not traversing a file hierarchy, always follow symbolic links  
 125674 named as arguments.

125675 Historical practice is that the **-h** option is specified if standard utilities are to act upon symbolic  
 125676 links instead of upon their targets. Examples of commands that have historically had a **-h** option  
 125677 for this purpose are the *chgrp*, *chown*, *file*, and *test* utilities.

#### 125678 *Third Domain*

125679 The third domain is symbolic links, referencing files not of type directory, specified to utilities  
 125680 that are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
 125681 command line pathname arguments or encountered during the traversal.)

125682 The intention of the Shell and Utilities volume of POSIX.1-2024 is that the operation that the  
 125683 utility is performing is applied to the symbolic link itself, if that operation is applicable to  
 125684 symbolic links. If the operation is not applicable to symbolic links, the symbolic link should be  
 125685 ignored. Specifically, by default, no change should be made to the file referenced by the symbolic  
 125686 link.

125687 *Fourth Domain*

125688 The fourth domain is symbolic links referencing files of type directory, specified to utilities that  
 125689 are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
 125690 command line pathname arguments or encountered during the traversal.)

125691 Most standard utilities do not, by default, indirect into the file hierarchy referenced by the  
 125692 symbolic link. (The Shell and Utilities volume of POSIX.1-2024 uses the informal term “physical  
 125693 walk” to describe this case. The case where the utility does indirect through the symbolic link is  
 125694 termed a “logical walk”.)

125695 There are three reasons for the default to be a physical walk:

- 125696 1. With very few exceptions, a physical walk has been the historical default on UNIX  
 125697 systems supporting symbolic links. Because some utilities (that is, *rm*) must default to a  
 125698 physical walk, regardless, changing historical practice in this regard would be confusing  
 125699 to users and needlessly incompatible.
- 125700 2. For systems where symbolic links have the historical file attributes (that is, *owner*, *group*,  
 125701 *mode*), defaulting to a logical traversal would require the addition of a new option to the  
 125702 commands to modify the attributes of the link itself. This is painful and more complex  
 125703 than the alternatives.
- 125704 3. There is a security issue with defaulting to a logical walk. Historically, the command  
 125705 *chown -R user file* has been safe for the superuser because *setuid* and *setgid* bits were lost  
 125706 when the ownership of the file was changed. If the walk were logical, changing  
 125707 ownership would no longer be safe because a user might have inserted a symbolic link  
 125708 pointing to any file in the tree. Again, this would necessitate the addition of an option to  
 125709 the commands doing hierarchy traversal to not indirect through the symbolic links, and  
 125710 historical scripts doing recursive walks would instantly become security problems. While  
 125711 this is mostly an issue for system administrators, it is preferable to not have different  
 125712 defaults for different classes of users.

125713 However, the standard developers agreed to leave it unspecified to achieve consensus.

125714 As consistently as possible, users may cause standard utilities performing a file hierarchy  
 125715 traversal to follow any symbolic links named on the command line, regardless of the type of file  
 125716 they reference, by specifying the **-H** (for half logical) option. This option is intended to make the  
 125717 command line name space look like the logical name space.

125718 As consistently as possible, users may cause standard utilities performing a file hierarchy  
 125719 traversal to follow any symbolic links named on the command line as well as any symbolic links  
 125720 encountered during the traversal, regardless of the type of file they reference, by specifying the  
 125721 **-L** (for logical) option. This option is intended to make the entire name space look like the  
 125722 logical name space.

125723 For consistency, implementors are encouraged to use the **-P** (for “physical”) flag to specify the  
 125724 physical walk in utilities that do logical walks by default for whatever reason.

125725 When one or more of the **-H**, **-L**, and **-P** flags can be specified, the last one specified determines  
 125726 the behavior of the utility. This permits users to alias commands so that the default behavior is a  
 125727 logical walk and then override that behavior on the command line.

125728 *Exceptions in the Third and Fourth Domains*

125729 The *ls* and *rm* utilities are exceptions to these rules. The *rm* utility never follows symbolic links  
 125730 and does not support the **-H**, **-L**, or **-P** options. Some historical versions of *ls* always followed  
 125731 symbolic links given on the command line whether the **-L** option was specified or not.  
 125732 Historical versions of *ls* did not support the **-H** option. In POSIX.1-2024, unless one of the **-H** or

125733            –**L** options is specified, the *ls* utility only follows symbolic links to directories that are given as  
125734            operands. The *ls* utility does not support the –**P** option.

125735            The Shell and Utilities volume of POSIX.1-2024 requires that the standard utilities *ls*, *find*, and  
125736            *pax* detect infinite loops when doing logical walks; that is, a directory, or more commonly a  
125737            symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is  
125738            corrupted, causing the infinite loop, it may be impossible to recover. Because *find* and *ls* are often  
125739            used in system administration and security applications, they should attempt to recover and  
125740            continue as best as they can. The *pax* utility should terminate because the archive it was creating  
125741            is by definition corrupted. Other, less vital, utilities should probably simply terminate as well.  
125742            Implementations are strongly encouraged to detect infinite loops in all utilities.

125743            Historical practice is shown in [Table A-1](#) (on page 3672). The heading **SVID3** stands for the  
125744            Third Edition of the System V Interface Definition.

125745            Historically, several shells have had built-in versions of the *pwd* utility. In some of these shells,  
125746            *pwd* reported the physical path, and in others, the logical path. Implementations of the shell  
125747            corresponding to POSIX.1-2024 must report the logical path by default.

125748            The *cd* command is required, by default, to treat the filename dot-dot logically. Implementors are  
125749            required to support the –**P** flag in *cd* so that users can have their current environment handled  
125750            physically. In 4.3 BSD, *chgrp* during tree traversal changed the group of the symbolic link, not  
125751            the target. Symbolic links in 4.4 BSD did not have *owner*, *group*, *mode*, or other standard UNIX  
125752            system file attributes.

125753

Table A-1 Historical Practice for Symbolic Links

| Utility | SVID3   | 4.3 BSD | 4.4 BSD | POSIX | Comments                                |
|---------|---------|---------|---------|-------|-----------------------------------------|
| 125754  |         |         |         | -L    | Treat ". ." logically.                  |
| 125755  |         |         |         | -P    | Treat ". ." physically.                 |
| 125756  |         |         | -H      | -H    | Follow command line symlinks.           |
| 125757  |         |         | -h      | -L    | Follow symlinks.                        |
| 125758  |         |         |         | -h    | Affect the symlink.                     |
| 125759  | -h      |         |         |       | Affect the symlink.                     |
| 125760  |         |         | -H      |       | Follow command line symlinks.           |
| 125761  |         |         | -h      |       | Follow symlinks.                        |
| 125762  |         |         | -H      | -H    | Follow command line symlinks.           |
| 125763  |         |         | -h      | -L    | Follow symlinks.                        |
| 125764  |         |         |         | -h    | Affect the symlink.                     |
| 125765  | -h      |         |         |       | Affect the symlink.                     |
| 125766  |         |         | -H      | -H    | Follow command line symlinks.           |
| 125767  |         |         | -h      | -L    | Follow symlinks.                        |
| 125768  | -L      |         | -L      |       | Follow symlinks.                        |
| 125769  |         |         | -H      | -H    | Follow command line symlinks.           |
| 125770  |         |         | -h      | -L    | Follow symlinks.                        |
| 125771  | -h      |         |         | -h    | Affect the symlink.                     |
| 125772  |         |         | -H      | -H    | Follow command line symlinks.           |
| 125773  |         |         | -h      | -L    | Follow symlinks.                        |
| 125774  | -follow |         | -follow |       | Follow symlinks.                        |
| 125775  | -s      | -s      | -s      | -s    | Create a symbolic link.                 |
| 125776  | -L      | -L      | -L      | -L    | Follow symlinks.                        |
| 125777  |         |         |         | -H    | Follow command line symlinks.           |
| 125778  |         |         |         |       | Operates on the symlink.                |
| 125779  |         |         | -H      | -H    | Follow command line symlinks.           |
| 125780  |         |         | -h      | -L    | Follow symlinks.                        |
| 125781  |         |         |         | -L    | Printed path may contain symlinks.      |
| 125782  |         |         |         | -P    | Printed path will not contain symlinks. |
| 125783  |         |         |         |       | Operates on the symlink.                |
| 125784  |         |         | -H      |       | Follow command line symlinks.           |
| 125785  |         | -h      | -h      |       | Follow symlinks.                        |
| 125786  | -h      |         | -h      | -h    | Affect the symlink.                     |

125787

**Synchronized I/O Data Integrity Completion**

125788

Austin Group Defect 672 is applied, clarifying how this definition applies to directories, and that it does not apply to symbolic links.

125789

125790

**Synchronously-Generated Signal**

125791

Those signals that may be generated synchronously include SIGABRT, SIGBUS, SIGILL, SIGFPE, SIGPIPE, and SIGSEGV.

125792

125793

Any signal sent via the *raise()* function or a *kill()* function targeting the current process is also considered synchronous.

125794



**125795 System Call\***

125796 The distinction between a “system call” and a “library routine” is an implementation detail that  
125797 may differ between implementations and has thus been excluded from POSIX.1.

125798 See “Interface, Not Implementation” in the Preface.

**125799 System Console**

125800 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/7 is applied, changing from “An  
125801 implementation-defined device” to “A device”.

**125802 System Databases**

125803 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/9 is applied, rewording the definition to  
125804 reference the existing definitions for “group database” and “user database”.

**125805 System Process**

125806 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/8 is applied, rewording the definition to  
125807 remove the requirement for an implementation to define the object.

**125808 System Reboot**

125809 A “system reboot” is an event initiated by an unspecified circumstance that causes all processes  
125810 (other than special system processes) to be terminated in an implementation-defined manner,  
125811 after which any changes to the state and contents of files created or written to by a Conforming  
125812 POSIX.1 Application prior to the event are implementation-defined.

125813 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/10 is applied, changing “An  
125814 implementation-defined sequence of events” to “An unspecified sequence of events”.

**125815 Synchronized I/O Data (and File) Integrity Completion**

125816 These terms specify that for synchronized read operations, pending writes must be successfully  
125817 completed before the read operation can complete. This is motivated by two circumstances.  
125818 Firstly, when synchronizing processes can access the same file, but not share common buffers  
125819 (such as for a remote file system), this requirement permits the reading process to guarantee that  
125820 it can read data written remotely. Secondly, having data written synchronously is insufficient to  
125821 guarantee the order with respect to a subsequent write by a reading process, and thus this extra  
125822 read semantic is necessary.

**125823 Text Domain**

125824 Austin Group Defect 1122 is applied, adding this definition.

**125825 Text File**

125826 The term “text file” does not prevent the inclusion of control or other non-printable characters  
125827 (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either  
125828 able to process the special characters or they explicitly describe their limitations within their  
125829 individual descriptions. The definition of “text file” has caused controversy. The only difference  
125830 between text and binary files is that text files have lines of less than {LINE\_MAX} bytes, with no  
125831 NUL characters, each terminated by a <newline>. The definition allows a file with a single  
125832 <newline>, or a totally empty file, to be called a text file. If a file ends with an incomplete line it  
125833 is not strictly a text file by this definition. The <newline> referred to in POSIX.1-2024 is not some  
125834 generic line separator, but a single character; files created on systems where they use multiple  
125835 characters for ends of lines are not portable to all conforming systems without some translation

125836 process unspecified by POSIX.1-2024.

125837 **Thread**

125838 POSIX.1-2024 defines a live thread to be a flow of control within a process. Each thread has a  
125839 minimal amount of private state; most of the state associated with a process is shared among all  
125840 of the threads in the process. While most multi-thread extensions to POSIX have taken this  
125841 approach, others have made different decisions.

125842 **Note:** The choice to put threads within a process does not constrain implementations to implement  
125843 threads in that manner. However, all functions have to behave as though threads share the  
125844 indicated state information with the process from which they were created.

125845 Threads need to share resources in order to cooperate. Memory has to be widely shared between  
125846 threads in order for the threads to cooperate at a fine level of granularity. Threads keep data  
125847 structures and the locks protecting those data structures in shared memory. For a data structure  
125848 to be usefully shared between threads, such structures should not refer to any data that can only  
125849 be interpreted meaningfully by a single thread. Thus, any system resources that might be  
125850 referred to in data structures need to be shared between all threads. File descriptors, pathnames,  
125851 and pointers to stack variables are all things that programmers want to share between their  
125852 threads. Thus, the file descriptor table, the root directory, the current working directory, and the  
125853 address space have to be shared.

125854 Library implementations are possible as long as the effective behavior is as if system services  
125855 invoked by one thread do not suspend other threads. This may be difficult for some library  
125856 implementations on systems that do not provide asynchronous facilities.

125857 See [Section B.2.9](#) (on page 3806) for additional rationale.

125858 Austin Group Defect 792 is applied, changing this definition.

125859 **Thread ID**

125860 See [Section B.2.9.2](#) (on page 3823) for additional rationale.

125861 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125862 **Thread Lifetime**

125863 Austin Group Defect 792 is applied, adding this definition.

125864 **Thread Termination**

125865 Austin Group Defect 792 is applied, adding this definition.

125866 **Thread-Safe**

125867 All functions required by POSIX.1-2024 need to be thread-safe; see [Section A.4.22](#) (on page 3687)  
125868 and [Section B.2.9.1](#) (on page 3820) for additional rationale.

125869 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125870 **Thread-Specific Data Key**

125871 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

125872 **User Database**

125873 There are no references in POSIX.1-2024 to a “passwd file” or a “group file”, and there is no  
 125874 requirement that the *group* or *passwd* databases be kept in files containing editable text. Many  
 125875 large timesharing systems use *passwd* databases that are hashed for speed. Certain security  
 125876 classifications prohibit certain information in the *passwd* database from being publicly readable.

125877 The term “encoded” is used instead of “encrypted” in order to avoid the implementation  
 125878 connotations (such as reversibility or use of a particular algorithm) of the latter term.

125879 The *getgrent()*, *setgrent()*, *endgrent()*, *getpwent()*, *setpwent()*, and *endpwent()* functions are not  
 125880 included as part of the base standard because they provide a linear database search capability  
 125881 that is not generally useful (the *getpwuid()*, *getpwnam()*, *getgrgid()*, and *getgrnam()* functions are  
 125882 provided for keyed lookup) and because in certain distributed systems, especially those with  
 125883 different authentication domains, it may not be possible or desirable to provide an application  
 125884 with the ability to browse the system databases indiscriminately. They are provided on XSI-  
 125885 conformant systems due to their historical usage by many existing applications.

125886 A change from historical implementations is that the structures used by these functions have  
 125887 fields of the types **gid\_t** and **uid\_t**, which are required to be defined in the **<sys/types.h>** header.  
 125888 POSIX.1-2024 requires implementations to ensure that these types are defined by inclusion of  
 125889 **<grp.h>** and **<pwd.h>**, respectively, without imposing any name space pollution or errors from  
 125890 redefinition of types.

125891 POSIX.1-2024 is silent about the content of the strings containing user or group names. These  
 125892 could be digit strings. POSIX.1-2024 is also silent as to whether such digit strings bear any  
 125893 relationship to the corresponding (numeric) user or group ID.

125894 *Database Access*

125895 The thread-safe versions of the user and group database access functions return values in user-  
 125896 supplied buffers instead of possibly using static data areas that may be overwritten by each call.

125897 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/11 is applied, removing the words “of  
 125898 implementation-defined format”.

125899 **User ID**

125900 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0016 [511] is applied.

125901 **User Name**

125902 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0017 [584] is applied.

125903 **Virtual Processor\***

125904 The term “virtual processor” was chosen as a neutral term describing all kernel-level  
 125905 schedulable entities, such as processes, Mach tasks, or lightweight processes. Implementing  
 125906 threads using multiple processes as virtual processors, or implementing multiplexed threads  
 125907 above a virtual processor layer, should be possible, provided some mechanism has also been  
 125908 implemented for sharing state between processes or virtual processors. Many systems may also  
 125909 wish to provide implementations of threads on systems providing “shared processes” or  
 125910 “variable-weight processes”. It was felt that exposing such implementation details would  
 125911 severely limit the type of systems upon which the threads interface could be supported and  
 125912 prevent certain types of valid implementations. It was also determined that a virtual processor

- 125913 interface was out of the scope of the Rationale (Informative) volume of POSIX.1-2024.
- 125914 **White Space**
- 125915 Austin Group Defect 1163 is applied, clarifying the definition of white space and adding  
125916 definitions of white-space byte, white-space character, and white-space wide character.
- 125917 **XSI**
- 125918 This is included to allow POSIX.1-2024 to be adopted as an IEEE standard and a standard of The  
125919 Open Group, serving both POSIX and the Single UNIX Specification in a core set of volumes.
- 125920 When POSIX.1 and the Single UNIX Specification were merged, the term “XSI” had been used  
125921 for over 10 years in connection with the XPG series and the first and second versions of the base  
125922 volumes of the Single UNIX Specification. The XSI margin code was introduced to denote the  
125923 extended or more restrictive semantics beyond POSIX that are applicable to UNIX systems.
- 125924 **Zombie Process**
- 125925 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0018 [690] is applied.
- 125926 **Zombie Thread**
- 125927 Austin Group Defect 792 is applied, adding this definition.
- 125928 **A.4 General Concepts**
- 125929 The general concepts are similar in nature to the definitions section, with the exception that a  
125930 term defined in general concepts can contain normative requirements.
- 125931 **A.4.1 Case Insensitive Comparisons**
- 125932 Case-insensitive matching is defined in this standard in terms of a simple algorithm whereby, for  
125933 each character in the string to be matched, if the character is uppercase then the lowercase  
125934 equivalent (if any) is also checked for a match, and if the character is lowercase then the  
125935 uppercase equivalent (if any) is also checked for a match. It is described this way to make the  
125936 expected behavior easier to understand; however, implementations may internally use more  
125937 sophisticated algorithms to improve efficiency, provided that the result is the same as the simple  
125938 algorithm would produce.
- 125939 Austin Group Defect 1031 is applied, adding case insensitive comparisons.
- 125940 **A.4.2 Concurrent Execution**
- 125941 There is no additional rationale provided for this section.

**125942 A.4.3 Default Initialization**

125943 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0019 [934] is applied.

125944 Austin Group Defect 940 is applied, removing text that was conditional on the all-zero bit  
125945 pattern of a pointer object being a null pointer, as this is now mandated.

**125946 A.4.4 Directory Operations**

125947 Earlier versions of this standard did not make clear that directory modifications are performed  
125948 atomically and serially, although that is the historical behavior and was always intended.

125949 Austin Group Defect 672 is applied, adding this subsection.

**125950 A.4.5 Directory Protection**

125951 There is no additional rationale provided for this section.

**125952 A.4.6 Extended Security Controls**

125953 Allowing an implementation to define extended security controls enables the use of  
125954 POSIX.1-2024 in environments that require different or more rigorous security than that  
125955 provided in POSIX.1. Extensions are allowed in two areas: privilege and file access permissions.  
125956 The semantics of these areas have been defined to permit extensions with reasonable, but not  
125957 exact, compatibility with all existing practices. For example, the elimination of the superuser  
125958 definition precludes identifying a process as privileged or not by virtue of its effective user ID.

**125959 A.4.7 File Access Permissions**

125960 A process should not try to anticipate the result of an attempt to access data by *a priori* use of  
125961 these rules. Rather, it should make the attempt to access data and examine the return value (and  
125962 possibly *errno* as well), or use *access()*. An implementation may include other security  
125963 mechanisms in addition to those specified in POSIX.1, and an access attempt may fail because of  
125964 those additional mechanisms, even though it would succeed according to the rules given in this  
125965 section. (For example, the user's security level might be lower than that of the object of the  
125966 access attempt.) The supplementary group IDs provide another reason for a process to not  
125967 attempt to anticipate the result of an access attempt.

125968 Since the current standard does not specify a method for opening a directory for searching, it is  
125969 unspecified whether search permission on the *fd* argument to *openat()* and related functions is  
125970 based on whether the directory was opened with search mode or on the current permissions  
125971 allowed by the directory at the time a search is performed. When there is existing practice that  
125972 supports opening directories for searching, it is expected that these functions will be modified to  
125973 specify that the search permissions will be granted based on the file access modes of the  
125974 directory's file descriptor identified by *fd*, and not on the mode of the directory at the time the  
125975 directory is searched.

125976 **A.4.8 File Hierarchy**

125977 Though the file hierarchy is commonly regarded to be a tree, POSIX.1 does not define it as such  
125978 for three reasons:

- 125979 1. Links may join branches.
- 125980 2. In some network implementations, there may be no single absolute root directory; see  
125981 *pathname resolution*.
- 125982 3. With symbolic links, the file system need not be a tree or even a directed acyclic graph.

125983 **A.4.9 Filenames**

125984 Historically, certain filenames and pathnames have been reserved. This list includes **core**,  
125985 **/etc/passwd**, and so on. Conforming applications should avoid these.

125986 Most historical implementations prohibit case folding in filenames; that is, treating uppercase  
125987 and lowercase alphabetic characters as identical. However, some consider case folding desirable:

- 125988 • For user convenience
- 125989 • For ease-of-implementation of the POSIX.1 interface as a hosted system on some popular  
125990 operating systems

125991 Variants, such as maintaining case distinctions in filenames, but ignoring them in comparisons,  
125992 have been suggested. Methods of allowing escaped characters of the case opposite the default  
125993 have been proposed.

125994 Many reasons have been expressed for not allowing case folding, including:

- 125995 • No solid evidence has been produced as to whether case-sensitivity or case-insensitivity is  
125996 more convenient for users.
- 125997 • Making case-insensitivity a POSIX.1 implementation option would be worse than either  
125998 having it or not having it, because:
- 125999 — More confusion would be caused among users.
- 126000 — Application developers would have to account for both cases in their code.
- 126001 — POSIX.1 implementors would still have other problems with native file systems, such  
126002 as short or otherwise constrained filenames or pathnames, and the lack of  
126003 hierarchical directory structure.
- 126004 • Case folding is not easily defined in many European languages, both because many of  
126005 them use characters outside the US ASCII alphabetic set, and because:
- 126006 — In Spanish, the digraph "ll" is considered to be a single letter, the capitalized form  
126007 of which may be either "Ll" or "LL", depending on context.
- 126008 — In French, the capitalized form of a letter with an accent may or may not retain the  
126009 accent, depending on the country in which it is written.
- 126010 — In German, the sharp ess may be represented as a single character resembling a  
126011 Greek beta ( $\beta$ ) in lowercase, but as the digraph "SS" in uppercase.
- 126012 — In Greek, there are several lowercase forms of some letters; the one to use depends on  
126013 its position in the word. Arabic has similar rules.

- 126014 • Many East Asian languages, including Japanese, Chinese, and Korean, do not distinguish
- 126015 case and are sometimes encoded in character sets that use more than one byte per
- 126016 character.
- 126017 • Multiple character codes may be used on the same machine simultaneously. There are
- 126018 several ISO character sets for European alphabets. In Japan, several Japanese character
- 126019 codes are commonly used together, sometimes even in filenames; this is evidently also the
- 126020 case in China. To handle case insensitivity, the kernel would have to at least be able to
- 126021 distinguish for which character sets the concept made sense.
- 126022 • The file system implementation historically deals only with bytes, not with characters.
- 126023 Limitations on valid encodings ensure that the byte sequences for the <slash> character,
- 126024 <period> character, and <NUL> character will not be confused with any other character in
- 126025 any locale. However, there exist common single-shift encodings where other single-byte
- 126026 characters from the portable filename character set can also occur as a subset of a multi-
- 126027 byte character, making case folding of portable filename bytes dependent on the context of
- 126028 whether a shift-state is active.
- 126029 • The purpose of POSIX.1 is to standardize the common, existing definition, not to change it.
- 126030 Mandating case-insensitivity would make all historical implementations non-standard.
- 126031 • Not only the interface, but also application programs would need to change, counter to the
- 126032 purpose of having minimal changes to existing application code.
- 126033 • At least one of the original developers of the UNIX system has expressed objection in the
- 126034 strongest terms to either requiring case-insensitivity or making it an option, mostly on the
- 126035 basis that POSIX.1 should not hinder portability of application programs across related
- 126036 implementations in order to allow compatibility with unrelated operating systems.

126037 Two proposals were entertained regarding case folding in filenames:

- 126038 1. Remove all wording that previously permitted case folding.
- 126039 Rationale Case folding is inconsistent with the portable filename character set and
- 126040 filename definitions (all bytes except <slash> and null). No known
- 126041 implementations allowing all bytes except <slash> and null also do case
- 126042 folding.
- 126043 2. Change “though this practice is not recommended:” to “although this practice is strongly
- 126044 discouraged.”
- 126045 Rationale If case folding must be included in POSIX.1, the wording should be stronger
- 126046 to discourage the practice.

126047 The consensus selected the first proposal. Otherwise, a conforming application would have to

126048 assume that case folding would occur when it was not wanted, but that it would not occur when

126049 it was wanted.

#### 126050 A.4.10 Filename Portability

126051 Filenames should be constructed from the portable filename character set because the use of

126052 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a

126053 <colon> ( ' : ' ) in a pathname could cause ambiguity if that pathname were included in a *PATH*

126054 definition.)

126055 The constraint on use of the <hyphen-minus> character as the first character of a portable

126056 filename is a constraint on application behavior and not on implementations, since applications

126057 might not work as expected when such a filename is passed as a command line argument.

126058 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0020 [584] is applied.

#### 126059 **A.4.11 File System Cache**

126060 Earlier versions of this standard did not specify the behavior of *aio\_fsync()*, *fdatasync()*, or  
126061 *fsync()* on directories, nor did they specify constraints on the underlying storage in the absence  
126062 of calls to *aio\_fsync()*, *fdatasync()*, or *fsync()*.

126063 Although directory operations are atomic and serializable, they are not necessarily durable. An  
126064 application that requires a directory modification to be durable should use *fdatasync()* or *fsync()*  
126065 (or *aio\_fsync()*) on the directory. However, the intention of the requirements for directory  
126066 modifications is that most applications should not need to do this. For example, a common  
126067 method of updating a file is to create a new temporary file, call *fdatasync()* or *fsync()* to  
126068 synchronize the new file, and then use *rename()* to replace the old file with the new file. If a  
126069 crash occurs after the *rename()*, then the file being updated will have either its old contents or its  
126070 new contents on the storage device when the system is rebooted. An application needs to  
126071 synchronize the directory only if it wants to be sure the updated file will have its new contents  
126072 on the storage device.

126073 Some operations, such as *rename()*, can affect more than one directory, whereas synchronization  
126074 calls such as *fsync()* can affect at most one directory at a time. Two calls to *fsync()* may be  
126075 needed after a *rename()* to ensure its durability.

126076 If the file system is inconsistent after a crash it is usually automatically checked and repaired  
126077 when the system is rebooted, or can be repaired manually using a utility such as *fsck*.

126078 If an unrecoverable I/O error occurs when cache is transferred to storage, this standard provides  
126079 no way for applications to discover the error reliably. Implementations are encouraged to report  
126080 such errors on subsequent reads of the storage.

126081 Austin Group Defect 672 is applied, adding this subsection.

#### 126082 **A.4.12 File Times Update**

126083 This section reflects the actions of historical implementations. The times are not updated  
126084 immediately, but are only marked for update by the functions. An implementation may update  
126085 these times immediately.

126086 The accuracy of the time update values is intentionally left unspecified so that systems can  
126087 control the bandwidth of a possible covert channel.

126088 The wording was carefully chosen to make it clear that there is no requirement that the  
126089 conformance document contain information that might incidentally affect file timestamps. Any  
126090 function that performs pathname resolution might update several last data access timestamps.  
126091 Functions such as *getpwnam()* and *getgrnam()* might update the last data access timestamp of  
126092 some specific file or files. It is intended that these are not required to be documented in the  
126093 conformance document, but they should appear in the system documentation.

126094 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0021 [626] is applied.



126095 **A.4.13 Host and Network Byte Order**

126096 There is no additional rationale provided for this section.

126097 **A.4.14 Measurement of Execution Time**

126098 The methods used to measure the execution time of processes and threads, and the precision of  
 126099 these measurements, may vary considerably depending on the software architecture of the  
 126100 implementation, and on the underlying hardware. Implementations can also make tradeoffs  
 126101 between the scheduling overhead and the precision of the execution time measurements.  
 126102 POSIX.1-2024 does not impose any requirement on the accuracy of the execution time; it instead  
 126103 specifies that the measurement mechanism and its precision are implementation-defined.

126104 **A.4.15 Memory Ordering and Synchronization**

126105 Austin Group Defect 1302 is applied, adding the *Memory Ordering* subsection, adapted from the  
 126106 ISO/IEC 9899:2018 standard.

126107 *A.4.15.1 Memory Ordering*

126108 There is no additional rationale provided for this section.

126109 *A.4.15.2 Memory Synchronization*

126110 In older multi-processors, access to memory by the processors was strictly multiplexed. This  
 126111 meant that a processor executing program code interrogates or modifies memory in the order  
 126112 specified by the code and that all the memory operation of all the processors in the system  
 126113 appear to happen in some global order, though the operation histories of different processors are  
 126114 interleaved arbitrarily. The memory operations of such machines are said to be sequentially  
 126115 consistent. In this environment, threads can synchronize using ordinary memory operations. For  
 126116 example, a producer thread and a consumer thread can synchronize access to a circular data  
 126117 buffer as follows:

```

126118 int rdptr = 0;
126119 int wrptr = 0;
126120 data_t buf[BUFSIZE];

126121 Thread 1:
126122     while (work_to_do) {
126123         int next;

126124         buf[wrptr] = produce();
126125         next = (wrptr + 1) % BUFSIZE;
126126         while (rdptr == next)
126127             ;
126128         wrptr = next;
126129     }

126130 Thread 2:
126131     while (work_to_do) {
126132         while (rdptr == wrptr)
126133             ;
126134         consume(buf[rdptr]);
  
```

```

126135         rdptr = (rdptr + 1) % BUFSIZE;
126136     }

```

126137 In modern multi-processors, these conditions are relaxed to achieve greater performance. If one  
 126138 processor stores values in location A and then location B, then other processors loading data  
 126139 from location B and then location A may see the new value of B but the old value of A. The  
 126140 memory operations of such machines are said to be weakly ordered. On these machines, the  
 126141 circular buffer technique shown in the example will fail because the consumer may see the new  
 126142 value of *wrptr* but the old value of the data in the buffer. In such machines, synchronization can  
 126143 only be achieved through the use of special instructions that enforce an order on memory  
 126144 operations. Most high-level language compilers only generate ordinary memory operations to  
 126145 take advantage of the increased performance. They usually cannot determine when memory  
 126146 operation order is important and generate the special ordering instructions. Instead, they rely on  
 126147 the programmer to use synchronization primitives correctly to ensure that modifications to a  
 126148 location in memory are ordered with respect to modifications and/or access to the same location  
 126149 in other threads. Access to read-only data need not be synchronized. The resulting program is  
 126150 said to be data race-free.

126151 Synchronization is still important even when accessing a single primitive variable (for example,  
 126152 an integer). On machines where the integer may not be aligned to the bus data width or be  
 126153 larger than the data width, a single memory load may require multiple memory cycles. This  
 126154 means that it may be possible for some parts of the integer to have an old value while other  
 126155 parts have a newer value. On some processor architectures this cannot happen, but portable  
 126156 programs cannot rely on this.

126157 In summary, a portable multi-threaded program, or a multi-process program that shares  
 126158 writable memory between processes, has to use the synchronization primitives to synchronize  
 126159 data access. It cannot rely on modifications to memory being observed by other threads in the  
 126160 order written in the application or even on modification of a single variable being seen  
 126161 atomically.

126162 Conforming applications may only use the functions listed to synchronize threads of control  
 126163 with respect to memory access. There are many other candidates for functions that might also be  
 126164 used. Examples are: signal sending and reception, or pipe writing and reading. In general, any  
 126165 function that allows one thread of control to wait for an action caused by another thread of  
 126166 control is a candidate. POSIX.1-2024 does not require these additional functions to synchronize  
 126167 memory access since this would imply the following:

- 126168 • All these functions would have to be recognized by advanced compilation systems so that  
 126169 memory operations and calls to these functions are not reordered by optimization.
- 126170 • All these functions would potentially have to have memory synchronization instructions  
 126171 added, depending on the particular machine.
- 126172 • The additional functions complicate the model of how memory is synchronized and make  
 126173 automatic data race detection techniques impractical.

126174 Formal definitions of the memory model were rejected as unreadable by the vast majority of  
 126175 programmers. In addition, most of the formal work in the literature has concentrated on the  
 126176 memory as provided by the hardware as opposed to the application programmer through the  
 126177 compiler and runtime system. It was believed that a simple statement intuitive to most  
 126178 programmers would be most effective. POSIX.1-2024 defines functions that can be used to  
 126179 synchronize access to memory, but it leaves open exactly how one relates those functions to the  
 126180 semantics of each function as specified elsewhere in POSIX.1-2024. POSIX.1-2024 also does not  
 126181 make a formal specification of the partial ordering in time that the functions can impose, as that  
 126182 is implied in the description of the semantics of each function. It simply states that the  
 126183 programmer has to ensure that modifications do not occur “simultaneously” with other access

- 126184 to a memory location.
- 126185 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/4 is applied, adding a new paragraph  
126186 beneath the table of functions: “The *pthread\_once()* function shall synchronize memory for the  
126187 first call in each thread for a given **pthread\_once\_t** object.”.
- 126188 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0022 [863] is applied.
- 126189 Austin Group Defect 1216 is applied, adding *pthread\_cond\_clockwait()*, *pthread\_mutex\_clocklock()*,  
126190 *pthread\_rwlock\_clockrdlock()*, *pthread\_rwlock\_clockwrlock()*, and *sem\_clockwait()* to the list of  
126191 functions that synchronize memory.
- 126192 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.
- 126193 Austin Group Defect 1426 is applied, clarifying under what conditions the functions named in  
126194 this section are required to synchronize memory, and adding *pthread\_mutex\_setprioceiling()* to the  
126195 named functions.
- 126196 Austin Group Defect 1625 is applied, adding *waitid()* to the list of functions that synchronize  
126197 memory with respect to other threads on all successful calls.

#### 126198 **A.4.16 Pathname Resolution**

- 126199 It is necessary to differentiate between the definition of pathname and the concept of pathname  
126200 resolution with respect to the handling of trailing <slash> characters. By specifying the behavior  
126201 here, it is not possible to provide an implementation that is conforming but extends all interfaces  
126202 that handle pathnames to also handle strings that are not legal pathnames (because they have  
126203 trailing <slash> characters).
- 126204 Pathnames that end with one or more trailing <slash> characters must refer to directory paths.  
126205 Earlier versions of this standard were not specific about the distinction between trailing <slash>  
126206 characters on files and directories, and both were permitted.
- 126207 Two types of implementation have been prevalent; those that ignored trailing <slash> characters  
126208 on all pathnames regardless, and those that permitted them only on existing directories.
- 126209 An earlier version of this standard required that a pathname with a trailing <slash> character be  
126210 treated as if it had a trailing " / ." everywhere. This specification was ambiguous. In situations  
126211 where the intent was that the application wanted to require the implementation to accept the  
126212 pathname only if it named a directory (existing or to be created as a result of the call performing  
126213 pathname resolution), literally adding a ' .' after the trailing <slash> could be interpreted to  
126214 require use of that pathname to fail. Some of the uses that created ambiguous requirements  
126215 included *mkdir("newdir/")* and *rmdir("existing-dir/")*. POSIX.1-2024 requires that a pathname  
126216 with a trailing <slash> be rejected unless it refers to a file that is a directory or to a file that is to  
126217 be created as a directory. The *rename()* function and the *mv* utility further specify that a trailing  
126218 <slash> cannot be used on a pathname naming a file that does not exist when used as the last  
126219 argument to *rename()* or *renameat()*, or as the last operand to *mv*.
- 126220 Note that this change does not break any conforming applications; since there were two different  
126221 types of implementation, no application could have portably depended on either behavior. This  
126222 change does however require some implementations to be altered to remain compliant.  
126223 Substantial discussion over a three-year period has shown that the benefits to application  
126224 developers outweighs the disadvantages for some vendors.
- 126225 On a historical note, some early applications automatically appended a '/' to every path.  
126226 Rather than fix the applications, the system implementation was modified to accept this  
126227 behavior by ignoring any trailing <slash>.

126228 Each directory has exactly one parent directory which is represented by the name **dot-dot** in the  
 126229 first directory. No other directory, regardless of linkages established by symbolic links, is  
 126230 considered the parent directory by POSIX.1-2024.

126231 There are two general categories of interfaces involving pathname resolution: those that follow  
 126232 the symbolic link, and those that do not. There are several exceptions to this rule; for example,  
 126233 *open(path,O\_CREAT|O\_EXCL)* will fail when *path* names a symbolic link. However, in all other  
 126234 situations, the *open()* function will follow the link.

126235 What the filename **dot-dot** refers to relative to the root directory is implementation-defined. In  
 126236 Version 7 it refers to the root directory itself; this is the behavior mentioned in POSIX.1-2024. In  
 126237 some networked systems the construction *././hostname/* is used to refer to the root directory of  
 126238 another host, and POSIX.1 permits this behavior.

126239 Other networked systems use the construct *//hostname* for the same purpose; that is, a double  
 126240 initial <slash> is used. There is a potential problem with existing applications that create full  
 126241 pathnames by taking a trunk and a relative pathname and making them into a single string  
 126242 separated by '/', because they can accidentally create networked pathnames when the trunk is  
 126243 '/'. This practice is not prohibited because such applications can be made to conform by  
 126244 simply changing to use "/" as a separator instead of '/':

- 126245 • If the trunk is '/', the full pathname will begin with "/" (the initial '/' and the  
 126246 separator "/"). This is the same as '/', which is what is desired. (This is the general  
 126247 case of making a relative pathname into an absolute one by prefixing with "/" instead  
 126248 of '/'.)
- 126249 • If the trunk is "/A", the result is "/A//..."; since non-leading sequences of two or more  
 126250 <slash> characters are treated as a single <slash>, this is equivalent to the desired  
 126251 "/A/...".
- 126252 • If the trunk is "//A", the implementation-defined semantics will apply. (The multiple  
 126253 <slash> rule would apply.)

126254 Application developers should avoid generating pathnames that start with "//".  
 126255 Implementations are strongly encouraged to avoid using this special interpretation since a  
 126256 number of applications currently do not follow this practice and may inadvertently generate  
 126257 "//...".

126258 The term "root directory" is only defined in POSIX.1 relative to the process. In some  
 126259 implementations, there may be no absolute root directory. The initialization of the root directory  
 126260 of a process is implementation-defined.

126261 When the standard says: "Pathname resolution for a given pathname shall yield the same results  
 126262 when used by any interface in POSIX.1-2024 as long as there are no changes to any files  
 126263 evaluated during pathname resolution for the given pathname between resolutions", this  
 126264 applies to absolute pathnames or to relative pathnames from the same current working  
 126265 directory. Using the same relative pathname from two different working directories may yield  
 126266 different results.

126267 Earlier versions of this standard were unclear as to whether a pathname was required to be a  
 126268 character string or just a string. This standard is now clear that filenames are just strings, and  
 126269 that pathname processing is locale-independent.

126270 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0023 [541,649,825] and  
 126271 XBD/TC2-2008/0024 [825] are applied.

126272 Austin Group Defect 1603 is applied, making a wording improvement related to symbolic links  
 126273 to directories.

126274 **A.4.17 Process ID Reuse**

126275 There is no additional rationale provided for this section.

126276 **A.4.18 Scheduling Policy**

126277 There is no additional rationale provided for this section.

126278 **A.4.19 Seconds Since the Epoch**

126279 Coordinated Universal Time (UTC) includes leap seconds. However, in POSIX time (seconds  
 126280 since the Epoch), leap seconds are ignored (not applied) to provide an easy and compatible  
 126281 method of computing time differences. Broken-down POSIX time is therefore not necessarily  
 126282 UTC, despite its appearance.

126283 As of December 2007, 23 leap seconds had been added to UTC since the Epoch, 1 January, 1970.  
 126284 Historically, one leap second is added every 15 months on average, so this offset can be expected  
 126285 to grow with time.

126286 Most systems' notion of "time" is that of a continuously increasing value, so this value should  
 126287 increase even during leap seconds. However, not only do most systems not keep track of leap  
 126288 seconds, but most systems are probably not synchronized to any standard time reference.  
 126289 Therefore, it is inappropriate to require that a time represented as seconds since the Epoch  
 126290 precisely represent the number of seconds between the referenced time and the Epoch.

126291 It is sufficient to require that applications be allowed to treat this time as if it represented the  
 126292 number of seconds between the referenced time and the Epoch. It is the responsibility of the  
 126293 vendor of the system, and the administrator of the system, to ensure that this value represents  
 126294 the number of seconds between the referenced time and the Epoch as closely as necessary for the  
 126295 application being run on that system.

126296 It is important that the interpretation of time names and seconds since the Epoch values be  
 126297 consistent across conforming systems; that is, it is important that all conforming systems  
 126298 interpret "536 457 599 seconds since the Epoch" as 59 seconds, 59 minutes, 23 hours 31 December  
 126299 1986, regardless of the accuracy of the system's idea of the current time. The expression is given  
 126300 to ensure a consistent interpretation, not to attempt to specify the calendar. The relationship  
 126301 between *tm\_yday* and the day of week, day of month, and month is in accordance with the  
 126302 Gregorian calendar, and so is not specified in POSIX.1.

126303 Consistent interpretation of seconds since the Epoch can be critical to certain types of distributed  
 126304 applications that rely on such timestamps to synchronize events. The accrual of leap seconds in a  
 126305 time standard is not predictable. The number of leap seconds since the Epoch will likely  
 126306 increase. POSIX.1 is more concerned about the synchronization of time between applications of  
 126307 astronomically short duration.

126308 Note that *tm\_yday* is zero-based, not one-based, so the day number in the example above is 364.  
 126309 Note also that the division is an integer division (discarding remainder) as in the C language.

126310 Note also that the meaning of *gmtime()*, *localtime()*, and *mktime()* is specified in terms of this  
 126311 expression. However, the ISO C standard computes *tm\_yday* from *tm\_mday*, *tm\_mon*, and *tm\_year*  
 126312 in *mktime()*. Because it is stated as a (bidirectional) relationship, not a function, and because the  
 126313 conversion between month-day-year and day-of-year dates is presumed well known and is also  
 126314 a relationship, this is not a problem.

126315 The number of seconds since the epoch overflows a signed 32-bit integer in 2038. This standard  
 126316 requires that **time\_t** is an integer type with a width of at least 64 bits (in conforming

126317 programming environments). The requirement that **time\_t** is an integer type is an additional  
 126318 constraint beyond the ISO C standard, which allows a real-floating **time\_t**. Implementation  
 126319 practice has shown that much existing code is unprepared to deal with a floating-point **time\_t**,  
 126320 and that use of **struct timespec** is a more uniform way to provide sub-second time manipulation  
 126321 within applications.

126322 See also [Epoch](#) (on page 3653).

126323 The topic of whether seconds since the Epoch should account for leap seconds has been debated  
 126324 on a number of occasions, and each time consensus was reached (with acknowledged dissent  
 126325 each time) that the majority of users are best served by treating all days identically. (That is, the  
 126326 majority of applications were judged to assume a single length—as measured in seconds since  
 126327 the Epoch—for all days. Thus, leap seconds are not applied to seconds since the Epoch.) Those  
 126328 applications which do care about leap seconds can determine how to handle them in whatever  
 126329 way those applications feel is best. This was particularly emphasized because there was  
 126330 disagreement about what the best way of handling leap seconds might be. It is a practical  
 126331 impossibility to mandate that a conforming implementation must have a fixed relationship to  
 126332 any particular official clock (consider isolated systems, or systems performing “reruns” by  
 126333 setting the clock to some arbitrary time).

126334 Note that as a practical consequence of this, the length of a second as measured by some external  
 126335 standard is not specified. This unspecified second is nominally equal to an International System  
 126336 (SI) second in duration. Applications must be matched to a system that provides the particular  
 126337 handling of external time in the way required by the application.

126338 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/12 is applied, making an editorial  
 126339 correction to the paragraph commencing “How any changes to the value of seconds ...”.

126340 Austin Group Defect 1627 is applied, clarifying that the relationship between the actual date and  
 126341 time in Coordinated Universal Time, as determined by the International Earth Rotation Service,  
 126342 and the system’s current value for seconds since the Epoch is unspecified.

#### 126343 **A.4.20 Semaphore**

126344 Austin Group Defect 502 is applied, clarifying the range of values that an XSI semaphore can  
 126345 have.

126346 Austin Group Defect 1116 is applied, removing a reference to the Semaphores option that existed  
 126347 in earlier versions of this standard.

#### 126348 **A.4.21 Special Device Drivers**

126349 POSIX systems interact with their physical environment using a variety of devices (such as  
 126350 analog-digital converters, digital-analog converters, counters, and video graphic equipment),  
 126351 which provide a set of services that cannot be fully utilized in terms of read and/or write  
 126352 semantics. Traditional practice uses a single function, called *ioctl()*, to encapsulate all the control  
 126353 operations on the different devices connected to the system, both special or common devices.  
 126354 The POSIX.1-1988 standard developers decided not to standardize this interface because it was  
 126355 not type safe, it had a variable number of parameters, and it had behaviors that could not be  
 126356 specified by the standard because they were driver-dependent. Instead, the POSIX.1-1988  
 126357 standard defined a device-specific application program interface (API) for a common class of  
 126358 drivers, Terminals. Later, The Single UNIX Specification, Version 1 included the *ioctl()* function,  
 126359 but restricted it to control of STREAMS devices.

126360 Although the POSIX.1-1988 standard’s solution for common classes of devices is the best from

126361 the point of view of application portability, there is still a need for a way to interact with special,  
126362 or even common devices, for which developing a full standard API is not practical. The device  
126363 control option standardized in POSIX.26 and now included in this standard is a general method  
126364 for interfacing to the widest possible range of devices, through a new service to pass control  
126365 information and commands between the application and the device drivers.

126366 A driver for a special device will normally not be portable between POSIX implementations, but  
126367 an application that uses such a driver can be made portable if all functions calling the driver are  
126368 well defined and standardized. Users and integrators of realtime systems often add drivers for  
126369 special devices, and a standardized function format for interfacing with these devices greatly  
126370 simplifies this process.

126371 Austin Group Defect 729 is applied, adding this subsection.

#### 126372 **A.4.22 Thread-Safety**

126373 Where the interface of a function required by POSIX.1-2024 precludes thread-safety, an alternate  
126374 thread-safe form is provided. The names of these thread-safe forms are the same as the non-  
126375 thread-safe forms with the addition of the suffix ``\_r''. The suffix ``\_r'' is historical, where the  
126376 'r' stood for ``reentrant''.

126377 In some cases, thread-safety is provided by restricting the arguments to an existing function.

126378 See also [Section B.2.9.1](#) (on page 3820).

#### 126379 **A.4.23 Treatment of Error Conditions for Mathematical Functions**

126380 It is intended that undeserved underflow and inexact floating-point exceptions are raised only if  
126381 avoiding them would be too costly.

126382 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0025 [543] is applied.

126383 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

#### 126384 **A.4.24 Treatment of NaN Arguments for Mathematical Functions**

126385 There is no additional rationale provided for this section.

#### 126386 **A.4.25 Utility**

126387 There is no additional rationale provided for this section.

#### 126388 **A.4.26 Variable Assignment**

126389 Austin Group Defect 351 is applied, adding a requirement relating to declaration utilities.

## 126390 A.5 File Format Notation

126391 The notation for spaces allows some flexibility for application output. Note that an empty  
 126392 character position in *format* represents one or more <blank> characters on the output (not *white*  
 126393 *space*, which can include <newline> characters). Therefore, another utility that reads that output  
 126394 as its input must be prepared to parse the data using *scanf()*, *awk*, and so on. The ' $\Delta$ ' character  
 126395 is used when exactly one <space> is output.

126396 The treatment of integers and spaces is different from the *printf()* function in that they can be  
 126397 surrounded with <blank> characters. This was done so that, given a format such as:

```
126398 "%d\n", <foo>
```

126399 the implementation could use a *printf()* call such as:

```
126400 printf("%6d\n", foo);
```

126401 and still conform. This notation is thus somewhat like *scanf()* in addition to *printf()*.

126402 The *printf()* function was chosen as a model because most of the standard developers were  
 126403 familiar with it. One difference from the C function *printf()* is that the l and h conversion  
 126404 specifier characters are not used. As expressed by the Shell and Utilities volume of  
 126405 POSIX.1-2024, there is no differentiation between decimal values for type **int**, type **long**, or type  
 126406 **short**. The conversion specifications %d or %i should be interpreted as an arbitrary length  
 126407 sequence of digits. Also, no distinction is made between single precision and double precision  
 126408 numbers (**float** or **double** in C). These are simply referred to as floating-point numbers.

126409 Many of the output descriptions in the Shell and Utilities volume of POSIX.1-2024 use the term  
 126410 “line”, such as:

```
126411 "%s", <input line>
```

126412 Since the definition of *line* includes the trailing <newline> already, there is no need to include a  
 126413 '\n' in the format; a double <newline> would otherwise result.

126414 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0026 [584] is applied.

126415 Austin Group Defect 1205 is applied, changing the description of the % conversion specifier.

126416 Austin Group Defect 1687 is applied, clarifying the references to <blank> characters to specify  
 126417 they are from the portable character set.

## 126418 A.6 Character Set

### 126419 A.6.1 Portable Character Set

126420 The portable character set is listed in full so there is no dependency on the ISO/IEC 646:1991  
 126421 standard (or historically ASCII) encoded character set, although the set is identical to the  
 126422 characters defined in the International Reference version of the ISO/IEC 646:1991 standard.

126423 POSIX.1-2024 poses no requirement that multiple character sets or codesets be supported,  
 126424 leaving this as a marketing differentiation for implementors. Although multiple charmap files  
 126425 are supported, it is the responsibility of the implementation to provide the file(s); if only one is  
 126426 provided, only that one will be accessible using the *localedef -f* option.

126427 The statement about invariance in codesets for the portable character set is worded to avoid



126428 precluding implementations where multiple incompatible codesets are available (for instance,  
126429 ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if  
126430 they access portable characters that vary on the same implementation.

126431 Not all character sets need include the portable character set, but each locale must include it. For  
126432 example, a Japanese-based locale might be supported by a mixture of character sets: JIS X 0201  
126433 Roman (a Japanese version of the ISO/IEC 646:1991 standard), JIS X 0208, and JIS X 0201  
126434 Katakana. Not all of these character sets include the portable characters, but at least one does  
126435 (JIS X 0201 Roman).

126436 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0027 [584,967] and  
126437 XBD/TC2-2008/0028 [745] are applied.

## 126438 **A.6.2 Character Encoding**

126439 Encoding mechanisms based on single shifts, such as the EUC encoding used in some Asian and  
126440 other countries, can be supported via the current charmap mechanism. With single-shift  
126441 encoding, each character is preceded by a shift code (SS2 or SS3). A complete EUC code,  
126442 consisting of the portable character set (G0) and up to three additional character sets (G1, G2,  
126443 G3), can be described using the current charmap mechanism; the encoding for each character in  
126444 additional character sets G2 and G3 must then include their single-shift code. Other mechanisms  
126445 to support locales based on encoding mechanisms such as locking shift are not addressed by this  
126446 volume of POSIX.1-2024.

126447 The encodings for <slash> and <period> are required to be the same across all locales, in part  
126448 because pathname resolution requires recognition of these bytes. It is a fortunate accident that all  
126449 common shift-based encodings did not use either <slash> or <period> as a valid second byte in  
126450 a multi-byte character.

126451 The encodings for <newline> and <carriage-return> are required to be the same across all  
126452 locales since they are special to the general terminal interface and cannot be changed (see XBD  
126453 [Section 11.1.9](#), on page 203).

126454 Earlier versions of this standard did not state the requirement that the POSIX locale contains 256  
126455 single-byte characters. This was an oversight; the intention was always that the POSIX locale  
126456 should have an 8-bit-clean single-byte encoding.

126457 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0029 [663,967] and  
126458 XBD/TC2-2008/0030 [745] are applied.

## 126459 **A.6.3 C Language Wide-Character Codes**

126460 The standard does not specify how wide characters are encoded or provide a method for  
126461 defining wide characters in a charmap. It specifies ways of translating between wide characters  
126462 and multi-byte characters. The standard does not prevent an extension from providing a method  
126463 to define wide characters.

126464 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/13 is applied, adding a statement that the  
126465 standard has no means of defining a wide-character codeset.

126466 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

126467 **A.6.4 Character Set Description File**

126468 IEEE PASC Interpretation 1003.2 #196 is applied, removing three lines of text dealing with  
 126469 ranges of symbolic names using position constant values which had been erroneously included  
 126470 in the final IEEE P1003.2b draft standard.

126471 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/14 is applied, correcting the example and  
 126472 adding a statement that the standard provides no means of defining a wide-character codeset.

126473 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/15 is applied, allowing the value zero for  
 126474 the width value of **WIDTH** and **WIDTH\_DEFAULT**. This is required to cover some existing  
 126475 locales.

126476 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0031 [967] is applied.

126477 *A.6.4.1 State-Dependent Character Encodings*

126478 A requirement was considered that would force utilities to eliminate any redundant locking  
 126479 shifts, but this was left as a quality of implementation issue.

126480 This change satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex  
 126481 H.1:

126482 *The support of state-dependent (shift encoding) character sets should be addressed fully. See*  
 126483 *descriptions of these in XBD Section 6.2 (on page 120). If such character encodings are supported,*  
 126484 *it is expected that this will impact XBD Section 6.2 (on page 120), Chapter 7 (on page 127),*  
 126485 *Chapter 9 (on page 179), and the comm, cut, diff, grep, head, join, paste, and tail utilities.*

126486 The character set description file provides:

- 126487 • The capability to describe character set attributes (such as collation order or character  
 126488 classes) independent of character set encoding, and using only the characters in the  
 126489 portable character set. This makes it possible to create generic *localedef* source files for all  
 126490 codesets that share the portable character set (such as the ISO 8859 family or IBM Extended  
 126491 ASCII).
- 126492 • Standardized symbolic names for all characters in the portable character set, making it  
 126493 possible to refer to any such character regardless of encoding.

126494 Implementations are free to choose their own symbolic names, as long as the names identified  
 126495 by the Base Definitions volume of POSIX.1-2024 are also defined; this provides support for  
 126496 already existing “character names”.

126497 The names selected for the members of the portable character set follow the  
 126498 ISO/IEC 8859-1:1998 standard and the ISO/IEC 10646:2020 standard. However, several  
 126499 commonly used UNIX system names occur as synonyms in the list:

- 126500 • The historical UNIX system names are used for control characters.
- 126501 • The word “slash” is given in addition to “solidus”.
- 126502 • The word “backslash” is given in addition to “reverse-solidus”.
- 126503 • The word “hyphen” is given in addition to “hyphen-minus”.
- 126504 • The word “period” is given in addition to “full-stop”.
- 126505 • For digits, the word “digit” is eliminated.

- 126506           • For letters, the words “Latin Capital Letter” and “Latin Small Letter” are eliminated.
- 126507           • The words “left brace” and “right brace” are given in addition to “left-curly-bracket” and  
126508           “right-curly-bracket”.
- 126509           • The names of the digits are preferred over the numbers to avoid possible confusion  
126510           between '0' and 'o', and between '1' and 'l' (one and the letter ell).
- 126511           The names for the control characters in XBD Chapter 6 (on page 117) were taken from the  
126512           ISO/IEC 4873:1991 standard.
- 126513           The charmap file was introduced to resolve problems with the portability of, especially, *localedef*  
126514           sources. POSIX.1-2024 assumes that the portable character set is constant across all locales, but  
126515           does not prohibit implementations from supporting two incompatible codings, such as both  
126516           ASCII and EBCDIC. Such dual-support implementations should have all charmaps and *localedef*  
126517           sources encoded using one portable character set, in effect cross-compiling for the other  
126518           environment. Naturally, charmaps (and *localedef* sources) are only portable without  
126519           transformation between systems using the same encodings for the portable character set. They  
126520           can, however, be transformed between two sets using only a subset of the actual characters (the  
126521           portable character set). However, the particular coded character set used for an application or an  
126522           implementation does not necessarily imply different characteristics or collation; on the contrary,  
126523           these attributes should in many cases be identical, regardless of codeset. The charmap provides  
126524           the capability to define a common locale definition for multiple codesets (the same *localedef*  
126525           source can be used for codesets with different extended characters; the ability in the charmap to  
126526           define empty names allows for characters missing in certain codesets).
- 126527           The `<escape_char>` declaration was added at the request of the international community to ease  
126528           the creation of portable charmap files on terminals not implementing the default  
126529           `<backslash>`-escape. The `<comment_char>` declaration was added at the request of the  
126530           international community to eliminate the potential confusion between the `<number-sign>` and  
126531           the hash sign.
- 126532           The octal number notation with no leading zero required was selected to match those of *awk* and  
126533           *tr* and is consistent with that used by *localedef*. To avoid confusion between an octal constant and  
126534           the back-references used in *localedef* source, the octal, hexadecimal, and decimal constants must  
126535           contain at least two digits. As single-digit constants are relatively rare, this should not impose  
126536           any significant hardship. Provision is made for more digits to account for systems in which the  
126537           byte size is larger than 8 bits. For example, a Unicode (ISO/IEC 10646:2020 standard) system  
126538           that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits.
- 126539           The decimal notation is supported because some newer international standards define character  
126540           values in decimal, rather than in the old column/row notation.
- 126541           The charmap identifies the coded character sets supported by an implementation. At least one  
126542           charmap must be provided, but no implementation is required to provide more than one.  
126543           Likewise, implementations can allow users to generate new charmaps (for instance, for a new  
126544           version of the ISO 8859 family of coded character sets), but does not have to do so. If users are  
126545           allowed to create new charmaps, the system documentation describes the rules that apply (for  
126546           instance, “only coded character sets that are supersets of the ISO/IEC 646:1991 standard IRV, no  
126547           multi-byte characters”).
- 126548           This addition of the **WIDTH** specification satisfies the following requirement from the  
126549           ISO POSIX-2:1993 standard, Annex H.1:
- 126550           (9) *The definition of column position relies on the implementation’s knowledge of the integral*  
126551           *width of the characters. The charmap or LC\_CTYPE locale definitions should be enhanced to*  
126552           *allow application specification of these widths.*

126553 The character “width” information was first considered for inclusion under *LC\_CTYPE* but was  
 126554 moved because it is more closely associated with the information in the charmap than  
 126555 information in the locale source (cultural conventions information). Concerns were raised that  
 126556 formalizing this type of information is moving the locale source definition from the codeset-  
 126557 independent entity that it was designed to be to a repository of codeset-specific information. A  
 126558 similar issue occurred with the `<code_set_name>`, `<mb_cur_max>`, and `<mb_cur_min>`  
 126559 information, which was resolved to reside in the charmap definition.

126560 The width definition was added to the IEEE P1003.2b draft standard with the intent that the  
 126561 *wcswidth()* and/or *wcwidth()* functions (currently specified in the System Interfaces volume of  
 126562 POSIX.1-2024) be the mechanism to retrieve the character width information.

## 126563 A.7 Locale

### 126564 A.7.1 General

126565 The description of locales is based on work performed in the UniForum Technical Committee,  
 126566 Subcommittee on Internationalization. Wherever appropriate, keywords are taken from the  
 126567 ISO C standard or the X/Open Portability Guide.

126568 The value used to specify a locale with environment variables is the name specified as the *name*  
 126569 operand to the *localedef* utility when the locale was created. This provides a verifiable method to  
 126570 create and invoke a locale.

126571 The “object” definitions need not be portable, as long as “source” definitions are. Strictly  
 126572 speaking, source definitions are portable only between implementations using the same  
 126573 character set(s). Such source definitions, if they use symbolic names only, easily can be ported  
 126574 between systems using different codesets, as long as the characters in the portable character set  
 126575 (see XBD Section 6.1, on page 117) have common values between the codesets; this is frequently  
 126576 the case in historical implementations. Of source, this requires that the symbolic names used for  
 126577 characters outside the portable character set be identical between character sets. The definition  
 126578 of symbolic names for characters is outside the scope of POSIX.1-2024, but is certainly within the  
 126579 scope of other standards organizations.

126580 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)  
 126581 with the appropriate value. If the function is invoked with an empty string, the value of the  
 126582 corresponding environment variable is used. If the environment variable is not set or is set to the  
 126583 empty string, the implementation sets the appropriate environment as defined in XBD Chapter 8  
 126584 (on page 167).

126585 The locale settings of individual categories cannot be truly independent and still guarantee  
 126586 correct results. For example, when collating two strings, characters must first be extracted from  
 126587 each string (governed by *LC\_CTYPE*) before being mapped to collating elements (governed by  
 126588 *LC\_COLLATE*) for comparison. That is, if *LC\_CTYPE* is causing parsing according to the rules of  
 126589 a large, multi-byte code set (potentially returning 20000 or more distinct character codeset  
 126590 values), but *LC\_COLLATE* is set to handle only an 8-bit codeset with 256 distinct characters,  
 126591 meaningful results are obviously impossible.

126592 Earlier versions of this standard stated that if different character sets are used by the locale  
 126593 categories, the results achieved by an application utilizing these categories are undefined. This  
 126594 was felt to be overly restrictive. For example, when setting:

126595 `LANG=en_US.utf8`

- 126596 LC\_TIME=POSIX
- 126597 on a system where the codeset for the POSIX locale is ASCII and the codeset for en\_US.utf8 is  
126598 UTF-8, all of the characters used in the LC\_TIME locale data exist, with the same encoding, in  
126599 the codeset used for LC\_CTYPE (via LANG), so there is no reason for the behavior to be  
126600 undefined in this case. This standard now has more precise requirements in this area.
- 126601 Austin Group Defect 1122 is applied, adding item 3 to the list of ways to select the locale to be  
126602 used by some C-language functions.
- 126603 Austin Group Defect 1477 is applied, clarifying the behavior when locale categories have  
126604 different character sets.

## 126605 A.7.2 POSIX Locale

- 126606 On POSIX.1 implementations the POSIX locale is equal to the C locale, even though the  
126607 requirements for the POSIX locale are more extensive than the ISO C standard requirements for  
126608 the C locale. To avoid being classified as a C-language function, the name has been changed to  
126609 the POSIX locale; the environment variable value can be either "POSIX" or, for historical  
126610 reasons, "C".
- 126611 The POSIX definitions mirror the historical UNIX system behavior.
- 126612 The use of symbolic names for characters in the tables does not imply that the POSIX locale must  
126613 be described using symbolic character names, but merely that it may be advantageous to do so.
- 126614 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0032 [796] and XBD/TC2-2008/0033  
126615 [663] are applied.

## 126616 A.7.3 Locale Definition

- 126617 The decision to separate the file format from the *localedef* utility description was only partially  
126618 editorial. Implementations may provide other interfaces than *localedef*. Requirements on “the  
126619 utility”, mostly concerning error messages, are described in this way because they are meant to  
126620 affect the other interfaces implementations may provide as well as *localedef*.
- 126621 The text about POSIX2\_LOCALEDEF does not mean that internationalization is optional; only  
126622 that the functionality of the *localedef* utility is. REs, for instance, must still be able to recognize,  
126623 for example, character class expressions such as "[[:alpha:]]". A possible analogy is with  
126624 an applications development environment; while all conforming implementations must be  
126625 capable of executing applications, not all need to have the development environment installed.  
126626 The assumption is that the capability to modify the behavior of utilities (and applications) via  
126627 locale settings must be supported. If the *localedef* utility is not present, then the only choice is to  
126628 select an existing (presumably implementation-documented) locale. An implementation could,  
126629 for example, choose to support only the POSIX locale, which would in effect limit the amount of  
126630 changes from historical implementations quite drastically. The *localedef* utility is still required,  
126631 but would always terminate with an exit code indicating that no locale could be created.  
126632 Supported locales must be documented using the syntax defined in this chapter. (This ensures  
126633 that users can accurately determine what capabilities are provided. If the implementation  
126634 decides to provide additional capabilities to the ones in this chapter, that is already provided  
126635 for.)
- 126636 If the option is present (that is, locales can be created), then the *localedef* utility must be capable  
126637 of creating locales based on the syntax and rules defined in this chapter. This does not mean that  
126638 the implementation cannot also provide alternate means for creating locales.

126639 The octal, decimal, and hexadecimal notations are the same employed by the charmap facility  
 126640 (see XBD Section 6.4, on page 121). To avoid confusion between an octal constant and a back-  
 126641 reference, the octal, hexadecimal, and decimal constants must contain at least two digits. As  
 126642 single-digit constants are relatively rare, this should not impose any significant hardship.  
 126643 Provision is made for more digits to account for systems in which the byte size is larger than 8  
 126644 bits. For example, a Unicode (see the ISO/IEC 10646:2020 standard) system that has defined  
 126645 16-bit bytes may require six octal, four hexadecimal, and five decimal digits. As with the  
 126646 charmap file, multi-byte characters are described in the locale definition file using “big-endian”  
 126647 notation for reasons of portability. There is no requirement that the internal representation in the  
 126648 computer memory be in this same order.

126649 One of the guidelines used for the development of this volume of POSIX.1-2024 is that  
 126650 characters outside the invariant part of the ISO/IEC 646:1991 standard should not be used in  
 126651 portable specifications. The <backslash> character is not in the invariant part; the <number-  
 126652 sign> is, but with multiple representations: as a <number-sign>, and as a hash sign. As far as  
 126653 general usage of these symbols, they are covered by the “grandfather clause”, but for newly  
 126654 defined interfaces, the WG15 POSIX working group has requested that POSIX provide alternate  
 126655 representations. Consequently, while the default escape character remains the <backslash> and  
 126656 the default comment character is the <number-sign>, implementations are required to recognize  
 126657 alternative representations, identified in the applicable source file via the <escape\_char> and  
 126658 <comment\_char> keywords.

#### 126659 A.7.3.1 LC\_CTYPE

126660 The *LC\_CTYPE* category is primarily used to define the encoding-independent aspects of a  
 126661 character set, such as character classification. In addition, certain encoding-dependent  
 126662 characteristics are also defined for an application via the *LC\_CTYPE* category. POSIX.1-2024 does  
 126663 not mandate that the encoding used in the locale is the same as the one used by the application  
 126664 because an implementation may decide that it is advantageous to define locales in a system-  
 126665 wide encoding rather than having multiple, logically identical locales in different encodings, and  
 126666 to convert from the application encoding to the system-wide encoding on usage. Other  
 126667 implementations could require encoding-dependent locales.

126668 In either case, the *LC\_CTYPE* attributes that are directly dependent on the encoding, such as  
 126669 <mb\_cur\_max> and the display width of characters, are not user-specifiable in a locale source  
 126670 and are consequently not defined as keywords.

126671 Implementations may define additional keywords or extend the *LC\_CTYPE* mechanism to allow  
 126672 application-defined keywords.

126673 The text “The ellipsis specification shall only be valid within a single encoded character set” is  
 126674 present because it is possible to have a locale supported by multiple character encodings, as  
 126675 explained in the rationale for XBD Section 6.1 (on page 117). An example given there is of a  
 126676 possible Japanese-based locale supported by a mixture of the character sets JIS X 0201 Roman,  
 126677 JIS X 0208, and JIS X 0201 Katakana. Attempting to express a range of characters across these sets  
 126678 is not logical and the implementation is free to reject such attempts.

126679 As the *LC\_CTYPE* character classes are based on the ISO C standard character class definition,  
 126680 the category does not support multi-character elements. For instance, the German character  
 126681 <sharp-s> is traditionally classified as a lowercase letter. There is no corresponding uppercase  
 126682 letter; in proper capitalization of German text, the <sharp-s> will be replaced by "SS"; that is, by  
 126683 two characters. This kind of conversion is outside the scope of the **toupper** and **tolower**  
 126684 keywords.

126685 Where POSIX.1-2024 specifies that only certain characters can be specified, as for the keywords  
 126686 **digit** and **xdigit**, the specified characters must be from the portable character set, as shown. As

- 126687 an example, only the Arabic digits 0 through 9 are acceptable as digits.
- 126688 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included  
126689 characters. These only need to be specified if the character values (that is, encoding) differs from  
126690 the implementation default values. It is not possible to define a locale without these  
126691 automatically included characters unless some implementation extension is used to prevent  
126692 their inclusion. Such a definition would not be a proper superset of the C locale, and thus, it  
126693 might not be possible for the standard utilities to be implemented as programs conforming to  
126694 the ISO C standard.
- 126695 The definition of character class **digit** requires that only ten characters—the ones defining  
126696 digits—can be specified; alternate digits (for example, Hindi or Kanji) cannot be specified here.  
126697 However, the encoding may vary if an implementation supports more than one encoding.
- 126698 The definition of character class **xdigit** requires that the characters included in character class  
126699 **digit** are included here also and allows for different symbols for the hexadecimal digits 10  
126700 through 15.
- 126701 The inclusion of the **charclass** keyword satisfies the following requirement from the  
126702 ISO POSIX-2: 1993 standard, Annex H.1:
- 126703 (3) *The LC\_CTYPE (2.5.2.1) locale definition should be enhanced to allow user-specified additional*  
126704 *character classes, similar in concept to the ISO C standard Multibyte Support Extension (MSE)*  
126705 *iswctype() function.*
- 126706 This keyword was previously included in The Open Group specifications and is now mandated  
126707 in the Shell and Utilities volume of POSIX.1-2024.
- 126708 The symbolic constant {CHARCLASS\_NAME\_MAX} was also adopted from The Open Group  
126709 specifications. Applications portability is enhanced by the use of symbolic constants.
- 126710 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0033 [663], XBD/TC2-2008/0034 [663],  
126711 XBD/TC2-2008/0035 [584], and XBD/TC2-2008/0036 [584] are applied.
- 126712 Austin Group Defect 1078 is applied, clarifying that only the specified set of characters can be  
126713 included in the **digit** and **xdigit** classes in all locales, not just the POSIX locale.
- 126714 Austin Group Defect 1589 is applied, disallowing some characters from being included in the  
126715 **blank** class.
- 126716 A.7.3.2 LC\_COLLATE
- 126717 The rules governing collation depend to some extent on the use. At least five different levels of  
126718 increasingly complex collation rules can be distinguished:
- 126719 1. *Byte/machine code order*: This is the historical collation order in the UNIX system and many  
126720 proprietary operating systems. Collation is here performed character by character,  
126721 without any regard to context. The primary virtue is that it usually is quite fast and also  
126722 completely deterministic; it works well when the native machine collation sequence  
126723 matches the user expectations.
  - 126724 2. *Character order*: On this level, collation is also performed character by character, without  
126725 regard to context. The order between characters is, however, not determined by the code  
126726 values, but on the expectations by the user of the “correct” order between characters. In  
126727 addition, such a (simple) collation order can specify that certain characters collate equally  
126728 (for example, uppercase and lowercase letters).

- 126729 3. *String ordering*: On this level, entire strings are compared based on relatively  
 126730 straightforward rules. Several “passes” may be required to determine the order between  
 126731 two strings. Characters may be ignored in some passes, but not in others; the strings may  
 126732 be compared in different directions; and simple string substitutions may be performed  
 126733 before strings are compared. This level is best described as “dictionary” ordering; it is  
 126734 based on the spelling, not the pronunciation, or meaning, of the words.
- 126735 4. *Text search ordering*: This is a further refinement of the previous level, best described as  
 126736 “telephone book ordering”; some common homonyms (words spelled differently but  
 126737 with the same pronunciation) are collated together; numbers are collated as if they were  
 126738 spelled out, and so on.
- 126739 5. *Semantic-level ordering*: Words and strings are collated based on their meaning; entire  
 126740 words (such as “the”) are eliminated; the ordering is not deterministic. This usually  
 126741 requires special software and is highly dependent on the intended use.

126742 While the historical collation order formally is at level 1, for the English language it corresponds  
 126743 roughly to elements at level 2. The user expects to see the output from the *ls* utility sorted very  
 126744 much as it would be in a dictionary. While telephone book ordering would be an optimal goal  
 126745 for standard collation, this was ruled out as the order would be language-dependent.  
 126746 Furthermore, a requirement was that the order must be determined solely from the text string  
 126747 and the collation rules; no external information (for example, “pronunciation dictionaries”)  
 126748 could be required.

126749 As a result, the goal for the collation support is at level 3. This also matches the requirements for  
 126750 the Canadian collation order, as well as other, known collation requirements for alphabetic  
 126751 scripts. It specifically rules out collation based on pronunciation rules or based on semantic  
 126752 analysis of the text.

126753 The syntax for the *LC\_COLLATE* category source meets the requirements for level 3 and has  
 126754 been verified to produce the correct result with examples based on French, Canadian, and  
 126755 Danish collation order. Because it supports multi-character collating elements, it is also capable  
 126756 of supporting collation in codesets where a character is expressed using non-spacing characters  
 126757 followed by the base character (such as the ISO/IEC 6937:2001 standard).

126758 The directives that can be specified in an operand to the **order\_start** keyword are based on the  
 126759 requirements specified in several proposed standards and in customary use. The following is a  
 126760 rephrasing of rules defined for “lexical ordering in English and French” by the Canadian  
 126761 Standards Association (the text in square brackets is rephrased):

- 126762 • Once special characters [punctuation] have been removed from original strings, the  
 126763 ordering is determined by scanning forwards (left to right) [disregarding case and  
 126764 diacriticals].
- 126765 • In case of equivalence, special characters are once again removed from original strings and  
 126766 the ordering is determined by scanning backwards (starting from the rightmost character  
 126767 of the string and back), character by character [disregarding case but considering  
 126768 diacriticals].
- 126769 • In case of repeated equivalence, special characters are removed again from original strings  
 126770 and the ordering is determined by scanning forwards, character by character [considering  
 126771 both case and diacriticals].
- 126772 • If there is still an ordering equivalence after the first three rules have been applied, then  
 126773 only special characters and the position they occupy in the string are considered to  
 126774 determine ordering. The string that has a special character in the lowest position comes  
 126775 first. If two strings have a special character in the same position, the character [with the  
 126776 lowest collation value] comes first. In case of equality, the other special characters are



- 126777 considered until there is a difference or until all special characters have been exhausted.
- 126778 It is estimated that this part of POSIX.1-2024 covers the requirements for all European  
126779 languages, and no particular problems are anticipated with Slavic or Middle East character sets.
- 126780 The Far East (particularly Japanese/Chinese) collations are often based on contextual  
126781 information and pronunciation rules (the same ideogram can have different meanings and  
126782 different pronunciations). Such collation, in general, falls outside the desired goal of  
126783 POSIX.1-2024. There are, however, several other collation rules (stroke/radical or “most  
126784 common pronunciation”) that can be supported with the mechanism described here.
- 126785 The character order is defined by the order in which characters and elements are specified  
126786 between the **order\_start** and **order\_end** keywords. Weights assigned to the characters and  
126787 elements define the collation sequence; in the absence of weights, the character order is also the  
126788 collation sequence.
- 126789 The **position** keyword provides the capability to consider, in a compare, the relative position of  
126790 characters not subject to **IGNORE**. As an example, consider the two strings "o-ring" and  
126791 "or-ing". Assuming the <hyphen-minus> is subject to **IGNORE** on the first pass, the two  
126792 strings compare equal, and the position of the <hyphen-minus> is immaterial. On second pass,  
126793 all characters except the <hyphen-minus> are subject to **IGNORE**, and in the normal case the  
126794 two strings would again compare equal. By taking position into account, the first collates before  
126795 the second.
- 126796 This standard requires that all implementation-provided locales define a collation sequence that  
126797 has a total ordering of all characters unless the locale name has an '@' modifier indicating that  
126798 it has a special collation sequence. Defining locales in this way eliminates unexpected behavior  
126799 when non-identical strings can collate equally (for example, `sort -u` and `sort | uniq` are  
126800 not equivalent). The exception for locales with a suitable '@' modifier in the name allows  
126801 implementations to supply locales which do not have a total ordering of all characters provided  
126802 that they draw attention to it in the modifier name. For example, `@icase` could indicate that  
126803 each upper and lowercase character pair collates equally. Even with an '@' modifier, total  
126804 ordering is preferred when possible; for example, characters that are “ignored” in dictionary  
126805 order need not be completely ignored (by using **IGNORE** for all collation weights), but can  
126806 instead be given a unique weight after one or more **IGNORE** weights.
- 126807 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0037 [938], XBD/TC2-2008/0038 [663],  
126808 and XBD/TC2-2008/0039 [584] are applied.
- 126809 Austin Group Defect 948 is applied, requiring that all implementation-provided locales define a  
126810 collation sequence that has a total ordering of all characters unless the locale name has an '@'  
126811 modifier indicating that it has a special collation sequence.
- 126812 Austin Group Defect 1740 is applied, noting that it is the responsibility of the locale writer to  
126813 ensure <NUL> has the lowest primary weight in a collation ordering.
- 126814 A.7.3.3 *LC\_MONETARY*
- 126815 The currency symbol does not appear in *LC\_MONETARY* because it is not defined in the C  
126816 locale of the ISO C standard.
- 126817 The ISO C standard limits the size of decimal points and thousands delimiters to single-byte  
126818 values. In locales based on multi-byte coded character sets, this cannot be enforced;  
126819 POSIX.1-2024 does not prohibit such characters, but makes the behavior unspecified (in the text  
126820 “In contexts where other standards ...”).
- 126821 The grouping specification is based on, but not identical to, the ISO C standard. The -1 indicates  
126822 that no further grouping is performed; the equivalent of {CHAR\_MAX} in the ISO C standard.

126823 The text “the value is not available in the locale” is taken from the ISO C standard and is used  
 126824 instead of the “unspecified” text in early proposals. There is no implication that omitting these  
 126825 keywords or assigning them values of " " or -1 produces unspecified results; such omissions or  
 126826 assignments eliminate the effects described for the keyword or produce zero-length strings, as  
 126827 appropriate.

126828 The locale definition is an extension of the ISO C standard *localeconv()* specification. In  
 126829 particular, rules on how **currency\_symbol** is treated are extended to also cover **int\_curr\_symbol**,  
 126830 and **p\_sep\_by\_space** and **n\_sep\_by\_space** have been augmented with the value 2, which places  
 126831 a <space> between the sign and the symbol. This has been updated to match the  
 126832 ISO/IEC 9899:1999 standard requirements and is an incompatible change from UNIX 98 and the  
 126833 ISO POSIX-2 standard and the ISO POSIX-1:1996 standard requirements. The following table  
 126834 shows the result of various combinations:

|                          |                        | <b>p_sep_by_space</b> |           |          |
|--------------------------|------------------------|-----------------------|-----------|----------|
|                          |                        | <b>2</b>              | <b>1</b>  | <b>0</b> |
| <b>p_cs_precedes = 1</b> | <b>p_sign_posn = 0</b> | (\$1.25)              | (\$ 1.25) | (\$1.25) |
|                          | <b>p_sign_posn = 1</b> | + \$1.25              | +\$ 1.25  | +\$1.25  |
|                          | <b>p_sign_posn = 2</b> | \$1.25 +              | \$ 1.25+  | \$1.25+  |
|                          | <b>p_sign_posn = 3</b> | + \$1.25              | +\$ 1.25  | +\$1.25  |
| <b>p_cs_precedes = 0</b> | <b>p_sign_posn = 4</b> | \$ +1.25              | \$+ 1.25  | \$+1.25  |
|                          | <b>p_sign_posn = 0</b> | (1.25 \$)             | (1.25 \$) | (1.25\$) |
|                          | <b>p_sign_posn = 1</b> | +1.25 \$              | +1.25 \$  | +1.25\$  |
|                          | <b>p_sign_posn = 2</b> | 1.25\$ +              | 1.25 \$+  | 1.25\$+  |
|                          | <b>p_sign_posn = 3</b> | 1.25+ \$              | 1.25 +\$  | 1.25+\$  |
|                          | <b>p_sign_posn = 4</b> | 1.25\$ +              | 1.25 \$+  | 1.25\$+  |

126847 The following is an example of the interpretation of the **mon\_grouping** keyword. Assuming that  
 126848 the value to be formatted is 123456789 and the **mon\_thousands\_sep** is <apostrophe>, then the  
 126849 following table shows the result. The third column shows the equivalent string in the ISO C  
 126850 standard that would be used by the *localeconv()* function to accommodate this grouping.

| <b>mon_grouping</b> | <b>Formatted Value</b> | <b>ISO C String</b> |
|---------------------|------------------------|---------------------|
| 3;-1                | 123456'789             | "\3\177"            |
| 3                   | 123'456'789            | "\3"                |
| 3;2;-1              | 1234'56'789            | "\3\2\177"          |
| 3;2                 | 12'34'56'789           | "\3\2"              |
| -1                  | 123456789              | "\177"              |

126857 In these examples, the octal value of {CHAR\_MAX} is 177.

126858 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/6 adds a correction that permits the Euro  
 126859 currency symbol and addresses extensibility. The correction is stated using the term “should”  
 126860 intentionally, in order to make this a recommendation rather than a restriction on  
 126861 implementations. This allows for flexibility in implementations on how they handle future  
 126862 currency symbol additions.

126863 IEEE Std 1003.1-2001/Cor 1-2002, tem XBD/TC1/D6/5 is applied, adding the **int\_[np]\_\*** values  
 126864 to the POSIX locale definition of *LC\_MONETARY*.

126865 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/16 is applied, updating the descriptions  
 126866 of **p\_sep\_by\_space**, **n\_sep\_by\_space**, **int\_p\_sep\_by\_space**, and **int\_n\_sep\_by\_space** to match  
 126867 the description of these keywords in the ISO C standard and the System Interfaces volume of  
 126868 POSIX.1-2024, *localeconv()*.

126869 Austin Group Defect 1199 is applied, adding a requirement that *localedef* does not accept certain

- 126870 combinations of **\*\_sign\_posn**, **positive\_sign**, and **negative\_sign** values.
- 126871 Austin Group Defect 1241 is applied, clarifying the meaning of empty string values.
- 126872 A.7.3.4 *LC\_NUMERIC*
- 126873 See the rationale for *LC\_MONETARY* for a description of the behavior of grouping.
- 126874 Austin Group Defect 1241 is applied, clarifying the meaning of empty string values.
- 126875 A.7.3.5 *LC\_TIME*
- 126876 Although certain of the conversion specifications in the POSIX locale (such as the name of the  
126877 month) are shown with initial capital letters, this need not be the case in other locales. Programs  
126878 using these conversion specifications may need to adjust the capitalization if the output is going  
126879 to be used at the beginning of a sentence.
- 126880 The *LC\_TIME* descriptions of **abday**, **day**, **mon**, and **abmon** imply a Gregorian style calendar  
126881 (7-day weeks, 12-month years, leap years, and so on). Formatting time strings for other types of  
126882 calendars is outside the scope of POSIX.1-2024.
- 126883 While the ISO 8601-1:2019 standard numbers the weekdays starting with Monday, historical  
126884 practice is to use the Sunday as the first day. Rather than change the order and introduce  
126885 potential confusion, the days must be specified beginning with Sunday; previous references to  
126886 “first day” have been removed. Note also that the Shell and Utilities volume of POSIX.1-2024  
126887 *date* utility supports numbering compliant with the ISO 8601-1:2019 standard.
- 126888 As specified under *date* in the Shell and Utilities volume of POSIX.1-2024 and *strftime()* in the  
126889 System Interfaces volume of POSIX.1-2024, the conversion specifications corresponding to the  
126890 optional keywords consist of a modifier followed by a traditional conversion specification (for  
126891 instance, %Ex). If the optional keywords are not supported by the implementation or are  
126892 unspecified for the current locale, these modified conversion specifications are treated as the  
126893 traditional conversion specifications. For example, assume the following keywords:
- 126894 alt\_digits "0th"; "1st"; "2nd"; "3rd"; "4th"; "5th"; \  
126895 "6th"; "7th"; "8th"; "9th"; "10th"
- 126896 d\_fmt "The %Od day of %B in %Y"
- 126897 On July 4th 1776, the %x conversion specifications would result in "The 4th day of July  
126898 in 1776", while on July 14th 1789 it would result in "The 14 day of July in 1789". It  
126899 can be noted that the above example is for illustrative purposes only; the %O modifier is  
126900 primarily intended to provide for Kanji or Hindi digits in *date* formats.
- 126901 The following is an example for Japan that supports the current plus last three Emperors and  
126902 reverts to Western style numbering for years prior to the Meiji era. The example also allows for  
126903 the custom of using a special name for the first year of an era instead of using 1. (The examples  
126904 substitute romaji where kanji should be used.)
- 126905 era\_d\_fmt "%EY%mgatsu%dnichi (%a) "
- 126906 era "+:2:1990/01/01:+\*:Heisei:%EC%Eynen"; \  
126907 "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen"; \  
126908 "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eynen"; \  
126909 "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen"; \  
126910 "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eynen"; \  
126911 "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen"; \  
126912 "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eynen"; \  
126913 "

126913 "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\  
 126914 "-:1868:1868/09/07:-\*::%Ey"

126915 Assuming that the current date is September 21, 1991, a request to *date* or *strftime()* would yield  
 126916 the following results:

126917 %Ec - Heisei3nen9gatsu21nichi (Sat) 14:39:26  
 126918 %EC - Heisei  
 126919 %Ex - Heisei3nen9gatsu21nichi (Sat)  
 126920 %Ey - 3  
 126921 %EY - Heisei3nen

126922 Example era definitions for the Republic of China:

126923 era "+:2:1913/01/01:+\*:ChungHwaMingGuo:%EC%EyNen";\  
 126924 "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%ECYuenNen";\  
 126925 "+:1:1911/12/31:-\*:MingChien:%EC%EyNen"

126926 Example definitions for the Christian Era:

126927 era "+:1:0001/01/01:+\*:AD:%EC %Ey";\  
 126928 "+:1:-0001/12/31:-\*:BC:%Ey %EC"

126929 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0040 [912] is applied.

126930 Austin Group Defects 258 and 1166 are applied, adding the **alt\_mon** and **ab\_alt\_mon** locale  
 126931 keywords.

126932 Austin Group Defect 1307 is applied, changing the **am\_pm** and **t\_fmt\_ampm** keywords and the  
 126933 AM\_STR, PM\_STR, and T\_FMT\_AMPM constants in relation to locales that do not support the  
 126934 12-hour clock format.

#### 126935 A.7.3.6 LC\_MESSAGES

126936 The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly  
 126937 used to match user affirmative and negative responses. In POSIX.1-2024, the **yesexpr**, **noexpr**,  
 126938 YESEXPR, and NOEXPR extended regular expressions have replaced them. Applications  
 126939 should use the general locale-based messaging facilities to issue prompting messages which  
 126940 include sample desired responses.

126941 Affirmative responses like:

126942 y  
 126943 Yes  
 126944 Yes!

126945 and negative responses like:

126946 N  
 126947 No  
 126948 Never  
 126949 No way!

126950 should all be recognized as affirmative and negative responses, respectively, by the EREs  
 126951 identified by the **yesexpr** and **noexpr** keywords for English language-based locales. There is no  
 126952 requirement that multi-line responses nor ambiguous responses like:

126953 no or yes  
 126954 yes or no  
 126955 maybe

126956 be correctly classified by either of these EREs. Application writers are encouraged to include  
126957 locale-specific suggestions for affirmative and negative responses in prompts.

#### 126958 **A.7.4 Locale Definition Grammar**

126959 There is no additional rationale provided for this section.

##### 126960 *A.7.4.1 Locale Lexical Conventions*

126961 There is no additional rationale provided for this section.

##### 126962 *A.7.4.2 Locale Grammar*

126963 Austin Group Defects 258 and 1166 are applied, adding the **alt\_mon** and **ab\_alt\_mon** locale  
126964 keywords.

#### 126965 **A.7.5 Locale Definition Example**

126966 The following is an example of a locale definition file that could be used as input to the *localedef*  
126967 utility. It assumes that the utility is executed with the `-f` option, naming a charmap file with (at  
126968 least) the following content:

```
126969 CHARMAP
126970 <space>      \x20
126971 <dollar>     \x24
126972 <A>          \101
126973 <a>          \141
126974 <A-acute>    \346
126975 <a-acute>    \365
126976 <A-grave>    \300
126977 <a-grave>    \366
126978 <b>          \142
126979 <C>          \103
126980 <c>          \143
126981 <c-cedilla>  \347
126982 <d>          \x64
126983 <H>          \110
126984 <h>          \150
126985 <eszet>     \xb7
126986 <s>          \x73
126987 <z>          \x7a
126988 END CHARMAP
```

126989 It should not be taken as complete or to represent any actual locale, but only to illustrate the  
126990 syntax.

```
126991 #
126992 LC_CTYPE
126993 lower  <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
126994 upper  A;B;C;Ç;...;Z
126995 space  \x20;\x09;\x0a;\x0b;\x0c;\x0d
126996 blank  \040;\011
```

```

126997 toupper (<a>,<A>); (b,B); (c,C); (ç,Ç); (d,D); (z,Z)
126998 END LC_CTYPE
126999 #
127000 LC_COLLATE
127001 #
127002 # The following example of collation is based on
127003 # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
127004 # Ordering Standard for Character Sets of CSA Z234.4 Standard".
127005 # (Other parts of this example locale definition file do not
127006 # purport to relate to Canada, or to any other real culture.)
127007 # The proposed standard defines a 4-weight collation, such that
127008 # in the first pass, characters are compared without regard to
127009 # case or accents; in the second pass, backwards-compare without
127010 # regard to case; in the third pass, forwards-compare without
127011 # regard to diacriticals. In the 3 first passes, non-alphabetic
127012 # characters are ignored; in the fourth pass, only special
127013 # characters are considered, such that "The string that has a
127014 # special character in the lowest position comes first. If two
127015 # strings have a special character in the same position, the
127016 # collation value of the special character determines ordering.
127017 #
127018 # Only a subset of the character set is used here; mostly to
127019 # illustrate the set-up.
127020 #
127021 collating-symbol <NULL>
127022 collating-symbol <LOW_VALUE>
127023 collating-symbol <LOWER-CASE>
127024 collating-symbol <SUBSCRIPT-LOWER>
127025 collating-symbol <SUPERSCRIPT-LOWER>
127026 collating-symbol <UPPER-CASE>
127027 collating-symbol <NO-ACCENT>
127028 collating-symbol <PECULIAR>
127029 collating-symbol <LIGATURE>
127030 collating-symbol <ACUTE>
127031 collating-symbol <GRAVE>
127032 # Further collating-symbols follow.
127033 #
127034 # Properly, the standard does not include any multi-character
127035 # collating elements; the one below is added for completeness.
127036 #
127037 collating_element <ch> from "<c><h>"
127038 collating_element <CH> from "<C><H>"
127039 collating_element <Ch> from "<C><h>"
127040 #
127041 order_start forward;backward;forward;forward,position
127042 #
127043 # Collating symbols are specified first in the sequence to allocate
127044 # basic collation values to them, lower than that of any character.
127045 <NULL>
127046 <LOW_VALUE>
127047 <LOWER-CASE>
127048 <SUBSCRIPT-LOWER>
127049 <SUPERSCRIPT-LOWER>

```

```

127050 <UPPER-CASE>
127051 <NO-ACCENT>
127052 <PECULIAR>
127053 <LIGATURE>
127054 <ACUTE>
127055 <GRAVE>
127056 <RING-ABOVE>
127057 <DIAERESIS>
127058 <TILDE>
127059 # Further collating symbols are given a basic collating value here.
127060 #
127061 # Here follow special characters.
127062 <space> IGNORE; IGNORE; IGNORE; <space>
127063 # Other special characters follow here.
127064 #
127065 # Here follow the regular characters.
127066 <a> <a>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
127067 <A> <a>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
127068 <a-acute> <a>; <ACUTE>; <LOWER-CASE>; IGNORE
127069 <A-acute> <a>; <ACUTE>; <UPPER-CASE>; IGNORE
127070 <a-grave> <a>; <GRAVE>; <LOWER-CASE>; IGNORE
127071 <A-grave> <a>; <GRAVE>; <UPPER-CASE>; IGNORE
127072 <ae> "<a><e>"; "<LIGATURE><LIGATURE>"; \
127073 "<LOWER-CASE><LOWER-CASE>"; IGNORE
127074 <AE> "<a><e>"; "<LIGATURE><LIGATURE>"; \
127075 "<UPPER-CASE><UPPER-CASE>"; IGNORE
127076 <b> <b>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
127077 <B> <b>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
127078 <c> <c>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
127079 <C> <c>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
127080 <ch> <ch>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
127081 <Ch> <ch>; <NO-ACCENT>; <PECULIAR>; IGNORE
127082 <CH> <ch>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
127083 #
127084 # As an example, the strings "Bach" and "bach" could be encoded (for
127085 # compare purposes) as:
127086 # "Bach" <b>; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; <NO_ACCENT>; \
127087 # <NO_ACCENT>; <LOW_VALUE>; <UPPER-CASE>; <LOWER-CASE>; \
127088 # <LOWER-CASE>; <NULL>
127089 # "bach" <b>; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; <NO_ACCENT>; \
127090 # <NO_ACCENT>; <LOW_VALUE>; <LOWER-CASE>; <LOWER-CASE>; \
127091 # <LOWER-CASE>; <NULL>
127092 #
127093 # The two strings are equal in pass 1 and 2, but differ in pass 3.
127094 #
127095 # Further characters follow.
127096 #
127097 UNDEFINED IGNORE; IGNORE; IGNORE; IGNORE
127098 #
127099 order_end
127100 #
127101 END LC_COLLATE
127102 #

```

```

127103 LC_MONETARY
127104 int_curr_symbol "USD "
127105 currency_symbol "$"
127106 mon_decimal_point "."
127107 mon_grouping 3;0
127108 positive_sign ""
127109 negative_sign "-"
127110 p_cs_precedes 1
127111 n_sign_posn 0
127112 END LC_MONETARY
127113 #
127114 LC_NUMERIC
127115 copy "US_en.ASCII"
127116 END LC_NUMERIC
127117 #
127118 LC_TIME
127119 abday "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
127120 #
127121 day "Sunday";"Monday";"Tuesday";"Wednesday";\
127122 "Thursday";"Friday";"Saturday"
127123 #
127124 abmon "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
127125 "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
127126 #
127127 mon "January";"February";"March";"April";\
127128 "May";"June";"July";"August";"September";\
127129 "October";"November";"December"
127130 #
127131 d_t_fmt "%a %b %d %T %Z %Y\n"
127132 END LC_TIME
127133 #
127134 LC_MESSAGES
127135 yesexpr "^( [yY] [[:alpha:]]* ) | (OK) "
127136 #
127137 noexpr "^[nN] [[:alpha:]]*"
127138 END LC_MESSAGES

```

## 127139 A.8 Environment Variables

### 127140 A.8.1 Environment Variable Definition

127141 The variable *environ* is not intended to be declared in any header, but rather to be declared by  
 127142 the user for accessing the array of strings that is the environment. This is the traditional usage of  
 127143 the symbol. Putting it into a header could break some programs that use the symbol for their  
 127144 own purposes.

127145 The decision to restrict conforming systems to the use of digits, uppercase letters, and  
 127146 underscores for environment variable names allows applications to use lowercase letters in their  
 127147 environment variable names without conflicting with any conforming system.



127148 In addition to the obvious conflict with the shell syntax for positional parameter substitution,  
 127149 some historical applications (including some shells) exclude names with leading digits from the  
 127150 environment.

127151 Some historical implementations removed certain environment variables during program  
 127152 startup when security criteria were not met, instead of just ignoring them at the point of use. The  
 127153 standard developers decided not to allow this behavior because if a process drops all privileges  
 127154 and sets its effective user and group IDs to be the same as its real user and group IDs before  
 127155 executing a program or utility, the behavior should be the same as if the process had originally  
 127156 met the security criteria.

127157 Austin Group Defect 367 is applied, adding requirements relating to the use of *readonly* on  
 127158 environment variables that are manipulated by shell built-in utilities.

127159 Austin Group Defect 922 is applied, allowing implementations to ignore some environment  
 127160 variables at the point of use for security reasons.

127161 Austin Group Defect 1561 is applied, clarifying that environment variable values can contain  
 127162 byte sequences that do not form valid characters.

## 127163 **A.8.2 Internationalization Variables**

127164 Utilities conforming to the Shell and Utilities volume of POSIX.1-2024 and written in standard C  
 127165 can access the locale variables by issuing the following call:

```
127166 setlocale(LC_ALL, "")
```

127167 If this were omitted, the ISO C standard specifies that the C (or POSIX) locale would be used.

127168 The DESCRIPTION of *setlocale()* requires that when setting all categories of a locale, if the value  
 127169 of any of the environment variable searches yields a locale that is not supported (and non-null),  
 127170 the *setlocale()* function returns a null pointer and the global locale is unchanged.

127171 For the standard utilities, if any of the environment variables are invalid, it makes sense to  
 127172 default to an implementation-defined, consistent locale environment. It is more confusing for a  
 127173 user to have partial settings occur in case of a mistake. All utilities would then behave in one  
 127174 language/cultural environment. Furthermore, it provides a way of forcing the whole  
 127175 environment to be the implementation-defined default. Disastrous results could occur if a  
 127176 pipeline of utilities partially uses the environment variables in different ways. In this case, it  
 127177 would be appropriate for utilities that use *LANG* and related variables to exit with an error if  
 127178 any of the variables are invalid. For example, users typing individual commands at a terminal  
 127179 might want *date* to work if *LC\_MONETARY* is invalid as long as *LC\_TIME* is valid. Since these  
 127180 are conflicting reasonable alternatives, POSIX.1-2024 leaves the results unspecified if the locale  
 127181 environment variables would not produce a complete locale matching the specification of the  
 127182 user.

127183 The *LC\_MESSAGES* variable affects the language of messages generated by the standard  
 127184 utilities.

127185 The description of the environment variable names starting with the characters “LC\_”  
 127186 acknowledges the fact that the interfaces presented may be extended as new international  
 127187 functionality is required. In the ISO C standard, names preceded by “LC\_” are reserved in the  
 127188 name space for future categories.

127189 To avoid name clashes, new categories and environment variables are divided into two  
 127190 classifications: “implementation-independent” and “implementation-defined”.

127191 Implementation-independent names will have the following format:

- 127192 LC\_NAME
- 127193 where *NAME* is the name of the new category and environment variable. Capital letters must be  
127194 used for implementation-independent names.
- 127195 Implementation-defined names must be in lowercase letters, as below:
- 127196 LC\_name
- 127197 Austin Group Defect 1122 is applied, adding the *LANGUAGE*, *TEXTDOMAIN*, and  
127198 *TEXTDOMAINDIR* environment variables and updating *NLS\_PATH* with requirements relating  
127199 to the *gettext* family of functions and the *gettext* and *ngettext* utilities.
- 127200 Austin Group Defect 1477 is applied, moving a paragraph of rationale about incompatible locale  
127201 categories to [Section A.7.1](#) (on page 3692).
- 127202 Austin Group Defect 1571 is applied, simplifying the final item in the precedence order for  
127203 internationalization environment variables.

### 127204 A.8.3 Other Environment Variables

#### 127205 COLUMNS, LINES

- 127206 The default values for the number of column positions when *COLUMNS* is unset or null, and  
127207 screen height when *LINES* is unset or null, are unspecified if the terminal window size cannot be  
127208 obtained (from *tcgetwinsize()*) because historical implementations use different methods to  
127209 determine the values. Users should not need to set these variables in the environment unless  
127210 there is a specific reason to override the default behavior of the implementation, such as to  
127211 display data in an area arbitrarily smaller than the terminal or window. Values for these  
127212 variables that are not decimal integers greater than zero are implicitly undefined values; it is  
127213 unnecessary to enumerate all of the possible values outside of the acceptable set.
- 127214 Austin Group Defect 1185 is applied, changing the descriptions of the *COLUMNS* and *LINES*  
127215 environment variables.

#### 127216 LOGNAME

- 127217 In most implementations, the value of such a variable is easily forged, so security-critical  
127218 applications should rely on other means of determining user identity. *LOGNAME* is required to  
127219 be constructed from the portable filename character set for reasons of interchange. No diagnostic  
127220 condition is specified for violating this rule, and no requirement for enforcement exists. The  
127221 intent of the requirement is that if extended characters are used, the “guarantee” of portability  
127222 implied by a standard is void.

#### 127223 PATH

- 127224 Many historical implementations of the Bourne shell do not interpret a trailing <colon> to  
127225 represent the current working directory and are thus non-conforming. The C Shell and the  
127226 KornShell conform to POSIX.1-2024 on this point. The usual name of dot may also be used to  
127227 refer to the current working directory.
- 127228 Many implementations historically have used a default value of */bin* and */usr/bin* for the *PATH*  
127229 variable. POSIX.1-2024 does not mandate this default path be identical to that retrieved from  
127230 *getconf PATH* because it is likely that the standardized utilities may be provided in another  
127231 directory separate from the directories used by some historical applications.
- 127232 The standard specifies that (when no <slash> character is included in a command pathname)  
127233 special built-in utilities and intrinsic utilities are not subject to a search using *PATH*. All other

127234 standard utilities, even if implemented as shell built-ins, are required to be found by searching  
 127235 *PATH*. This means that if a shell includes a built-in for a standard utility that is not intrinsic, a  
 127236 user can write a utility that will override that built-in. The standard also requires that all  
 127237 standard utilities can be executed by commands like:

```
127238 find . -type d -exec printf 'Found directory: %s\n' '{}' +
```

127239 So, other than differences caused by using different shell execution environments, a standard  
 127240 utility that is implemented as a built-in and the non-built-in version of that standard utility are  
 127241 both required to behave as the standard specifies. But, if a non-standard utility is found in *PATH*  
 127242 before the standard utility's location in *PATH*, the non-standard utility must be invoked rather  
 127243 than the built-in. For instance, if the shell includes a built-in *printf* utility (which most shells do),  
 127244 *PATH* is initialized using:

```
127245 PATH="$HOME/bin:$(command -p getconf PATH) "
```

127246 and ***\$HOME/bin/printf*** is an executable file containing:

```
127247 command -p printf 'In %s with args:\n' "${0##*/}" >&2  

  127248 command -p printf '%s\n' "$@" >&2  

  127249 command -V printf >&2  

  127250 command -Vp printf >&2  

  127251 command -p printf "$@"
```

127252 then the command:

```
127253 printf '%s %s\n' HOME "$HOME" PATH "$PATH"
```

127254 should produce output similar to:

```
127255 In printf with args:  

  127256     %s %s\n  

  127257     HOME  

  127258     /Users/dwc  

  127259     PATH  

  127260     /Users/dwc/bin:/usr/bin:/bin:/usr/sbin:/sbin  

  127261 printf is a tracked alias for /Users/dwc/bin/printf  

  127262 printf is a shell builtin  

  127263 HOME /Users/dwc  

  127264 PATH /Users/dwc/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

127265 The current version of the Korn shell installs built-ins into the shell using a *builtin* utility that  
 127266 allows the built-in to be associated with the pathname of the non-built-in version of that utility.  
 127267 (Unfortunately, some implementations that use *ksh93* as their standard *sh* utility do not make use  
 127268 of this feature and install built-ins for standard utilities that are not associated with a *PATH*  
 127269 search. And, most other shells incorrectly always use a built-in utility if one is installed, even  
 127270 when it should be overridden by a *PATH* search that should find the non-standard version of a  
 127271 utility with the name of that built-in.) Some other shells use a `<percent-sign>` character in a  
 127272 directory pathname in *PATH* to indicate one or more directories that should be used when  
 127273 processing *PATH* to determine when non-intrinsic standard utilities should be found. The  
 127274 POSIX.1-2024 revision of the standard allows either of these methods to be used to install built-  
 127275 ins that meet the requirements stated in XCU [Section 2.9.1.4](#) (on page 2502) by making the  
 127276 behavior of the built-in path search implementation-defined when a `<percent-sign>` character is  
 127277 found in *PATH*.

127278 Austin Group Defect 854 is applied, changing how *PATH* searching applies to built-in utilities.

127279 Austin Group Defect 1340 is applied, clarifying the description of *PATH*.

127280 **SHELL**

127281 The *SHELL* variable names the preferred shell of the user; it is a guide to applications. There is  
 127282 no direct requirement that that shell conform to POSIX.1-2024; that decision should rest with the  
 127283 user. It is the intention of the standard developers that alternative shells be permitted, if the user  
 127284 chooses to develop or acquire one. An operating system that builds its shell into the “kernel” in  
 127285 such a manner that alternative shells would be impossible does not conform to the spirit of  
 127286 POSIX.1-2024.

127287 **TZ**

127288 The quoted form of the timezone variable allows timezone names of the form UTC+1 (or any  
 127289 name that contains the <plus-sign> ('+'), the <hyphen-minus> ('-'), or digits), which may be  
 127290 appropriate for countries that do not have an official timezone name. It would be coded as  
 127291 <UTC+1>+1<UTC+2>, which would cause *std* to have a value of UTC+1 and *dst* a value of  
 127292 UTC+2, each with a length of 5 characters. This does not appear to conflict with any existing  
 127293 usage. The characters '<' and '>' were chosen for quoting because they are easier to parse  
 127294 visually than a quoting character that does not provide some sense of bracketing (and in a string  
 127295 like this, such bracketing is helpful). They were also chosen because they do not need special  
 127296 treatment when assigning to the *TZ* variable. Users are often confused by embedding quotes in a  
 127297 string. Because '<' and '>' are meaningful to the shell, the whole string would have to be  
 127298 quoted, but that is easily explained. (Parentheses would have presented the same problems.)  
 127299 Although the '>' symbol could have been permitted in the string by either escaping it or  
 127300 doubling it, it seemed of little value to require that. This could be provided as an extension if  
 127301 there was a need. Timezone names of this new form lead to a requirement that the value of  
 127302 {\_POSIX\_TZNAME\_MAX} change from 3 to 6.

127303 Since the *TZ* environment variable is usually inherited by all applications started by a user after  
 127304 the value of the *TZ* environment variable is changed and since many applications run using the  
 127305 C or POSIX locale, using characters that are not in the portable character set in the *std* and *dst*  
 127306 fields could cause unexpected results.

127307 Implementations are encouraged to incorporate the IANA timezone database into the timezone  
 127308 database used for *TZ* values specifying geographical and special timezones, and to provide a  
 127309 method to allow it to be updated in accordance with RFC 6557.

127310 The *TZ* format beginning with <colon> was originally introduced as a way for implementations  
 127311 to support geographical timezones in the form :*Area/Location* as an extension, but  
 127312 implementations started to support them without the leading <colon> (as well as with it) and  
 127313 their use without the <colon> became the de-facto standard. Consequently when geographical  
 127314 timezones were added to this standard, it was without the <colon>.

127315 The format of the *TZ* environment variable is changed in Issue 6 to allow for the quoted form, as  
 127316 defined in earlier versions of the ISO POSIX-1 standard.

127317 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/7 is applied, adding the *ctime\_r()* and  
 127318 *localtime\_r()* functions to the list of functions that use the *TZ* environment variable.

127319 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0041 [584] is applied.

127320 Austin Group Defect 1030 is applied, making it implementation-defined when the changes to  
 127321 and from Daylight Saving Time occur if the *dst* field is specified in *TZ* and the *rule* field is not.

127322 Austin Group Defect 1252 is applied, changing the *time* field to allow the hour to range from  
 127323 zero to 167 and allowing a leading sign.

127324 Austin Group Defect 1253 is applied, changing “alternative time” to “Daylight Saving Time”.

127325 Austin Group Defect 1410 is applied, removing the *ctime\_r()* function.

127326 Austin Group Defect 1619 is applied, adding support for a third TZ format with values  
127327 specifying geographical and special timezones.

127328 Austin Group Defects 1638 and 1639 are applied, clarifying the length limits for the *std* and *dst*  
127329 fields of TZ.

## 127330 A.9 Regular Expressions

127331 Rather than repeating the description of REs for each utility supporting REs, the standard  
127332 developers preferred a common, comprehensive description of regular expressions in one place.  
127333 The most common behavior is described here, and exceptions or extensions to this are  
127334 documented for the respective utilities, as appropriate.

127335 The BRE corresponds to the *ed* or historical *grep* type, and the ERE corresponds to the historical  
127336 *egrep* type (now *grep -E*).

127337 The text is based on the *ed* description and substantially modified, primarily to aid developers  
127338 and others in the understanding of the capabilities and limitations of REs. Much of this was  
127339 influenced by internationalization requirements.

127340 It should be noted that the definitions in this section do not cover the *tr* utility; the *tr* syntax does  
127341 not employ REs.

127342 The specification of REs is particularly important to internationalization because pattern  
127343 matching operations are very basic operations in business and other operations. The syntax and  
127344 rules of REs are intended to be as intuitive as possible to make them easy to understand and use.  
127345 The historical rules and behavior do not provide that capability to non-English language users,  
127346 and do not provide the necessary support for commonly used characters and language  
127347 constructs. It was necessary to provide extensions to the historical RE syntax and rules to  
127348 accommodate other languages.

127349 As they are limited to bracket expressions, the rationale for these modifications is in XBD [Section](#)  
127350 [9.3.5](#) (on page 182).

### 127351 A.9.1 Regular Expression Definitions

127352 It is possible to determine what strings correspond to subexpressions by recursively applying  
127353 the leftmost longest rule to each subexpression, but only with the proviso that the overall match  
127354 is leftmost longest. For example, matching "`\(ac*\\)c*d[ac]*\1`" against *acdacaaa* matches  
127355 *acdacaaa* (with `\1=a`); simply matching the longest match for "`\(ac*\\)`" would yield `\1=ac`, but  
127356 the overall match would be smaller (*acdac*). Conceptually, the implementation must examine  
127357 every possible match and among those that yield the leftmost longest total matches, pick the one  
127358 that does the longest match for the leftmost subexpression, and so on. Note that this means that  
127359 matching by subexpressions is context-dependent: a subexpression within a larger RE may  
127360 match a different string from the one it would match as an independent RE, and two instances of  
127361 the same subexpression within the same larger RE may match different lengths even in similar  
127362 sequences of characters. For example, in the ERE "`(a.*b)(a.*b)`", the two identical  
127363 subexpressions would match four and six characters, respectively, of *acbbaccbb*.

127364 The definition of single character has been expanded to include also collating elements  
127365 consisting of two or more characters; this expansion is applicable only when a bracket  
127366 expression is included in the BRE or ERE. An example of such a collating element may be the  
127367 Dutch *ij*, which collates as a 'y'. In some encodings, a ligature ``i with j'' exists as a character  
127368 and would represent a single-character collating element. In another encoding, no such ligature  
127369 exists, and the two-character sequence *ij* is defined as a multi-character collating element.

127370 Outside brackets, the *ij* is treated as a two-character RE and matches the same characters in a  
 127371 string. Historically, a bracket expression only matched a single character. The ISO POSIX-2: 1993  
 127372 standard required bracket expressions like "[[:lower:]]" to match multi-character collating  
 127373 elements such as "ij". However, this requirement led to behavior that many users did not  
 127374 expect and that could not feasibly be mimicked in user code, and it was rarely if ever  
 127375 implemented correctly. The current standard leaves it unspecified whether a bracket expression  
 127376 matches a multi-character collating element, allowing both historical and ISO POSIX-2: 1993  
 127377 standard implementations to conform.

127378 Also, in the current standard, it is unspecified whether character class expressions like  
 127379 "[[:lower:]]" can include multi-character collating elements like "ij"; hence  
 127380 "[[:lower:]]" can match "ij", and "[^[:lower:]]" can fail to match "ij". Common  
 127381 practice is for a character class expression to match a collating element if it matches the collating  
 127382 element's first character.

127383 Austin Group Defect 1329 is applied, adding a definition of "leftmost" and updating the  
 127384 definition of "matched" to include an example ERE using the repetition modifier '?'.  
 127385

127385 Austin Group Defect 1546 is applied, adding a definition of "escape sequence".

## 127386 A.9.2 Regular Expression General Requirements

127387 The definition of which sequence is matched when several are possible is based on the leftmost-  
 127388 longest rule historically used by deterministic recognizers. This rule is easier to define and  
 127389 describe, and arguably more useful, than the first-match rule historically used by non-  
 127390 deterministic recognizers. It is thought that dependencies on the choice of rule are rare; carefully  
 127391 contrived examples are needed to demonstrate the difference.

127392 A formal expression of the leftmost-longest rule is:

127393       The search is performed as if all possible suffixes of the string were tested for a prefix  
 127394       matching the pattern; the longest suffix containing a matching prefix is chosen, and the  
 127395       longest possible matching prefix of the chosen suffix is identified as the matching  
 127396       sequence.

127397 EREs can optionally use a leftmost-shortest rule for repetitions (enabled via the REG\_MINIMAL  
 127398 flag or the '?' repetition modifier), in which case the shortest possible matching prefix is  
 127399 instead identified as the matching sequence for the affected repetition(s).

127400 Historically, most RE implementations only match lines, not strings. However, that is more an  
 127401 effect of the usage than of an inherent feature of REs themselves. Consequently, POSIX.1-2024  
 127402 does not regard <newline> characters as special; they are ordinary characters, and both a  
 127403 <period> and a non-matching list can match them. Those utilities (like *grep*) that do not allow  
 127404 <newline> characters to match are responsible for eliminating any <newline> from strings  
 127405 before matching against the RE. The *regcomp()* function, however, can provide support for such  
 127406 processing without violating the rules of this section.

127407 Some implementations of *egrep* have had very limited flexibility in handling complex EREs.  
 127408 POSIX.1-2024 does not attempt to define the complexity of a BRE or ERE, but does place a lower  
 127409 limit on it—any RE must be handled, as long as it can be expressed in 256 bytes or less. (Of  
 127410 course, this does not place an upper limit on the implementation.) There are historical programs  
 127411 using a non-deterministic-recognizer implementation that should have no difficulty with this  
 127412 limit. It is possible that a good approach would be to attempt to use the faster, but more limited,  
 127413 deterministic recognizer for simple expressions and to fall back on the non-deterministic  
 127414 recognizer for those expressions requiring it. Non-deterministic implementations must be  
 127415 careful to observe the rules on which match is chosen; the longest match, not the first match,

127416 starting at a given character is used.

127417 The term “invalid” highlights a difference between this section and some others: POSIX.1-2024  
 127418 frequently avoids mandating of errors for syntax violations because they can be used by  
 127419 implementors to trigger extensions. However, the authors of the internationalization features of  
 127420 REs wanted to mandate errors for certain conditions to identify usage problems or non-portable  
 127421 constructs. These are identified within this rationale as appropriate. The remaining syntax  
 127422 violations have been left implicitly or explicitly undefined. For example, the BRE construct  
 127423 “\{1, 2, 3\}” does not comply with the grammar. A conforming application cannot rely on it  
 127424 producing an error nor matching the literal characters “\{1, 2, 3\}”.

127425 The term “undefined” was used in favor of “unspecified” because many of the situations are  
 127426 considered errors on some implementations, and the standard developers considered that  
 127427 consistency throughout the section was preferable to mixing undefined and unspecified.

127428 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0042 [554] is applied.

127429 Austin Group Defect 1031 is applied, replacing text relating to case insensitive comparisons with  
 127430 a reference to XBD [Section 4.1](#) (on page 95).

### 127431 **A.9.3 Basic Regular Expressions**

127432 Austin Group Defect 1139 is applied, making minor editorial changes to several subsections of  
 127433 this section and changing them to require that, when not inside a bracket expression, “\]”  
 127434 matches ‘]’.

#### 127435 *A.9.3.1 BREs Matching a Single Character or Collating Element*

127436 There is no additional rationale provided for this section.

#### 127437 *A.9.3.2 BRE Ordinary Characters*

127438 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0043 [554] is applied.

127439 Austin Group Defect 1546 is applied, adding optional support for “\?”, “\+”, and “\|”.

#### 127440 *A.9.3.3 BRE Special Characters*

127441 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0043 [554] is applied.

127442 Austin Group Defect 1546 is applied, adding optional support for “\?”, “\+”, and “\|”.

#### 127443 *A.9.3.4 Periods in BREs*

127444 There is no additional rationale provided for this section.

#### 127445 *A.9.3.5 RE Bracket Expression*

127446 Range expressions are, historically, an integral part of REs. However, the requirements of  
 127447 “natural language behavior” and portability do conflict. In the POSIX locale, ranges must be  
 127448 treated according to the collating sequence and include such characters that fall within the range  
 127449 based on that collating sequence, regardless of character values. In other locales, ranges have  
 127450 unspecified behavior.

127451 Some historical implementations allow range expressions where the ending range point of one  
 127452 range is also the starting point of the next (for instance, "[a-m-o]"). This behavior should not  
 127453 be permitted, but to avoid breaking historical implementations, it is now *undefined* whether it is  
 127454 a valid expression and how it should be interpreted.

127455 Current practice in *awk* and *lex* is to accept escape sequences in bracket expressions as per XBD  
 127456 Table 5-1 (on page 113), while the normal ERE behavior is to regard such a sequence as  
 127457 consisting of two characters. Allowing the *awk/lex* behavior in EREs would change the normal  
 127458 behavior in an unacceptable way; it is expected that *awk* and *lex* will decode escape sequences in  
 127459 EREs before passing them to *regcomp()* or comparable routines. Each utility describes the escape  
 127460 sequences it accepts as an exception to the rules in this section; the list is not the same, for  
 127461 historical reasons.

127462 As noted previously, the new syntax and rules have been added to accommodate other  
 127463 languages than English. The remainder of this section describes the rationale for these  
 127464 modifications.

127465 In the POSIX locale, a regular expression that starts with a range expression matches a set of  
 127466 strings that are contiguously sorted, but this is not necessarily true in other locales. For example,  
 127467 a French locale might have the following behavior:

```
127468 $ ls
127469 alpha Alpha estimé ESTIMÉ été eurêka
127470 $ ls [a-e]*
127471 alpha Alpha estimé eurêka
```

127472 Such disagreements between matching and contiguous sorting are unavoidable because POSIX  
 127473 sorting cannot be implemented in terms of a deterministic finite-state automaton (DFA), but  
 127474 range expressions by design are implementable in terms of DFAs.

127475 Historical implementations used native character order to interpret range expressions. The  
 127476 ISO POSIX-2:1993 standard instead required collating element order (CEO): the order that  
 127477 collating elements were specified between the **order\_start** and **order\_end** keywords in the  
 127478 *LC\_COLLATE* category of the current locale. CEO had some advantages in portability over the  
 127479 native character order, but it also had some disadvantages:

- 127480 • CEO could not feasibly be mimicked in user code, leading to inconsistencies between  
 127481 POSIX matchers and matchers in popular user programs like Emacs, *ksh*, and Perl.
- 127482 • CEO caused range expressions to match accented and capitalized letters contrary to many  
 127483 users' expectations. For example, "[a-e]" typically matched both 'E' and 'á' but  
 127484 neither 'A' nor 'é'.
- 127485 • CEO was not consistent across implementations. In practice, CEO was often less portable  
 127486 than native character order. For example, it was common for the CEOs of two  
 127487 implementation-supplied locales to disagree, even if both locales were named "da\_DK".

127488 Because of these problems, some implementations of regular expressions continued to use native  
 127489 character order. Others used the collation sequence, which is more consistent with sorting than  
 127490 either CEO or native order, but which departs further from the traditional POSIX semantics  
 127491 because it generally requires "[a-e]" to match either 'A' or 'E' but not both. As a result of  
 127492 this kind of implementation variation, programmers who wanted to write portable regular  
 127493 expressions could not rely on the ISO POSIX-2:1993 standard guarantees in practice.

127494 While revising the standard, lengthy consideration was given to proposals to attack this problem  
 127495 by adding an API for querying the CEO to allow user-mode matchers, but none of these  
 127496 proposals had implementation experience and none achieved consensus. Leaving the standard  
 127497 alone was also considered, but rejected due to the problems described above.



127498 The current standard leaves unspecified the behavior of a range expression outside the POSIX  
 127499 locale. This makes it clearer that conforming applications should avoid range expressions  
 127500 outside the POSIX locale, and it allows implementations and compatible user-mode matchers to  
 127501 interpret range expressions using native order, CEO, collation sequence, or other, more  
 127502 advanced techniques. The concerns which led to this change were raised in IEEE PASC  
 127503 interpretation 1003.2 #43 and others, and related to ambiguities in the specification of how  
 127504 multi-character collating elements should be handled in range expressions. These ambiguities  
 127505 had led to multiple interpretations of the specification, in conflicting ways, which led to varying  
 127506 implementations. As noted above, efforts were made to resolve the differences, but no solution  
 127507 has been found that would be specific enough to allow for portable software while not  
 127508 invalidating existing implementations.

127509 The standard developers recognize that collating elements are important, such elements being  
 127510 common in several European languages; for example, 'ch' or 'll' in traditional Spanish;  
 127511 'aa' in several Scandinavian languages. Existing internationalized implementations have  
 127512 processed, and continue to process, these elements in range expressions. Efforts are expected to  
 127513 continue in the future to find a way to define the behavior of these elements precisely and  
 127514 portably.

127515 The ISO POSIX-2:1993 standard required "[b-a]" to be an invalid expression in the POSIX  
 127516 locale, but this requirement has been relaxed in this version of the standard so that "[b-a]" can  
 127517 instead be treated as a valid expression that does not match any string.

127518 The standard specifies three possible behaviors for regular expressions such as "[alpha:]".  
 127519 One behavior is the traditional implementation, which behaves like "[ahlp]". Another, for  
 127520 alignment with the *tr* utility, is to treat it like "[[:alpha:]]". And finally, the standard allows  
 127521 rejecting the regular expression as invalid, as a means of alerting a user to the non-portable  
 127522 aspect of that regular expression. The set of regular expressions with this undefined behavior is  
 127523 limited solely to the expressions where the outer '[' and ']' of the bracket expression can be  
 127524 confused with the missing bracket pair '[' and ']' necessary to form a collating symbol,  
 127525 equivalence class, or character class; thus "[\_alpha:]" or "[::]" do not trigger the  
 127526 unspecified behavior.

127527 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0044 [938], XBD/TC2-2008/0045 [872],  
 127528 XBD/TC2-2008/0046 [938], XBD/TC2-2008/0047 [584], and XBD/TC2-2008/0048 [584] are  
 127529 applied.

127530 Austin Group Defect 948 is applied, requiring that an ordinary character in a matching list only  
 127531 matches that character.

127532 Austin Group Defect 1190 is applied, clarifying which characters lose their special meaning  
 127533 inside a bracket expression.

127534 Austin Group Defect 1288 is applied, changing "rejected as an error" to "treated as an invalid  
 127535 bracket expression".

#### 127536 A.9.3.6 BREs Matching Multiple Characters

127537 The limit of nine back-references to subexpressions in the RE is based on the use of a single-digit  
 127538 identifier; increasing this to multiple digits would break historical applications. This does not  
 127539 imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten  
 127540 subexpressions:

```
127541 \ (\ ( (ab\)*c\)*d\)\ (e\)\* \ (gh\)\{2\}\ (ij\)* \ (kl\)* \ (mn\)* \ (op\)* \ (qr\)*
```

127542 The standard developers regarded the common historical behavior, which supported "\n\*", but  
 127543 not "\n{min,max}", "\(...)\*", or "\(...)\{min,max\}", as a non-intentional

- 127544 result of a specific implementation, and they supported both duplication and interval  
127545 expressions following subexpressions and back-references.
- 127546 The changes to the processing of the back-reference expression remove an unspecified or  
127547 ambiguous behavior in the Shell and Utilities volume of POSIX.1-2024, aligning it with the  
127548 requirements specified for the *regcomp()* expression, and is the result of PASC Interpretation  
127549 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.
- 127550 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0049 [595] is applied.
- 127551 *A.9.3.7 BRE Precedence*
- 127552 There is no additional rationale provided for this section.
- 127553 *A.9.3.8 BRE Expression Anchoring*
- 127554 Often, the <dollar-sign> is viewed as matching the ending <newline> in text files. This is not  
127555 strictly true; the <newline> is typically eliminated from the strings to be matched, and the  
127556 <dollar-sign> matches the terminating null character.
- 127557 The ability of '^', '\$', and '\*' to be non-special in certain circumstances may be confusing to  
127558 some programmers, but this situation was changed only in a minor way from historical practice  
127559 to avoid breaking many historical scripts. Some consideration was given to making the use of  
127560 the anchoring characters undefined if not escaped and not at the beginning or end of strings.  
127561 This would cause a number of historical BREs, such as "2^10", "\$HOME", and "\$1.35", that  
127562 relied on the characters being treated literally, to become invalid.
- 127563 However, one relatively uncommon case was changed to allow an extension used on some  
127564 implementations. Historically, the BREs "^foo" and "\(^foo\)" did not match the same  
127565 string, despite the general rule that subexpressions and entire BREs match the same strings. To  
127566 increase consensus, POSIX.1-2024 has allowed an extension on some implementations to treat  
127567 these two cases in the same way by declaring that anchoring *may* occur at the beginning or end  
127568 of a subexpression. Therefore, portable BREs that require a literal <circumflex> at the beginning  
127569 or a <dollar-sign> at the end of a subexpression must escape them. Note that a BRE such as  
127570 "a\(^bc\)" will either match "a^bc" or nothing on different systems under the rules.
- 127571 ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped  
127572 anchor character has never matched its literal counterpart outside a bracket expression. Some  
127573 implementations treated "foo\$bar" as a valid expression that never matched anything; others  
127574 treated it as invalid. POSIX.1-2024 mandates the former, valid unmatched behavior.
- 127575 Some implementations have extended the BRE syntax to add alternation. For example, the  
127576 subexpression "\ (foo\$ | bar\)" would match either "foo" at the end of the string or "bar"  
127577 anywhere. The extension is triggered by the use of the undefined "\|" sequence. Because the  
127578 BRE is undefined for portable scripts, the extending system is free to make other assumptions,  
127579 such that the '\$' represents the end-of-line anchor in the middle of a subexpression. If it were  
127580 not for the extension, the '\$' would match a literal <dollar-sign> under the rules.
- 127581 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0049 [595] is applied.
- 127582 Austin Group Defect 1546 is applied, adding optional support for "\?", "\+", and "\|".
- 127583 Austin Group Defect 1579 is applied, eliminating an inconsistency between the list items relating  
127584 to <circumflex> and <dollar-sign>.

127585 **A.9.4 Extended Regular Expressions**

127586 As with BREs, the standard developers decided to make the interpretation of escaped ordinary  
127587 characters undefined.

127588 The <right-parenthesis> is not listed as an ERE special character because it is only special in the  
127589 context of a preceding <left-parenthesis>. If found without a preceding <left-parenthesis>, the  
127590 <right-parenthesis> has no special meaning.

127591 The interval expression, " $\{m, n\}$ ", has been added to EREs. Historically, the interval expression  
127592 has only been supported in some ERE implementations. The standard developers estimated that  
127593 the addition of interval expressions to EREs would not decrease consensus and would also make  
127594 BREs more of a subset of EREs than in many historical implementations.

127595 It was suggested that, in addition to interval expressions, back-references (' $\backslash n$ ') should also be  
127596 added to EREs. This was rejected by the standard developers as likely to decrease consensus.

127597 In historical implementations, multiple duplication symbols are usually interpreted from left to  
127598 right and treated as additive. As an example, " $a^*b$ " matches zero or more instances of 'a'  
127599 followed by a 'b'. In POSIX.1-2024, multiple duplication symbols are undefined; that is, they  
127600 cannot be relied upon for conforming applications. One reason for this is to provide some scope  
127601 for future enhancements.

127602 The precedence of operations differs between EREs and those in *lex*; in *lex*, for historical reasons,  
127603 interval expressions have a lower precedence than concatenation.

127604 Austin Group Defect 1139 is applied, making minor editorial changes to several subsections of  
127605 this section and changing them to require that, when not inside a bracket expression, " $\backslash ]$ "  
127606 matches ']' and " $\backslash }$ " matches '}'.

127607 *A.9.4.1 EREs Matching a Single Character or Collating Element*

127608 There is no additional rationale provided for this section.

127609 *A.9.4.2 ERE Ordinary Characters*

127610 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0050 [554] is applied.

127611 *A.9.4.3 ERE Special Characters*

127612 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0050 [554] is applied.

127613 *A.9.4.4 Periods in EREs*

127614 There is no additional rationale provided for this section.

127615 *A.9.4.5 ERE Bracket Expression*

127616 There is no additional rationale provided for this section.

127617 A.9.4.6 EREs Matching Multiple Characters

127618 Austin Group Defects 793 and 1329 are applied, adding the repetition modifier '?' and the  
127619 REG\_MINIMAL flag.

127620 A.9.4.7 ERE Alternation

127621 There is no additional rationale provided for this section.

127622 A.9.4.8 ERE Precedence

127623 There is no additional rationale provided for this section.

127624 A.9.4.9 ERE Expression Anchoring

127625 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0051 [595] is applied.

## 127626 A.9.5 Regular Expression Grammar

127627 The grammars are intended to represent the range of acceptable syntaxes available to  
127628 conforming applications. There are instances in the text where undefined constructs are  
127629 described; as explained previously, these allow implementation extensions. There is no intended  
127630 requirement that an implementation extension must somehow fit into the grammars shown  
127631 here.

127632 The BRE grammar does not permit L\_ANCHOR or R\_ANCHOR inside "\ (" and "\ )" (which  
127633 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the  
127634 application, as noted in XBD Section 9.3.8 (on page 186). Implementations are permitted to  
127635 extend the language to interpret '^' and '\$' as anchors in these locations, and as such,  
127636 conforming applications cannot use unescaped '^' and '\$' in positions inside "\ (" and "\ )"   
127637 that might be interpreted as anchors.

127638 The ERE grammar does not permit several constructs that XBD Section 9.4.2 (on page 187) and  
127639 Section 9.4.3 (on page 188) specify as having undefined results:

- 127640 • ORD\_CHAR preceded by <backslash>
- 127641 • ERE\_dupl\_symbol(s) appearing first in an ERE, or immediately following '|', '^', or '('
- 127642 • '{' not part of a valid ERE\_dupl\_symbol
- 127643 • '|' appearing first or last in an ERE, or immediately following '|' or '(' , or  
127644 immediately preceding ')'

127645 Implementations are permitted to extend the language to allow these. Conforming applications  
127646 cannot use such constructs.

127647 A.9.5.1 BRE/ERE Grammar Lexical Conventions

127648 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0052 [554] is applied.

127649 Austin Group Defect 1139 is applied, updating QUOTED\_CHAR to add \] to the BRE list and  
127650 add \] and \} to the ERE list, and changing "outside bracket expressions" to "except inside  
127651 bracket expressions".

127652 Austin Group Defect 1546 is applied, adding optional support for \?, \+, and \| in BREs.

## 127653 A.9.5.2 RE and Bracket Expression Grammar

127654 The removal of the `Back_open_paren Back_close_paren` option from the `nondupl_RE` specification is  
 127655 the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2:1993 standard.  
 127656 Although the grammar required support for null subexpressions, this section does not describe  
 127657 the meaning of, and historical practice did not support, this construct.

127658 Austin Group Defect 1546 is applied, adding optional support for `\?`, `\+`, and `\|` in BREs.

## 127659 A.9.5.3 ERE Grammar

127660 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0052 [554] and XBD/TC2-2008/0053  
 127661 [916] are applied.

127662 **A.10 Directory Structure and Devices**127663 **A.10.1 Directory Structure and Files**

127664 A description of the historical `/usr/tmp` was omitted, removing any concept of differences in  
 127665 emphasis between the `/` and `/usr` directories. The descriptions of `/bin`, `/usr/bin`, `/lib`, and `/usr/lib`  
 127666 were omitted because they are not useful for applications. In an early draft, a distinction was  
 127667 made between system and application directory usage, but this was not found to be useful.

127668 The directories `/` and `/dev` are included because the notion of a hierarchical directory structure is  
 127669 key to other information presented elsewhere in POSIX.1-2024. In early drafts, it was argued that  
 127670 special devices and temporary files could conceivably be handled without a directory structure  
 127671 on some implementations. For example, the system could treat the characters `"/tmp"` as a  
 127672 special token that would store files using some non-POSIX file system structure. This notion was  
 127673 rejected by the standard developers, who required that all the files in this section be  
 127674 implemented via POSIX file systems.

127675 The `/tmp` directory is retained in POSIX.1-2024 to accommodate historical applications that  
 127676 assume its availability. Implementations are encouraged to provide suitable directory names in  
 127677 the environment variable `TMPDIR` and applications are encouraged to use the contents of  
 127678 `TMPDIR` for creating temporary files.

127679 The standard files `/dev/null` and `/dev/tty` are required to be both readable and writable to allow  
 127680 applications to have the intended historical access to these files.

127681 The standard file `/dev/console` has been added for alignment with the Single UNIX  
 127682 Specification.

127683 **A.10.2 Output Devices and Terminal Types**

127684 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/17 is applied, making it clear that the  
 127685 requirements for documenting terminal support are in the system documentation.

127686 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0054 [967] is applied.

## 127687 A.11 General Terminal Interface

127688 If the implementation does not support this interface on any device types, it should behave as if  
 127689 it were being used on a device that is not a terminal device (in most cases *errno* will be set to  
 127690 [ENOTTY] on return from functions defined by this interface). This is based on the fact that  
 127691 many applications are written to run both interactively and in some non-interactive mode, and  
 127692 they adapt themselves at runtime. Requiring that they all be modified to test an environment  
 127693 variable to determine whether they should try to adapt is unnecessary. On a system that  
 127694 provides no general terminal interface, providing all the entry points as stubs that return  
 127695 [ENOTTY] (or an equivalent, as appropriate) has the same effect and requires no changes to the  
 127696 application.

127697 Although the needs of both interface implementors and application developers were addressed  
 127698 throughout POSIX.1-2024, this section pays more attention to the needs of the latter. This is  
 127699 because, while many aspects of the programming interface can be hidden from the user by the  
 127700 application developer, the terminal interface is usually a large part of the user interface.  
 127701 Although to some extent the application developer can build missing features or work around  
 127702 inappropriate ones, the difficulties of doing that are greater in the terminal interface than  
 127703 elsewhere. For example, efficiency prohibits the average program from interpreting every  
 127704 character passing through it in order to simulate character erase, line kill, and so on. These  
 127705 functions should usually be done by the operating system, possibly at the interrupt level.

127706 The *tc\**() functions were introduced as a way of avoiding the problems inherent in the  
 127707 traditional *ioctl*() function and in variants of it that were proposed. For example, *tcsetattr*() is  
 127708 specified in place of the use of the TCSETA *ioctl*() command function. This allows specification  
 127709 of all the arguments in a manner consistent with the ISO C standard unlike the varying third  
 127710 argument of *ioctl*(), which is sometimes a pointer (to any of many different types) and  
 127711 sometimes an **int**.

127712 The advantages of this new method include:

- 127713 • It allows strict type checking.
- 127714 • The direction of transfer of control data is explicit.
- 127715 • Portable capabilities are clearly identified.
- 127716 • The need for a general interface routine is avoided.
- 127717 • Size of the argument is well-defined (there is only one type).

127718 The disadvantages include:

- 127719 • No historical implementation used the new method.
- 127720 • There are many small routines instead of one general-purpose one.
- 127721 • The historical parallel with *fcntl*() is broken.

127722 The issue of modem control was excluded from POSIX.1-2024 on the grounds that:

- 127723 • It was concerned with setting and control of hardware timers.
- 127724 • The appropriate timers and settings vary widely internationally.
- 127725 • Feedback from European computer manufacturers indicated that this facility was not  
 127726 consistent with European needs and that specification of such a facility was not a  
 127727 requirement for portability.

127728 **A.11.1 Interface Characteristics**127729 *A.11.1.1 Opening a Terminal Device File*

127730 The `O_TTY_INIT` flag for `open()` has been added to POSIX.1-2024 to solve a problem  
 127731 encountered by applications written for earlier versions of this standard which need to open a  
 127732 modem or similar device and initialize all of the parameter settings. Using the  
 127733 `tcgetattr()-modify-tcsetattr()` method mandated by the standard could result in non-conforming  
 127734 behavior if the device had previously been used with non-conforming parameter settings, on  
 127735 implementations which do not reset the parameter settings in between the last close of the  
 127736 device by one application and the first open by another application. To avoid this problem, some  
 127737 application developers were resorting to using `memset()` to zero the **termios** structure before  
 127738 setting all of the standard parameters, but this risks non-conforming behavior on systems where  
 127739 some non-standard parameter needs a non-zero value in order for the terminal to behave in a  
 127740 conforming manner.

127741 On systems which do reset the parameter settings to defaults between uses of a terminal device,  
 127742 it is expected that either `O_TTY_INIT` will have the value zero or `open(ttypath,  
 127743 O_RDWR|O_TTY_INIT)` will do nothing additional.

127744 The standard developers considered an alternative solution of a special *fildev* argument for the  
 127745 `tcgetattr()` call to obtain default parameters. However, this would not be adequate if a system  
 127746 supports several different types of terminal device and the default settings need to differ  
 127747 between the different types. With the `O_TTY_INIT` open flag, the implementor can determine  
 127748 which device type is being opened.

127749 The standard developers also considered a special `POSIX_TTY_INIT` value for the **termios**  
 127750 structure used in `tcsetattr()`, which would reset the values if used immediately after an `open()`  
 127751 call. However, it was felt that this would lead to confusion amongst application developers who  
 127752 wanted to reset the parameters at other points, and implementations might diverge.

127753 Austin Group Defect 1466 is applied, changing the terminology used for pseudo-terminal  
 127754 devices.

127755 *A.11.1.2 Process Groups*

127756 There is a potential race when the members of the foreground process group on a terminal leave  
 127757 that process group, either by exit or by changing process groups. After the last process exits the  
 127758 process group, but before the foreground process group ID of the terminal is changed (usually  
 127759 by a job control shell), it would be possible for a new process to be created with its process ID  
 127760 equal to the terminal's foreground process group ID. That process might then become the  
 127761 process group leader and accidentally be placed into the foreground on a terminal that was not  
 127762 necessarily its controlling terminal. As a result of this problem, the controlling terminal is  
 127763 defined to not have a foreground process group during this time.

127764 The cases where a controlling terminal has no foreground process group occur when all  
 127765 processes in the foreground process group either terminate and are waited for or join other  
 127766 process groups via `setpgid()` or `setsid()`. If the process group leader terminates, this is the first  
 127767 case described; if it leaves the process group via `setpgid()`, this is the second case described (a  
 127768 process group leader cannot successfully call `setsid()`). When one of those cases causes a  
 127769 controlling terminal to have no foreground process group, it has two visible effects on  
 127770 applications. The first is the value returned by `tcgetpgrp()`. The second (which occurs only in the  
 127771 case where the process group leader terminates) is the sending of signals in response to special

127772 input characters. The intent of POSIX.1-2024 is that no process group be wrongly identified as  
 127773 the foreground process group by *tcgetpgrp()* or unintentionally receive signals because of  
 127774 placement into the foreground.

127775 In 4.3 BSD, the old process group ID continues to be used to identify the foreground process  
 127776 group and is returned by the function equivalent to *tcgetpgrp()*. In that implementation it is  
 127777 possible for a newly created process to be assigned the same value as a process ID and then form  
 127778 a new process group with the same value as a process group ID. The result is that the new  
 127779 process group would receive signals from this terminal for no apparent reason, and  
 127780 POSIX.1-2024 precludes this by forbidding a process group from entering the foreground in this  
 127781 way. It would be more direct to place part of the requirement made by the last sentence under  
 127782 *fork()*, but there is no convenient way for that section to refer to the value that *tcgetpgrp()*  
 127783 returns, since in this case there is no process group and thus no process group ID.

127784 One possibility for a conforming implementation is to behave similarly to 4.3 BSD, but to  
 127785 prevent this reuse of the ID, probably in the implementation of *fork()*, as long as it is in use by  
 127786 the terminal.

127787 Another possibility is to recognize when the last process stops using the terminal's foreground  
 127788 process group ID, which is when the process group lifetime ends, and to change the terminal's  
 127789 foreground process group ID to a reserved value that is never used as a process ID or process  
 127790 group ID. (See the definition of *process group lifetime* in the definitions section.) The process ID  
 127791 can then be reserved until the terminal has another foreground process group.

127792 The 4.3 BSD implementation permits the leader (and only member) of the foreground process  
 127793 group to leave the process group by calling the equivalent of *setpgid()* and to later return,  
 127794 expecting to return to the foreground. There are no known application needs for this behavior,  
 127795 and POSIX.1-2024 neither requires nor forbids it (except that it is forbidden for session leaders)  
 127796 by leaving it unspecified.

#### 127797 A.11.1.3 The Controlling Terminal

127798 POSIX.1-2024 does not specify a mechanism by which to allocate a controlling terminal. This is  
 127799 normally done by a system utility (such as *getty*) and is considered an administrative feature  
 127800 outside the scope of POSIX.1-2024.

127801 Historical implementations allocate controlling terminals on certain *open()* calls. Since *open()* is  
 127802 part of POSIX.1, its behavior had to be dealt with. The traditional behavior is not required  
 127803 because it is not very straightforward or flexible for either implementations or applications.  
 127804 However, because of its prevalence, it was not practical to disallow this behavior either. Thus, a  
 127805 mechanism was standardized to ensure portable, predictable behavior in *open()*.

127806 Some historical implementations deallocate a controlling terminal on the last system-wide close.  
 127807 This behavior is neither required nor prohibited. Even on implementations that do provide this  
 127808 behavior, applications generally cannot depend on it due to its system-wide nature.

#### 127809 A.11.1.4 Terminal Access Control

127810 The access controls described in this section apply only to a process that is accessing its  
 127811 controlling terminal. A process accessing a terminal that is not its controlling terminal is  
 127812 effectively treated the same as a member of the foreground process group. While this may seem  
 127813 unintuitive, note that these controls are for the purpose of job control, not security, and job  
 127814 control relates only to the controlling terminal of a process. Normal file access permissions  
 127815 handle security.

127816 If the process calling *read()* or *write()* is in a background process group that is orphaned, it is not



127817 desirable to stop the process group, as it is no longer under the control of a job control shell that  
 127818 could put it into the foreground again. Accordingly, calls to *read()* or *write()* functions by such  
 127819 processes receive an immediate error return. This is different from 4.2 BSD, which kills orphaned  
 127820 processes that receive terminal stop signals.

127821 The foreground/background/orphaned process group check performed by the terminal driver  
 127822 must be repeatedly performed until the calling process moves into the foreground or until the  
 127823 process group of the calling process becomes orphaned. That is, when the terminal driver  
 127824 determines that the calling process is in the background and should receive a job control signal,  
 127825 it sends the appropriate signal (SIGTTIN or SIGTTOU) to every process in the process group of  
 127826 the calling process and then it allows the calling process to immediately receive the signal. The  
 127827 latter is typically performed by blocking the process so that the signal is immediately noticed.  
 127828 Note, however, that after the process finishes receiving the signal and control is returned to the  
 127829 driver, the terminal driver must re-execute the foreground/background/orphaned process  
 127830 group check. The process may still be in the background, either because it was continued in the  
 127831 background by a job control shell, or because it caught the signal and did nothing.

127832 The terminal driver repeatedly performs the foreground/background/orphaned process group  
 127833 checks whenever a process is about to access the terminal. In the case of *write()* or the control  
 127834 *tc\*()* functions, the check is performed at the entry of the function. In the case of *read()*, the  
 127835 check is performed not only at the entry of the function, but also after blocking the process to  
 127836 wait for input characters (if necessary). That is, once the driver has determined that the process  
 127837 calling the *read()* function is in the foreground, it attempts to retrieve characters from the input  
 127838 queue. If the queue is empty, it blocks the process waiting for characters. When characters are  
 127839 available and control is returned to the driver, the terminal driver must return to the repeated  
 127840 foreground/background/orphaned process group check again. The process may have moved  
 127841 from the foreground to the background while it was blocked waiting for input characters.

127842 Austin Group Defect 1151 is applied, adding *tcsetwinsize()*.

#### 127843 A.11.1.5 *Input Processing and Reading Data*

127844 There is no additional rationale provided for this section.

#### 127845 A.11.1.6 *Canonical Mode Input Processing*

127846 The term “character” is intended here. ERASE should erase the last character, not the last byte.  
 127847 In the case of multi-byte characters, these two may be different.

127848 4.3 BSD has a WERASE character that erases the last “word” typed (but not any preceding  
 127849 <blank> or <tab> characters). A word is defined as a sequence of non-<blank> characters, with  
 127850 <tab> characters counted as <blank> characters. Like ERASE, WERASE does not erase beyond  
 127851 the beginning of the line. This WERASE feature has not been specified in POSIX.1 because it is  
 127852 difficult to define in the international environment. It is only useful for languages where words  
 127853 are delimited by <blank> characters. In some ideographic languages, such as Japanese and  
 127854 Chinese, words are not delimited at all. The WERASE character should presumably go back to  
 127855 the beginning of a sentence in those cases; practically, this means it would not be used much for  
 127856 those languages.

127857 It should be noted that there is a possible inherent deadlock if the application and  
 127858 implementation conflict on the value of {MAX\_CANON}. With ICANON set (if IXOFF is  
 127859 enabled) and more than {MAX\_CANON} characters transmitted without a <linefeed>,  
 127860 transmission will be stopped, the <linefeed> (or <carriage-return> when ICRLF is set) will never  
 127861 arrive, and the *read()* will never be satisfied.

127862 An application should not set IXOFF if it is using canonical mode unless it knows that (even in  
 127863 the face of a transmission error) the conditions described previously cannot be met or unless it is  
 127864 prepared to deal with the possible deadlock in some other way, such as timeouts.

127865 It should also be noted that this can be made to happen in non-canonical mode if the trigger  
 127866 value for sending IXOFF is less than VMIN and VTIME is zero.

#### 127867 A.11.1.7 Non-Canonical Mode Input Processing

127868 Some points to note about MIN and TIME:

- 127869 1. The interactions of MIN and TIME are not symmetric. For example, when MIN>0 and  
 127870 TIME=0, TIME has no effect. However, in the opposite case where MIN=0 and TIME>0,  
 127871 both MIN and TIME play a role in that MIN is satisfied with the receipt of a single  
 127872 character.
- 127873 2. Also note that in case A (MIN>0, TIME>0), TIME represents an inter-character timer,  
 127874 while in case C (MIN=0, TIME>0), TIME represents a read timer.

127875 These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where  
 127876 MIN>0, exist to handle burst-mode activity (for example, file transfer programs) where a  
 127877 program would like to process at least MIN characters at a time. In case A, the inter-character  
 127878 timer is activated by a user as a safety measure; in case B, it is turned off.

127879 Cases C and D exist to handle single-character timed transfers. These cases are readily adaptable  
 127880 to screen-based applications that need to know if a character is present in the input queue before  
 127881 refreshing the screen. In case C, the read is timed; in case D, it is not.

127882 Another important note is that MIN is always just a minimum. It does not denote a record  
 127883 length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20  
 127884 characters are returned to the user. In the special case of MIN=0, this still applies: if more than  
 127885 one character is available, they all will be returned immediately.

#### 127886 A.11.1.8 Writing Data and Output Processing

127887 There is no additional rationale provided for this section.

#### 127888 A.11.1.9 Special Characters

127889 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0055 [745] is applied.

#### 127890 A.11.1.10 Modem Disconnect

127891 There is no additional rationale provided for this section.

#### 127892 A.11.1.11 Closing a Terminal Device File

127893 POSIX.1-2024 does not specify that a *close()* on a terminal device file include the equivalent of a  
 127894 call to *tcflow(fd,TCOON)*.

127895 An implementation that discards output at the time *close()* is called after reporting the return  
 127896 value to the *write()* call that data was written does not conform with POSIX.1-2024. An  
 127897 application has functions such as *tcdrain()*, *tcflush()*, and *tcflow()* available to obtain the detailed  
 127898 behavior it requires with respect to flushing of output.

127899 At the time of the last close on a terminal device, an application relinquishes any ability to exert

127900 flow control via *tcflow()*.

## 127901 A.11.2 Parameters that Can be Set

### 127902 A.11.2.1 *The termios Structure*

127903 This structure is part of an interface that, in general, retains the historic grouping of flags.  
127904 Although a more optimal structure for implementations may be possible, the degree of change  
127905 to applications would be significantly larger.

### 127906 A.11.2.2 *Input Modes*

127907 Some historical implementations treated a long break as multiple events, as many as one per  
127908 character time. The wording in POSIX.1 explicitly prohibits this.

127909 Although the ISTRIP flag is normally superfluous with today's terminal hardware and software,  
127910 it is historically supported. Therefore, applications may be using ISTRIP, and there is no  
127911 technical problem with supporting this flag. Also, applications may wish to receive only 7-bit  
127912 input bytes and may not be connected directly to the hardware terminal device (for example,  
127913 when a connection traverses a network).

127914 Also, there is no requirement in general that the terminal device ensures that high-order bits  
127915 beyond the specified character size are cleared. ISTRIP provides this function for 7-bit  
127916 characters, which are common.

127917 In dealing with multi-byte characters, the consequences of a parity error in such a character, or  
127918 in an escape sequence affecting the current character set, are beyond the scope of POSIX.1 and  
127919 are best dealt with by the application processing the multi-byte characters.

### 127920 A.11.2.3 *Output Modes*

127921 POSIX.1 does not describe post-processing of output to a terminal or detailed control of that  
127922 from a conforming application. (That is, translation of <newline> to <carriage-return> followed  
127923 by <linefeed> or <tab> processing.) There is nothing that a conforming application should do to  
127924 its output for a terminal because that would require knowledge of the operation of the terminal.  
127925 It is the responsibility of the operating system to provide post-processing appropriate to the  
127926 output device, whether it is a terminal or some other type of device.

127927 Extensions to POSIX.1 to control the type of post-processing already exist and are expected to  
127928 continue into the future. The control of these features is primarily to adjust the interface between  
127929 the system and the terminal device so the output appears on the display correctly. This should  
127930 be set up before use by any application.

127931 In general, both the input and output modes should not be set absolutely, but rather modified  
127932 from the inherited state.

127933 A.11.2.4 *Control Modes*

127934 This section could be misread that the symbol ``CSIZE'' is a title in the **termios** *c\_flag* field.  
 127935 Although it does serve that function, it is also a required symbol, as a literal reading of POSIX.1  
 127936 (and the caveats about typography) would indicate.

127937 A.11.2.5 *Local Modes*

127938 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while still  
 127939 allowing single-character input.

127940 The ECHONL function historically has been in many implementations. Since there seems to be  
 127941 no technical problem with supporting ECHONL, it is included in POSIX.1 to increase consensus.

127942 The alternate behavior possible when ECHOK or ECHOE are specified with ICANON is  
 127943 permitted as a compromise depending on what the actual terminal hardware can do. Erasing  
 127944 characters and lines is preferred, but is not always possible.

127945 A.11.2.6 *Special Control Characters*

127946 Permitting VMIN and VTIME to overlap with VEOF and VEOL was a compromise for historical  
 127947 implementations. Only when backwards-compatibility of object code is a serious concern to an  
 127948 implementor should an implementation continue this practice. Correct applications that work  
 127949 with the overlap (at the source level) should also work if it is not present, but not the reverse.

127950 **A.12 Utility Conventions**127951 **A.12.1 Utility Argument Syntax**

127952 The standard developers considered that recent trends toward diluting the SYNOPSIS sections  
 127953 of historical reference pages to the equivalent of:

127954 `command [options] [operands]`

127955 were a disservice to the reader. Therefore, considerable effort was placed into rigorous  
 127956 definitions of all the command line arguments and their interrelationships. The relationships  
 127957 depicted in the synopses are normative parts of POSIX.1-2024; this information is sometimes  
 127958 repeated in textual form, but that is only for clarity within context.

127959 The use of ``undefined'' for conflicting argument usage and for repeated usage of the same  
 127960 option is meant to prevent conforming applications from using conflicting arguments or  
 127961 repeated options unless specifically allowed (as is the case with *ls*, which allows simultaneous,  
 127962 repeated use of the *-C*, *-l*, and *-1* options). Many historical implementations will tolerate this  
 127963 usage, choosing either the first or the last applicable argument. This tolerance can continue, but  
 127964 conforming applications cannot rely upon it. (Other implementations may choose to print usage  
 127965 messages instead.)

127966 The use of ``undefined'' for conflicting argument usage also allows an implementation to make  
 127967 reasonable extensions to utilities where the implementor considers mutually-exclusive options  
 127968 according to POSIX.1-2024 to have a sensible meaning and result.

127969 POSIX.1-2024 does not define the result of a command when an option-argument or operand is  
 127970 not followed by ellipses and the application specifies more than one of that option-argument or

127971 operand. This allows an implementation to define valid (although non-standard) behavior for  
 127972 the utility when more than one such option or operand is specified.

127973 The requirements for option-arguments are summarized as follows:

|                                      | SYNOPSIS Shows:                 |                      |
|--------------------------------------|---------------------------------|----------------------|
|                                      | -a <i>arg</i>                   | -c [ <i>arg</i> ]    |
| Conforming application uses:         | -a <i>arg</i>                   | -c <i>arg</i> or -c  |
| System supports:                     | -a <i>arg</i> and -a <i>arg</i> | -c <i>arg</i> and -c |
| Non-conforming applications may use: | -a <i>arg</i>                   | N/A                  |

127979 Earlier versions of this standard included obsolescent syntax which showed some options with  
 127980 (mandatory) adjacent option-arguments in the SYNOPSIS for some utilities. These have since  
 127981 been removed. For all options with mandatory option-arguments, the SYNOPSIS now shows  
 127982 <blank> characters between the option and the option-argument; however, historical usage has  
 127983 not been consistent in this area; therefore, <blank> characters are required to be used by  
 127984 conforming applications and to be handled by all implementations, but implementations are  
 127985 also required to handle an adjacent option-argument in order to preserve backwards-  
 127986 compatibility for old scripts. One of the justifications for selecting the multiple-argument  
 127987 method was that the single-argument case is inherently ambiguous when the option-argument  
 127988 can legitimately be a null string.

127989 POSIX.1-2024 explicitly states that digits are permitted as operands and option-arguments. The  
 127990 lower and upper bounds for the values of the numbers used for operands and option-arguments  
 127991 were derived from the ISO C standard values for {LONG\_MIN} and {LONG\_MAX}. The  
 127992 requirement on the standard utilities is that numbers in the specified range do not cause a  
 127993 syntax error, although the specification of a number need not be semantically correct for a  
 127994 particular operand or option-argument of a utility. For example, the specification of:

127995 `dd obs=3000000000`

127996 would yield undefined behavior for the application and could be a syntax error because the  
 127997 number 3 000 000 000 is outside of the range -2 147 483 647 to +2 147 483 647. On the other hand:

127998 `dd obs=2000000000`

127999 may cause some error, such as “blocksize too large”, rather than a syntax error.

128000 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0056 [584] and XBD/TC2-2008/0057  
 128001 [813] are applied.

128002 Austin Group Defect 1062 is applied, correcting the spacing in some example SYNOPSIS lines.

## 128003 A.12.2 Utility Syntax Guidelines

128004 This section is based on the rules listed in the SVID. It was included for two reasons:

- 128005 1. The individual utility descriptions in XCU [Chapter 3](#) (on page 2573) needed a set of  
 128006 common (although not universal) actions on which they could anchor their descriptions  
 128007 of option and operand syntax. Most of the standard utilities actually do use these  
 128008 guidelines, and many of their historical implementations use the *getopt()* function for  
 128009 their parsing. Therefore, it was simpler to cite the rules and merely identify exceptions.
- 128010 2. Developers of conforming applications need suggested guidelines if the POSIX  
 128011 community is to avoid the chaos of historical UNIX system command syntax.

128012 It is recommended that all *future* utilities and applications use these guidelines to enhance “user

128013 portability". The fact that some historical utilities could not be changed (to avoid breaking  
128014 historical applications) should not deter this future goal.

128015 The voluntary nature of the guidelines is highlighted by repeated uses of the word *should*  
128016 throughout. This usage should not be misinterpreted to imply that utilities that claim  
128017 conformance in their OPTIONS sections do not always conform.

128018 Guidelines 1 and 2 encourage utility writers to use only characters from the portable character  
128019 set because use of locale-specific characters may make the utility inaccessible from other locales.  
128020 Use of uppercase letters is discouraged due to problems associated with porting utilities to  
128021 systems that do not distinguish between uppercase and lowercase characters in filenames. Use  
128022 of non-alphanumeric characters is discouraged due to the number of utilities that treat non-  
128023 alphanumeric characters in "special" ways depending on context (such as the shell using white-  
128024 space characters to delimit arguments, various quote characters for quoting, the <dollar-sign> to  
128025 introduce variable expansion, etc.).

128026 In XCU Section 2.9.1 (on page 2500), it is further stated that a command used in the Shell  
128027 Command Language cannot be named with a trailing <colon>.

128028 Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the  
128029 character set to allow compatibility with historical usage. Historical practice allows the use of  
128030 digits wherever practical, and there are no portability issues that would prohibit the use of  
128031 digits. In fact, from an internationalization viewpoint, digits (being non-language-dependent)  
128032 are preferable over letters (a -2 is intuitively self-explanatory to any user, while in the -f filename  
128033 the letter 'f' is a mnemonic aid only to speakers of Latin-based languages where "filename"  
128034 happens to translate to a word that begins with 'f'). Since Guideline 3 still retains the word  
128035 "single", multi-digit options are not allowed. Instances of historical utilities that used them have  
128036 been marked obsolescent, with the numbers being changed from option names to option-  
128037 arguments.

128038 It was difficult to achieve a satisfactory solution to the problem of name space in option  
128039 characters. When the standard developers desired to extend the historical *cc* utility to accept  
128040 ISO C standard programs, they found that all of the portable alphabet was already in use by  
128041 various vendors. Thus, they had to devise a new name, *c89* (subsequently superseded by *c99*  
128042 and now by *c17*), rather than something like *cc -X*. There were suggestions that implementors  
128043 be restricted to providing extensions through various means (such as using a <plus-sign> as the  
128044 option delimiter or using option characters outside the alphanumeric set) that would reserve all  
128045 of the remaining alphanumeric characters for future POSIX standards. These approaches were  
128046 resisted because they lacked the historical style of UNIX systems. Furthermore, if a vendor-  
128047 provided option should become commonly used in the industry, it would be a candidate for  
128048 standardization. It would be desirable to standardize such a feature using historical practice for  
128049 the syntax (the semantics can be standardized with any syntax). This would not be possible if  
128050 the syntax was one reserved for the vendor. However, since the standardization process may  
128051 lead to minor changes in the semantics, it may prove to be better for a vendor to use a syntax  
128052 that will not be affected by standardization.

128053 Guideline 8 includes the concept of <comma>-separated lists in a single argument. It is up to the  
128054 utility to parse such a list itself because *getopt()* just returns the single string. This situation was  
128055 retained so that certain historical utilities would not violate the guidelines. Applications  
128056 preparing for international use should be aware of an occasional problem with  
128057 <comma>-separated lists: in some locales, the <comma> is used as the radix character. Thus, if  
128058 an application is preparing operands for a utility that expects a <comma>-separated list, it  
128059 should avoid generating non-integer values through one of the means that is influenced by  
128060 setting the *LC\_NUMERIC* variable (such as *awk*, *bc*, *printf*, or *printf()*).

128061 Unless explicitly stated otherwise in the utility description, Guideline 9 requires applications to

128062 put options before operands, and requires utilities to accept any such usage without  
 128063 misinterpreting operands as options. For example, if an implementation of the *printf* utility  
 128064 supports a *-e* option as an extension, the command:

```
128065 printf %s -e
```

128066 must output the string "-e" without interpreting the *-e* as an option. Similarly, the command:

```
128067 ls myfile -l
```

128068 must interpret the *-l* argument as a second file operand, not as a *-l* option.

128069 Applications calling any utility with a first operand starting with '-' should usually specify *--*,  
 128070 as indicated by Guideline 10, to mark the end of the options. This is true even if the SYNOPSIS  
 128071 in the Shell and Utilities volume of POSIX.1-2024 does not specify any options; implementations  
 128072 may provide options as extensions to the Shell and Utilities volume of POSIX.1-2024. The  
 128073 standard utilities that do not support Guideline 10 indicate that fact in the OPTIONS section of  
 128074 the utility description.

128075 Guideline 7 allows any string to be an option-argument; an option-argument can begin with any  
 128076 character, can be *-* or *--*, and can be an empty string. For example, the commands *pr -h -*, *pr -h*  
 128077 *--*, *pr -h -d*, *pr -h +2*, and *pr -h ''* contain the option-arguments *-*, *--*, *-d*, *+2*, and an empty  
 128078 string, respectively. Conversely, the command *pr -h -- -d* treats *-d* as an option, not as an  
 128079 argument, because the *—* is an option-argument here, not a delimiter.

128080 Guideline 11 was modified to clarify that the order of different options should not matter  
 128081 relative to one another. However, the order of repeated options that also have option-arguments  
 128082 may be significant; therefore, such options are required to be interpreted in the order that they  
 128083 are specified. The *make* utility is an instance of a historical utility that uses repeated options in  
 128084 which the order is significant. Multiple files are specified by giving multiple instances of the *-f*  
 128085 option; for example:

```
128086 make -f common_header -f specific_rules target
```

128087 Guideline 13 does not imply that all of the standard utilities automatically accept the operand  
 128088 '-' to mean standard input or output, nor does it specify the actions of the utility upon  
 128089 encountering multiple '-' operands. It simply says that, by default, '-' operands are not used  
 128090 for other purposes in the file reading or writing (but not when using *stat()*, *unlink()*, *touch*, and  
 128091 so on) utilities. In earlier versions of this standard, all information concerning actual treatment of  
 128092 the '-' operand is found in the individual utility sections. Many implementations, however,  
 128093 treated '-' as standard input or output and many applications depended on this behavior even  
 128094 though it was not standard. This behavior is now implementation-defined. Portable applications  
 128095 should not use '-' to mean standard input or output unless it is explicitly stated to do so in the  
 128096 utility description and they should always use './-' if they intend to refer to a file named *-* in  
 128097 the current working directory.

128098 Guideline 14 is intended to prohibit implementations that would treat the command *ls -l -d* as if  
 128099 it were *ls -- -l -d* or *ls -l -- -d*.

128100 The standard permits implementations to have extensions that violate the Utility Syntax  
 128101 Guidelines so long as when the utility is used in line with the forms defined by the standard it  
 128102 follows the Utility Syntax Guidelines. Thus, *head-42file* and *ls--help* are permitted extensions.  
 128103 The intent is to allow extensions so long as the standard form is accepted and follows the  
 128104 guidelines.

128105 An area of concern was that as implementations mature, implementation-defined utilities and  
 128106 implementation-defined utility options will result. The idea was expressed that there needed to  
 128107 be a standard way, say an environment variable or some such mechanism, to identify  
 128108 implementation-defined utilities separately from standard utilities that may have the same

128109 name. It was decided that there already exist several ways of dealing with this situation and that  
 128110 it is outside of the scope to attempt to standardize in the area of non-standard items. A method  
 128111 that exists on some historical implementations is the use of the so-called **/local/bin** or  
 128112 **/usr/local/bin** directory to separate local or additional copies or versions of utilities. Another  
 128113 method that is also used is to isolate utilities into completely separate domains. Still another  
 128114 method to ensure that the desired utility is being used is to request the utility by its full  
 128115 pathname. There are many approaches to this situation; the examples given above serve to  
 128116 illustrate that there is more than one.

## 128117 **A.13 Namespace and Future Directions**

128118 **A.14** Austin Group Defect 1071 is applied, adding this chapter.

## **Headers**

### 128119 **A.14.1 Format of Entries**

128120 Each header reference page has a common layout of sections describing the interface. This  
 128121 layout is similar to the manual page or “man” page format shipped with most UNIX systems,  
 128122 and each header has sections describing the SYNOPSIS and DESCRIPTION. These are the two  
 128123 sections that relate to conformance.

128124 Additional sections are informative, and add considerable information for the application  
 128125 developer. APPLICATION USAGE sections provide additional caveats, issues, and  
 128126 recommendations to the developer. RATIONALE sections give additional information on the  
 128127 decisions made in defining the interface.

128128 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
 128129 the future, and often cautions the developer to architect the code to account for a change in this  
 128130 area. Note that a future directions statement should not be taken as a commitment to adopt a  
 128131 feature or interface in the future.

128132 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
 128133 changed.

128134 Option labels and margin markings in the page can be useful in guiding the application  
 128135 developer.

### 128136 **A.14.2 Removed Headers in Issue 8**

128137 The headers removed in Issue 8 (from the Issue 7 base document) are as follows:

128138


128139

128140

| Removed Headers in Issue 8 |           |
|----------------------------|-----------|
| <stropts.h>                | <trace.h> |
| <ulimit.h>                 | <utime.h> |



128141

 *Rationale (Informative)*

128142

**Part B:**

128143

**System Interfaces**

128144

*The Open Group*

128145

*The Institute of Electrical and Electronics Engineers, Inc.*



# Rationale for System Interfaces

## 128148 **B.1 Introduction**

### 128149 **B.1.1 Change History**

128150 The change history is provided as an informative section, to track changes from earlier versions  
128151 of this standard.

128152 The following sections describe changes made to the System Interfaces volume of POSIX.1-2024  
128153 since Issue 7 of the base document. The CHANGE HISTORY section for each entry details the  
128154 technical changes that have been made in Issue 5 and later. Changes made before Issue 5 are not  
128155 included.

#### 128156 **Changes from Issue 7 to Issue 8 (POSIX.1-2024)**

128157 The following list summarizes the major changes that were made in the System Interfaces  
128158 volume of POSIX.1-2024 from Issue 7 to Issue 8:

- 128159 • The Open Group Standard, 2021, Additional APIs for the Base Specifications Issue 8, Part 1  
128160 is incorporated.
- 128161 • The Open Group Standard, 2022, Additional APIs for the Base Specifications Issue 8, Part 2  
128162 is incorporated.
- 128163 • IEEE Std 1003.26-2003 is incorporated.
- 128164 • Existing functionality is aligned with the ISO/IEC 9899:2018 standard.
- 128165 • New functionality from the ISO/IEC 9899:2018 standard is incorporated.
- 128166 • Austin Group defect reports and IEEE Interpretations against IEEE Std 1003.1 are applied.
- 128167 • The Open Group corrigenda and resolutions are applied.
- 128168 • Features, marked obsolescent in the base document, have been considered for removal in  
128169 this version.
- 128170 • The Device Control option is added.
- 128171 • The IEC 60559 Complex Floating-Point option is added.

128172 **New Features in Issue 8**

128173 The functions first introduced in Issue 8 (over the Issue 7 base document) are as follows:

| New Functions in Issue 8 |                                                        |                                           |
|--------------------------|--------------------------------------------------------|-------------------------------------------|
| 128174                   |                                                        |                                           |
| 128175                   | <code>_Fork()</code>                                   | <code>c32rtomb()</code>                   |
| 128176                   | <code>aligned_alloc()</code>                           | <code>call_once()</code>                  |
| 128177                   | <code>at_quick_exit()</code>                           | <code>cnd_broadcast()</code>              |
| 128178                   | <code>atomic_compare_exchange_strong()</code>          | <code>cnd_destroy()</code>                |
| 128179                   | <code>atomic_compare_exchange_strong_explicit()</code> | <code>cnd_init()</code>                   |
| 128180                   | <code>atomic_compare_exchange_weak()</code>            | <code>cnd_signal()</code>                 |
| 128181                   | <code>atomic_compare_exchange_weak_explicit()</code>   | <code>cnd_timedwait()</code>              |
| 128182                   | <code>atomic_exchange()</code>                         | <code>cnd_wait()</code>                   |
| 128183                   | <code>atomic_exchange_explicit()</code>                | <code>dcgettext()</code>                  |
| 128184                   | <code>atomic_fetch_add()</code>                        | <code>dcgettext_l()</code>                |
| 128185                   | <code>atomic_fetch_add_explicit()</code>               | <code>dcngettext()</code>                 |
| 128186                   | <code>atomic_fetch_and()</code>                        | <code>dcngettext_l()</code>               |
| 128187                   | <code>atomic_fetch_and_explicit()</code>               | <code>dgettext()</code>                   |
| 128188                   | <code>atomic_fetch_or()</code>                         | <code>dgettext_l()</code>                 |
| 128189                   | <code>atomic_fetch_or_explicit()</code>                | <code>dladdr()</code>                     |
| 128190                   | <code>atomic_fetch_sub()</code>                        | <code>dngettext()</code>                  |
| 128191                   | <code>atomic_fetch_sub_explicit()</code>               | <code>dngettext_l()</code>                |
| 128192                   | <code>atomic_fetch_xor()</code>                        | <code>getentropy()</code>                 |
| 128193                   | <code>atomic_fetch_xor_explicit()</code>               | <code>getlocalename_l()</code>            |
| 128194                   | <code>atomic_flag_clear()</code>                       | <code>getresgid()</code>                  |
| 128195                   | <code>atomic_flag_clear_explicit()</code>              | <code>getresuid()</code>                  |
| 128196                   | <code>atomic_flag_test_and_set()</code>                | <code>gettext()</code>                    |
| 128197                   | <code>atomic_flag_test_and_set_explicit()</code>       | <code>gettext_l()</code>                  |
| 128198                   | <code>atomic_init()</code>                             | <code>mbrtoc16()</code>                   |
| 128199                   | <code>atomic_is_lock_free()</code>                     | <code>mbrtoc32()</code>                   |
| 128200                   | <code>atomic_load()</code>                             | <code>memmem()</code>                     |
| 128201                   | <code>atomic_load_explicit()</code>                    | <code>mtx_destroy()</code>                |
| 128202                   | <code>atomic_signal_fence()</code>                     | <code>mtx_init()</code>                   |
| 128203                   | <code>atomic_store()</code>                            | <code>mtx_lock()</code>                   |
| 128204                   | <code>atomic_store_explicit()</code>                   | <code>mtx_timedlock()</code>              |
| 128205                   | <code>atomic_thread_fence()</code>                     | <code>mtx_trylock()</code>                |
| 128206                   | <code>bind_textdomain_codeset()</code>                 | <code>mtx_unlock()</code>                 |
| 128207                   | <code>bindtextdomain()</code>                          | <code>ngettext()</code>                   |
| 128208                   | <code>c16rtomb()</code>                                | <code>ngettext_l()</code>                 |
|                          |                                                        | <code>posix_close()</code>                |
|                          |                                                        | <code>posix_devctl()</code>               |
|                          |                                                        | <code>posix_getdents()</code>             |
|                          |                                                        | <code>ppoll()</code>                      |
|                          |                                                        | <code>pthread_cond_clockwait()</code>     |
|                          |                                                        | <code>pthread_mutex_clocklock()</code>    |
|                          |                                                        | <code>pthread_rwlock_clockrdlock()</code> |
|                          |                                                        | <code>pthread_rwlock_clockwrlock()</code> |
|                          |                                                        | <code>qsort_r()</code>                    |
|                          |                                                        | <code>quick_exit()</code>                 |
|                          |                                                        | <code>reallocarray()</code>               |
|                          |                                                        | <code>sem_clockwait()</code>              |
|                          |                                                        | <code>setresgid()</code>                  |
|                          |                                                        | <code>setresuid()</code>                  |
|                          |                                                        | <code>sig2str()</code>                    |
|                          |                                                        | <code>str2sig()</code>                    |
|                          |                                                        | <code>strlcat()</code>                    |
|                          |                                                        | <code>strncpy()</code>                    |
|                          |                                                        | <code>textdomain()</code>                 |
|                          |                                                        | <code>thrd_create()</code>                |
|                          |                                                        | <code>thrd_current()</code>               |
|                          |                                                        | <code>thrd_detach()</code>                |
|                          |                                                        | <code>thrd_equal()</code>                 |
|                          |                                                        | <code>thrd_exit()</code>                  |
|                          |                                                        | <code>thrd_join()</code>                  |
|                          |                                                        | <code>thrd_sleep()</code>                 |
|                          |                                                        | <code>thrd_yield()</code>                 |
|                          |                                                        | <code>timespec_get()</code>               |
|                          |                                                        | <code>tss_create()</code>                 |
|                          |                                                        | <code>tss_delete()</code>                 |
|                          |                                                        | <code>tss_get()</code>                    |
|                          |                                                        | <code>tss_set()</code>                    |
|                          |                                                        | <code>wcslcat()</code>                    |
|                          |                                                        | <code>wcsncpy()</code>                    |

128209 The following new headers are introduced in Issue 8:

| New Headers in Issue 8 |                                 |                                    |
|------------------------|---------------------------------|------------------------------------|
| 128210                 |                                 |                                    |
| 128211                 | <code>&lt;devctl.h&gt;</code>   | <code>&lt;stdatomic.h&gt;</code>   |
| 128212                 | <code>&lt;libintl.h&gt;</code>  | <code>&lt;stdnoreturn.h&gt;</code> |
| 128213                 | <code>&lt;stdalign.h&gt;</code> | <code>&lt;threads.h&gt;</code>     |
|                        |                                 | <code>&lt;uchar.h&gt;</code>       |

128214 **Obsolescent Functions in Issue 8**

128215 The base functions moved to obsolescent status in Issue 8 (from the Issue 7 base document) are  
128216 as follows:

128217

128218

|                                              |  |
|----------------------------------------------|--|
| <b>Obsolescent Base Functions in Issue 8</b> |  |
|----------------------------------------------|--|

|                    |                    |
|--------------------|--------------------|
| <i>inet_addr()</i> | <i>inet_ntoa()</i> |
|--------------------|--------------------|

128219 The XSI functions moved to obsolescent status in Issue 8 (from the Issue 7 base document) are as  
128220 follows:

128221

128222

|                                             |  |
|---------------------------------------------|--|
| <b>Obsolescent XSI Functions in Issue 8</b> |  |
|---------------------------------------------|--|

|                  |                 |
|------------------|-----------------|
| <i>encrypt()</i> | <i>setkey()</i> |
|------------------|-----------------|

128223 **Removed Functions in Issue 8**

128224 The functions removed in Issue 8 (from the Issue 7 base document) are as follows:

128225

**Removed Functions in Issue 8**

|        |                                                     |                                               |
|--------|-----------------------------------------------------|-----------------------------------------------|
| 128226 | <i>_longjmp()</i>                                   | <i>posix_trace_eventid_equal()</i>            |
| 128227 | <i>_setjmp()</i>                                    | <i>posix_trace_eventid_get_name()</i>         |
| 128228 | <i>_tolower()</i>                                   | <i>posix_trace_eventid_open()</i>             |
| 128229 | <i>_toupper()</i>                                   | <i>posix_trace_eventset_add()</i>             |
| 128230 | <i>fattach()</i>                                    | <i>posix_trace_eventset_del()</i>             |
| 128231 | <i>fdetach()</i>                                    | <i>posix_trace_eventset_empty()</i>           |
| 128232 | <i>ftw()</i>                                        | <i>posix_trace_eventset_fill()</i>            |
| 128233 | <i>getitimer()</i>                                  | <i>posix_trace_eventset_ismember()</i>        |
| 128234 | <i>getmsg()</i>                                     | <i>posix_trace_eventtypelist_getnext_id()</i> |
| 128235 | <i>getpmsg()</i>                                    | <i>posix_trace_eventtypelist_rewind()</i>     |
| 128236 | <i>gets()</i>                                       | <i>posix_trace_flush()</i>                    |
| 128237 | <i>gettimeofday()</i>                               | <i>posix_trace_get_attr()</i>                 |
| 128238 | <i>ioctl()</i>                                      | <i>posix_trace_get_filter()</i>               |
| 128239 | <i>isascii()</i>                                    | <i>posix_trace_get_status()</i>               |
| 128240 | <i>isastream()</i>                                  | <i>posix_trace_getnext_event()</i>            |
| 128241 | <i>posix_trace_attr_destroy()</i>                   | <i>posix_trace_open()</i>                     |
| 128242 | <i>posix_trace_attr_getclockres()</i>               | <i>posix_trace_rewind()</i>                   |
| 128243 | <i>posix_trace_attr_getcreatetime()</i>             | <i>posix_trace_set_filter()</i>               |
| 128244 | <i>posix_trace_attr_getgenversion()</i>             | <i>posix_trace_shutdown()</i>                 |
| 128245 | <i>posix_trace_attr_getinherited()</i>              | <i>posix_trace_start()</i>                    |
| 128246 | <i>posix_trace_attr_getlogfullpolicy()</i>          | <i>posix_trace_stop()</i>                     |
| 128247 | <i>posix_trace_attr_getlogsize()</i>                | <i>posix_trace_timedgetnext_event()</i>       |
| 128248 | <i>posix_trace_attr_getmaxdatasize()</i>            | <i>posix_trace_trid_eventid_open()</i>        |
| 128249 | <i>posix_trace_attr_getmaxsystemeventsizesize()</i> | <i>posix_trace_trygetnext_event()</i>         |
| 128250 | <i>posix_trace_attr_getmaxusereventsizesize()</i>   | <i>pthread_getconcurrency()</i>               |
| 128251 | <i>posix_trace_attr_getname()</i>                   | <i>pthread_setconcurrency()</i>               |
| 128252 | <i>posix_trace_attr_getstreamfullpolicy()</i>       | <i>putmsg()</i>                               |
| 128253 | <i>posix_trace_attr_getstreamsize()</i>             | <i>putpmsg()</i>                              |
| 128254 | <i>posix_trace_attr_init()</i>                      | <i>rand_r()</i>                               |
| 128255 | <i>posix_trace_attr_setinherited()</i>              | <i>setitimer()</i>                            |
| 128256 | <i>posix_trace_attr_setlogfullpolicy()</i>          | <i>setpgrp()</i>                              |
| 128257 | <i>posix_trace_attr_setlogsize()</i>                | <i>sighold()</i>                              |
| 128258 | <i>posix_trace_attr_setmaxdatasize()</i>            | <i>sigignore()</i>                            |
| 128259 | <i>posix_trace_attr_setname()</i>                   | <i>siginterrupt()</i>                         |
| 128260 | <i>posix_trace_attr_setstreamfullpolicy()</i>       | <i>sigpause()</i>                             |
| 128261 | <i>posix_trace_attr_setstreamsize()</i>             | <i>sigrelse()</i>                             |
| 128262 | <i>posix_trace_clear()</i>                          | <i>sigset()</i>                               |
| 128263 | <i>posix_trace_close()</i>                          | <i>tempnam()</i>                              |
| 128264 | <i>posix_trace_create()</i>                         | <i>toascii()</i>                              |
| 128265 | <i>posix_trace_create_withlog()</i>                 | <i>ulimit()</i>                               |
| 128266 | <i>posix_trace_event()</i>                          | <i>utime()</i>                                |

## 128267 **B.1.2 Relationship to Other Formal Standards**

128268 There is no additional rationale provided for this section.

## 128269 **B.1.3 Format of Entries**

128270 Each system interface reference page has a common layout of sections describing the interface.  
 128271 This layout is similar to the manual page or “man” page format shipped with most UNIX  
 128272 systems, and each header has sections describing the SYNOPSIS, DESCRIPTION, RETURN  
 128273 VALUE, and ERRORS. These are the four sections that relate to conformance.

128274 Additional sections are informative, and add considerable information for the application  
 128275 developer. EXAMPLES sections provide example usage. APPLICATION USAGE sections  
 128276 provide additional caveats, issues, and recommendations to the developer. RATIONALE  
 128277 sections give additional information on the decisions made in defining the interface.

128278 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
 128279 the future, and often cautions the developer to architect the code to account for a change in this  
 128280 area. Note that a future directions statement should not be taken as a commitment to adopt a  
 128281 feature or interface in the future.

128282 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
 128283 changed.

128284 Option labels and margin markings in the page can be useful in guiding the application  
 128285 developer.

## 128286 **B.2 General Information**

### 128287 **B.2.1 Use and Implementation of Interfaces**

#### 128288 *B.2.1.1 Use and Implementation of Functions*

128289 The information concerning the use of functions was adapted from a description in the ISO C  
 128290 standard. Here is an example of how an application program can protect itself from functions  
 128291 that may or may not be macros, rather than true functions:

128292 The *atoi()* function may be used in any of several ways:

- 128293 • By use of its associated header (possibly generating a macro expansion):

```
128294 #include <stdlib.h>
128295 /* ... */
128296 i = atoi(str);
```

- 128297 • By use of its associated header (assuredly generating a true function call):

```
128298 #include <stdlib.h>
128299 #undef atoi
128300 /* ... */
128301 i = atoi(str);
```

128302           or:

128303           #include <stdlib.h>

128304           /\* ... \*/

128305           i = (atoi) (str);

128306           • By explicit declaration:

128307           extern int atoi (const char \*);

128308           /\* ... \*/

128309           i = atoi(str);

128310           • By implicit declaration:

128311           /\* ... \*/

128312           i = atoi(str);

128313           (Assuming no function prototype is in scope. This is not allowed by the ISO C standard for

128314           functions with variable arguments; furthermore, parameter type conversion “widening” is

128315           subject to different rules in this case.)

128316           Note that the ISO C standard reserves names starting with ‘\_’ for the compiler. Therefore, the

128317           compiler could, for example, implement an intrinsic, built-in function *\_asm\_builtin\_atoi()*, which

128318           it recognized and expanded into inline assembly code. Then, in <stdlib.h>, there could be the

128319           following:

128320           #define atoi(X) \_asm\_builtin\_atoi(X)

128321           The user’s “normal” call to *atoi()* would then be expanded inline, but the implementor would

128322           also be required to provide a callable function named *atoi()* for use when the application

128323           requires it; for example, if its address is to be stored in a function pointer variable.

128324           Implementors should note that since applications can **#undef** a macro in order to ensure that the

128325           function is used, this means that it is not safe for implementations to use the names of any

128326           standard functions in macro values, since the application could use **#undef** to ensure that no

128327           macro exists and then use the same name for an identifier with local scope. For example,

128328           historically it was common for a *getchar()* macro to be defined in <stdio.h> as:

128329           #define getchar() getc(stdin)

128330           This definition does not conform, because an application is allowed to use the identifier *getc*

128331           with local scope, and the expansion of the *getchar()* macro would then pick up the local *getc*.

128332           The following is conforming code, but would not compile with the above definition of *getchar()*:

128333           #include <stdio.h>

128334           #undef getc

128335           int main(void)

128336           {

128337            int getc;

128338            getc = getchar();

128339            return getc;

128340           }

128341           This does not only affect function-like macros. For example, the following definition does not

128342           conform because there could be a local *sysconf* variable in scope when SIGRTMIN is expanded:

128343           #define SIGRTMIN ((int)sysconf(\_SC\_SIGRT\_MIN))

128344           Implementors can avoid the problem by using aliases for standard functions instead of the



128345 actual function, with names that conforming applications cannot use for local variables. For  
128346 example:

```
128347 #define SIGRTMIN ((int)__sysconf(_SC_SIGRT_MIN))
```

128348 Austin Group Defect 655 is applied, making the requirement relating to explicit function  
128349 declarations apply only to functions from the ISO C standard.

128350 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

128351 Austin Group Defect 1404 is applied, adding to the examples of invalid values for function  
128352 arguments.

128353 *B.2.1.2 Use and Implementation of Macros*

128354 There is no additional rationale provided for this section.

## 128355 **B.2.2 The Compilation Environment**

128356 *B.2.2.1 POSIX.1 Symbols*

128357 This and the following section address the issue of “name space pollution”. The ISO C standard  
128358 requires that the name space beyond what it reserves not be altered except by explicit action of  
128359 the application developer. This section defines the actions to add the POSIX.1 symbols for those  
128360 headers where both the ISO C standard and POSIX.1 need to define symbols, and also where the  
128361 XSI option extends the base standard.

128362 When headers are used to provide symbols, there is a potential for introducing symbols that the  
128363 application developer cannot predict. Ideally, each header should only contain one set of  
128364 symbols, but this is not practical for historical reasons. Thus, the concept of feature test macros is  
128365 included. Two feature test macros are explicitly defined by POSIX.1-2024; it is expected that  
128366 future versions may add to this.

128367 **Note:** Feature test macros allow an application to announce to the implementation its desire to have  
128368 certain symbols and prototypes exposed. They should not be confused with the version test  
128369 macros and constants for options in `<unistd.h>` which are the implementation’s way of  
128370 announcing functionality to the application.

128371 It is further intended that these feature test macros apply only to the headers specified by  
128372 POSIX.1-2024. Implementations are expressly permitted to make visible symbols not specified  
128373 by POSIX.1-2024, within both POSIX.1 and other headers, under the control of feature test  
128374 macros that are not defined by POSIX.1-2024.

### 128375 **The `_POSIX_C_SOURCE` Feature Test Macro**

128376 The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been  
128377 superseded by `_POSIX_C_SOURCE`. This symbol will allow implementations to support various  
128378 versions of this standard simultaneously. For instance, when `_POSIX_C_SOURCE` is defined as  
128379 `202405L`, the system should make visible the same name space as permitted and required by the  
128380 POSIX.1-2024 standard. A special case is the one where the implementation wishes to make  
128381 available support for the 1990 version of the POSIX standard, in which instance when either  
128382 `_POSIX_SOURCE` is defined or `_POSIX_C_SOURCE` is defined as `1`, the system should make  
128383 visible the same name space as permitted and required by the POSIX.1-1990 standard.

128384 It is expected that C bindings to future POSIX standards will define new values for

128385 `_POSIX_C_SOURCE`, with each new value reserving the name space for that new standard.

#### 128386 **The `_XOPEN_SOURCE` Feature Test Macro**

128387 The feature test macro `_XOPEN_SOURCE` is provided as the announcement mechanism for the  
128388 application that it requires functionality from the Single UNIX Specification. `_XOPEN_SOURCE`  
128389 must be defined to the value 800 before the inclusion of any header to enable the functionality in  
128390 the Single UNIX Specification Version 5. Its definition subsumes the use of `_POSIX_C_SOURCE`.

128391 An extract of code from a conforming application, that appears before any **#include** statements,  
128392 is given below:

```
128393 #define _XOPEN_SOURCE 800 /* Single UNIX Specification, Version 5 */
128394 #include ...
```

128395 Note that the definition of `_XOPEN_SOURCE` with the value 800 makes the definition of  
128396 `_POSIX_C_SOURCE` redundant and it can safely be omitted.

#### 128397 **The `__STDC_WANT_LIB_EXT1__` Feature Test Macro**

128398 The ISO C standard specifies the feature test macro `__STDC_WANT_LIB_EXT1__` as the  
128399 announcement mechanism for the application that it requires functionality from Annex K. It  
128400 specifies that the symbols specified in Annex K (if supported) are made visible when  
128401 `__STDC_WANT_LIB_EXT1__` is 1 and are not made visible when it is 0, but leaves it unspecified  
128402 whether they are made visible when `__STDC_WANT_LIB_EXT1__` is undefined. POSIX.1  
128403 requires that they are not made visible when the macro is undefined (except for those symbols  
128404 that are already explicitly allowed to be visible through the definition of `_POSIX_C_SOURCE` or  
128405 `_XOPEN_SOURCE`, or both).

128406 POSIX.1 does not include the interfaces specified in Annex K of the ISO C standard, but allows  
128407 the symbols to be made visible in headers when requested by the application in order that  
128408 applications can use symbols from Annex K and symbols from POSIX.1 in the same translation  
128409 unit.

128410 Austin Group Defect 1302 is applied, adding this subsection.

#### 128411 *B.2.2.2 The Name Space*

128412 The reservation of identifiers is paraphrased from the ISO C standard. The text is included  
128413 because it needs to be part of POSIX.1-2024, regardless of possible changes in future versions of  
128414 the ISO C standard.

128415 These identifiers may be used by implementations, particularly for feature test macros.  
128416 Implementations should not use feature test macro names that might be reasonably used by a  
128417 standard.

128418 Including headers more than once is a reasonably common practice, and it should be carried  
128419 forward from the ISO C standard. More significantly, having definitions in more than one  
128420 header is explicitly permitted. Where the potential declaration is “benign” (the same definition  
128421 twice) the declaration can be repeated, if that is permitted by the compiler. (This is usually true  
128422 of macros, for example.) In those situations where a repetition is not benign (for example,  
128423 **typedefs**), conditional compilation must be used. The situation actually occurs both within the  
128424 ISO C standard and within POSIX.1: **time\_t** should be in `<sys/types.h>`, and the ISO C standard  
128425 mandates that it be in `<time.h>`.

128426 The area of name space pollution *versus* additions to structures is difficult because of the macro  
128427 structure of C. The following discussion summarizes all the various problems with and

128428 objections to the issue.

128429 Note the phrase “user-defined macro”. Users are not permitted to define macro names (or any  
128430 other name) beginning with “\_[A-Z\_]”. Thus, the conflict cannot occur for symbols reserved  
128431 to the vendor’s name space, and the permission to add fields automatically applies, without  
128432 qualification, to those symbols.

128433 1. Data structures (and unions) need to be defined in headers by implementations to meet  
128434 certain requirements of POSIX.1 and the ISO C standard.

128435 2. The structures defined by POSIX.1 are typically minimal, and any practical  
128436 implementation would wish to add fields to these structures either to hold additional  
128437 related information or for backwards-compatibility (or both). Future standards (and *de*  
128438 *facto* standards) would also wish to add to these structures. Issues of field alignment  
128439 make it impractical (at least in the general case) to simply omit fields when they are not  
128440 defined by the particular standard involved.

128441 The **dirent** structure is an example of such a minimal structure (although one could argue  
128442 about whether the other fields need visible names). The *st\_rdev* field of most  
128443 implementations’ **stat** structure is a common example where extension is needed and  
128444 where a conflict could occur.

128445 3. Fields in structures are in an independent name space, so the addition of such fields  
128446 presents no problem to the C language itself in that such names cannot interact with  
128447 identically named user symbols because access is qualified by the specific structure name.

128448 4. There is an exception to this: macro processing is done at a lexical level. Thus, symbols  
128449 added to a structure might be recognized as user-provided macro names at the location  
128450 where the structure is declared. This only can occur if the user-provided name is declared  
128451 as a macro before the header declaring the structure is included. The user’s use of the  
128452 name after the declaration cannot interfere with the structure because the symbol is  
128453 hidden and only accessible through access to the structure. Presumably, the user would  
128454 not declare such a macro if there was an intention to use that field name.

128455 5. Macros from the same or a related header might use the additional fields in the structure,  
128456 and those field names might also collide with user macros. Although this is a less  
128457 frequent occurrence, since macros are expanded at the point of use, no constraint on the  
128458 order of use of names can apply.

128459 6. An “obvious” solution of using names in the reserved name space and then redefining  
128460 them as macros when they should be visible does not work because this has the effect of  
128461 exporting the symbol into the general name space. For example, given a (hypothetical)  
128462 system-provided header **<h.h>**, and two parts of a C program in **a.c** and **b.c**, in header  
128463 **<h.h>**:

```
128464 struct foo {
128465     int __i;
128466 }

128467 #ifdef _FEATURE_TEST
128468 #define i __i;
128469 #endif
```

128470 In file **a.c**:

```
128471 #include h.h
128472 extern int i;
128473 ...
```

128474 In file **b.c**:  
 128475 `extern int i;`  
 128476 `...`

128477 The symbol that the user thinks of as *i* in both files has an external name of `__i` in **a.c**; the  
 128478 same symbol *i* in **b.c** has an external name *i* (ignoring any hidden manipulations the  
 128479 compiler might perform on the names). This would cause a mysterious name resolution  
 128480 problem when **a.o** and **b.o** are linked.

128481 Simply avoiding definition then causes alignment problems in the structure.

128482 A structure of the form:

```
128483 struct foo {
128484     union {
128485         int __i;
128486 #ifdef _FEATURE_TEST
128487         int i;
128488 #endif
128489     } __ii;
128490 }
```

128491 does not work because the name of the logical field *i* is `__ii.i`, and introduction of a macro  
 128492 to restore the logical name immediately reintroduces the problem discussed previously  
 128493 (although its manifestation might be more immediate because a syntax error would result  
 128494 if a recursive macro did not cause it to fail first).

128495 7. A more workable solution would be to declare the structure:

```
128496 struct foo {
128497 #ifdef _FEATURE_TEST
128498     int i;
128499 #else
128500     int __i;
128501 #endif
128502 }
```

128503 However, if a macro (particularly one required by a standard) is to be defined that uses  
 128504 this field, two must be defined: one that uses *i*, the other that uses `__i`. If more than one  
 128505 additional field is used in a macro and they are conditional on distinct combinations of  
 128506 features, the complexity goes up as  $2^n$ .

128507 All this leaves a difficult situation: vendors must provide very complex headers to deal with  
 128508 what is conceptually simple and safe—adding a field to a structure. It is the possibility of user-  
 128509 provided macros with the same name that makes this difficult.

128510 Several alternatives were proposed that involved constraining the user's access to part of the  
 128511 name space available to the user (as specified by the ISO C standard). In some cases, this was  
 128512 only until all the headers had been included. There were two proposals discussed that failed to  
 128513 achieve consensus:

- 128514 1. Limiting it for the whole program.
- 128515 2. Restricting the use of identifiers containing only uppercase letters until after all system  
 128516 headers had been included. It was also pointed out that because macros might wish to  
 128517 access fields of a structure (and macro expansion occurs totally at point of use) restricting  
 128518 names in this way would not protect the macro expansion, and thus the solution was  
 128519 inadequate.

- 128520 It was finally decided that reservation of symbols would occur, but as constrained.
- 128521 The current wording also allows the addition of fields to a structure, but requires that user  
128522 macros of the same name not interfere. This allows vendors to do one of the following:
- 128523 • Not create the situation (do not extend the structures with user-accessible names or use the  
128524 solution in (7) above)
  - 128525 • Extend their compilers to allow some way of adding names to structures and macros safely
- 128526 There are at least two ways that the compiler might be extended: add new preprocessor  
128527 directives that turn off and on macro expansion for certain symbols (without changing the value  
128528 of the macro) and a function or lexical operation that suppresses expansion of a word. The latter  
128529 seems more flexible, particularly because it addresses the problem in macros as well as in  
128530 declarations.
- 128531 The following seems to be a possible implementation extension to the C language that will do  
128532 this: any token that during macro expansion is found to be preceded by three '#' symbols shall  
128533 not be further expanded in exactly the same way as described for macros that expand to their  
128534 own name as in Section 6.10.3.4 of the ISO C standard. A vendor may also wish to implement  
128535 this as an operation that is lexically a function, which might be implemented as:
- ```
128536 #define __safe_name(x) ###x
```
- 128537 Using a function notation would insulate vendors from changes in standards until such a  
128538 functionality is standardized (if ever). Standardization of such a function would be valuable  
128539 because it would then permit third parties to take advantage of it portably in software they may  
128540 supply.
- 128541 The symbols that are “explicitly permitted, but not required by POSIX.1-2024” include those  
128542 classified below. (That is, the symbols classified below might, but are not required to, be present  
128543 when `_POSIX_C_SOURCE` is defined to have the value 202405L.)
- 128544 • Symbols in `<limits.h>` and `<unistd.h>` that are defined to indicate support for options or  
128545 limits that are constant at compile-time
  - 128546 • Symbols in the name space reserved for the implementation by the ISO C standard
  - 128547 • Symbols in a name space reserved for a particular type of extension (for example, type  
128548 names ending with `_t` in `<sys/types.h>`)
  - 128549 • Additional members of structures or unions whose names do not reduce the name space  
128550 reserved for applications
- 128551 Since both implementations and future versions of this standard and other POSIX standards  
128552 may use symbols in the reserved spaces described in these tables, there is a potential for name  
128553 space clashes. To avoid future name space clashes when adding symbols, implementations  
128554 should not use the `posix_`, `POSIX_`, or `_POSIX_` prefixes.
- 128555 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/2 is applied, deleting the entries `POSIX_`,  
128556 `_POSIX_`, and `posix_` from the column of allowed name space prefixes for use by an  
128557 implementation in the first table. The presence of these prefixes was contradicting later text  
128558 which states that: “The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by XCU  
128559 [Chapter 2](#) (on page 2472) and other POSIX standards. Implementations may add symbols to the  
128560 headers shown in the following table, provided the identifiers ... do not use the reserved  
128561 prefixes `posix_`, `POSIX_`, or `_POSIX_`.”
- 128562 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/3 is applied, correcting the reserved  
128563 macro prefix from: “`PRI[a-z]`, `SCN[a-z]`” to: “`PRI[Xa-z]`, `SCN[Xa-z]`” in the second table. The  
128564 change was needed since the ISO C standard allows implementations to define macros of the

- 128565 form PRI or SCN followed by any lowercase letter or 'x' in `<inttypes.h>`. (The  
128566 ISO/IEC 9899:1999 standard, Subclause 7.26.4.)
- 128567 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/4 is applied, adding a new section listing  
128568 reserved names for the `<stdint.h>` header. This change is for alignment with the ISO C standard.
- 128569 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/2 is applied, making it clear that  
128570 implementations are permitted to have symbols with the prefix `_POSIX_` visible in any header.
- 128571 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/3 is applied, updating the table of  
128572 allowed macro prefixes to include the prefix `FP_[A-Z]` for `<math.h>`. This text is added for  
128573 consistency with the `<math.h>` reference page in the Base Definitions volume of POSIX.1-2024  
128574 which permits additional implementation-defined floating-point classifications.
- 128575 Austin Group Interpretation 1003.1-2001 #048 is applied, reserving `SEEK_` in the name space.
- 128576 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0001 [801], XSH/TC2-2008/0002 [780],  
128577 XSH/TC2-2008/0003 [790], XSH/TC2-2008/0004 [780], XSH/TC2-2008/0005 [790],  
128578 XSH/TC2-2008/0006 [782], XSH/TC2-2008/0007 [790], and XSH/TC2-2008/0008 [790] are  
128579 applied.
- 128580 Austin Group Defect 162 is applied, adding the `<endian.h>` header.
- 128581 Austin Group Defect 697 is applied, reserving `DT_` in the name space.
- 128582 Austin Group Defect 845 is applied, reserving `in6addr_` in the name space.
- 128583 Austin Group Defect 993 is applied, reserving `dli_` in the name space.
- 128584 Austin Group Defect 1003 is applied, correcting a mismatch with the ISO C standard regarding  
128585 reservation of each identifier with file scope described in the header section.
- 128586 Austin Group Defect 1122 is applied, adding `<libintl.h>`.
- 128587 Austin Group Defect 1151 is applied, adding `ws_` as a reserved prefix for `<termios.h>`.
- 128588 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.
- 128589 Austin Group Defect 1456 is applied, clarifying the reservation of symbolic constants with the  
128590 prefix `_CS_`, `_PC_`, and `_SC_` for `<unistd.h>`.

### 128591 **B.2.3 Error Numbers**

- 128592 It was the consensus of the standard developers that to allow the conformance document to state  
128593 that an error occurs and under what conditions, but to disallow a statement that it never occurs,  
128594 does not make sense. It could be implied by the current wording that this is allowed, but to  
128595 reduce the possibility of future interpretation requests, it is better to make an explicit statement.
- 128596 The original ISO C standard just required that `errno` be a modifiable lvalue. Since the  
128597 introduction of threads in 2011, the ISO C standard has instead required that `errno` be a macro  
128598 which expands to a modifiable lvalue that has thread local storage duration.
- 128599 Checking the value of `errno` alone is not sufficient to determine the existence or type of an error,  
128600 since it is not required that a successful function call clear `errno`. The variable `errno` should only  
128601 be examined when the return value of a function indicates that the value of `errno` is meaningful.  
128602 In that case, the function is required to set the variable to something other than zero.
- 128603 The variable `errno` is never set to zero by any function call; to do so would contradict the ISO C  
128604 standard.
- 128605 POSIX.1 requires (in the ERRORS sections of function descriptions) certain error values to be set

128606 in certain conditions because many existing applications depend on them. Some error numbers,  
 128607 such as [EFAULT], are entirely implementation-defined and are noted as such in their  
 128608 description in the ERRORS section. This section otherwise allows wide latitude to the  
 128609 implementation in handling error reporting.

128610 Some of the ERRORS sections in POSIX.1-2024 have two subsections. The first:

128611        ``The function shall fail if:``

128612 could be called the ``mandatory`` section.

128613 The second:

128614        ``The function may fail if:``

128615 could be informally known as the ``optional`` section.

128616 Attempting to infer the quality of an implementation based on whether it detects optional error  
 128617 conditions is not useful.

128618 Following each one-word symbolic name for an error, there is a description of the error. The  
 128619 rationale for some of the symbolic names follows:

128620 [ECANCELED] This spelling was chosen as being more common.

128621 [EFAULT] Most historical implementations do not catch an error and set *errno* when an  
 128622 invalid address is given to the functions *wait()*, *time()*, or *times()*. Some  
 128623 implementations cannot reliably detect an invalid address. And most systems  
 128624 that detect invalid addresses will do so only for a system call, not for a library  
 128625 routine.

128626 [EFTYPE] This error code was proposed in earlier proposals as ``Inappropriate operation  
 128627 for file type``, meaning that the operation requested is not appropriate for the  
 128628 file specified in the function call. This code was proposed, although the same  
 128629 idea was covered by [ENOTTY], because the connotations of the name would  
 128630 be misleading. It was pointed out that the *fcntl()* function uses the error code  
 128631 [EINVAL] for this notion, and hence all instances of [EFTYPE] were changed  
 128632 to this code.

128633 [EINTR] POSIX.1 prohibits conforming implementations from restarting interrupted  
 128634 system calls of conforming applications unless the SA\_RESTART flag is in  
 128635 effect for the signal. However, it does not require that [EINTR] be returned  
 128636 when another legitimate value may be substituted; for example, a partial  
 128637 transfer count when *read()* or *write()* are interrupted. This is only given when  
 128638 the signal-catching function returns normally as opposed to returns by  
 128639 mechanisms like *longjmp()* or *siglongjmp()*.

128640 [ELOOP] In specifying conditions under which implementations would generate this  
 128641 error, the following goals were considered:

- 128642        • To ensure that actual loops are detected, including loops that result from  
 128643        symbolic links across distributed file systems.
- 128644        • To ensure that during pathname resolution an application can rely on  
 128645        the ability to follow at least {SYMLOOP\_MAX} symbolic links in the  
 128646        absence of a loop.
- 128647        • To allow implementations to provide the capability of traversing more  
 128648        than {SYMLOOP\_MAX} symbolic links in the absence of a loop.

- 128649                   • To allow implementations to detect loops and generate the error prior to  
128650                   encountering {SYMLOOP\_MAX} symbolic links.
- 128651           [ENAMETOOLONG]
- 128652           When a symbolic link is encountered during pathname resolution, the  
128653           contents of that symbolic link are used to create a new pathname. The  
128654           standard developers intended to allow, but not require, that implementations  
128655           enforce the restriction of {PATH\_MAX} on the result of this pathname  
128656           substitution.
- 128657           Implementations are allowed, but not required, to treat a pathname longer  
128658           than {PATH\_MAX} passed into the system as an error. Implementations are  
128659           required to return a pathname (even if it is longer than {PATH\_MAX}) when  
128660           the user supplies a buffer with an interface that specifies the buffer size, as  
128661           long as the user-supplied buffer is large enough to hold the entire pathname  
128662           (see XSH *getcwd()* for an example of this type of interface). Implementations  
128663           are required to treat a request to pass a pathname longer than {PATH\_MAX}  
128664           from the system to a user-supplied buffer of an unspecified size (usually  
128665           assumed to be of size {PATH\_MAX}) as an error (see XSH *realpath()* for an  
128666           example of this type of interface).
- 128667           [ENOMEM]       The term “main memory” is not used in POSIX.1 because it is  
128668           implementation-defined.
- 128669           [ENOTSUP]       This error code is to be used when an implementation chooses to implement  
128670           the required functionality of POSIX.1-2024 but does not support optional  
128671           facilities defined by POSIX.1-2024. In some earlier versions of this standard,  
128672           the difference between [ENOTSUP] and [ENOSYS] was that [ENOSYS]  
128673           indicated that the function was not supported at all. This is no longer the case  
128674           as [ENOSYS] can also be used to indicate non-support of optional  
128675           functionality for a function that has some required functionality. (See XSH  
128676           *encrypt()*.)
- 128677           [ENOTTY]       The symbolic name for this error is derived from a time when device control  
128678           was done by *ioctl()* and that operation was only permitted on a terminal  
128679           interface. The term “TTY” is derived from “teletypewriter”, the devices to  
128680           which this error originally applied.
- 128681           [E\_OVERFLOW]   Most of the uses of this error code are related to large file support. Typically,  
128682           these cases occur on systems which support multiple programming  
128683           environments with different sizes for **off\_t**, but they may also occur in  
128684           connection with remote file systems.
- 128685           In addition, when different programming environments have different widths  
128686           for types such as **int** and **uid\_t**, several functions may encounter a condition  
128687           where a value in a particular environment is too wide to be represented. In  
128688           that case, this error should be raised. For example, suppose the currently  
128689           running process has 64-bit **int**, and file descriptor 9 223 372 036 854 775 807 is  
128690           open and does not have the close-on-exec flag set. If the process then uses  
128691           *execl()* to *exec* a file compiled in a programming environment with 32-bit **int**,  
128692           the call to *execl()* can fail with *errno* set to [E\_OVERFLOW]. A similar failure  
128693           can occur with *execl()* if any of the user IDs or any of the group IDs to be  
128694           assigned to the new process image are out of range for the executed file’s  
128695           programming environment.
- 128696           Note, however, that this condition cannot occur for functions that are  
128697           explicitly described as always being successful, such as *getpid()*.



128698 [EPIPE] This condition normally generates the signal SIGPIPE; the error is returned if  
 128699 the generation of the signal is suppressed or the signal does not terminate the  
 128700 process.

128701 [EROFS] In historical implementations, attempting to *unlink()* or *rmdir()* a mount point  
 128702 would generate an [EBUSY] error. An implementation could be envisioned  
 128703 where such an operation could be performed without error. In this case, if  
 128704 *either* the directory entry or the actual data structures reside on a read-only file  
 128705 system, [EROFS] is the appropriate error to generate. (For example, changing  
 128706 the link count of a file on a read-only file system could not be done, as is  
 128707 required by *unlink()*, and thus an error should be reported.)

128708 Three error numbers, [EDOM], [EILSEQ], and [ERANGE], were added to this section primarily  
 128709 for consistency with the ISO C standard.

128710 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0009 [496] and XSH/TC2-2008/0010  
 128711 [681] are applied.

128712 Austin Group Defect 1067 is applied, adding [ESOCKTNOSUPPORT].

128713 Austin Group Defect 1380 is applied, changing the descriptions of [EMLINK] and [EXDEV].

128714 Austin Group Defect 1669 is applied, changing the description of [EFBIG].

#### 128715 **Alternative Solutions for Per-Thread *errno***

128716 The historical implementation of *errno* as a single global variable does not work in a multi-  
 128717 threaded environment. In such an environment, a thread may make a POSIX.1 call and get a -1  
 128718 error return, but before that thread can check the value of *errno*, another thread might have  
 128719 made a second POSIX.1 call that also set *errno*. This behavior is unacceptable in robust  
 128720 programs. There were a number of alternatives that were considered for handling the *errno*  
 128721 problem:

- 128722 • Implement *errno* as a per-thread integer variable.
- 128723 • Implement *errno* as a service that can access the per-thread error number.
- 128724 • Change all POSIX.1 calls to accept an extra status argument and avoid setting *errno*.
- 128725 • Change all POSIX.1 calls to raise a language exception.

128726 The first option offers the highest level of compatibility with existing practice but requires  
 128727 special support in the linker, compiler, and/or virtual memory system to support the new  
 128728 concept of thread private variables. When compared with current practice, the third and fourth  
 128729 options are much cleaner, more efficient, and encourage a more robust programming style, but  
 128730 they require new versions of all of the POSIX.1 functions that might detect an error. The second  
 128731 option offers compatibility with existing code that uses the **<errno.h>** header to define the  
 128732 symbol *errno*. In this option, *errno* may be a macro defined:

```
128733 #define errno (*__errno())
128734 extern int      *__errno();
```

128735 This option may be implemented as a per-thread variable whereby an *errno* field is allocated in  
 128736 the user space object representing a thread, and whereby the function *\_\_errno()* makes a system  
 128737 call to determine the location of its user space object and returns the address of the *errno* field of  
 128738 that object. Another implementation, one that avoids calling the kernel, involves allocating  
 128739 stacks in chunks. The stack allocator keeps a side table indexed by chunk number containing a  
 128740 pointer to the thread object that uses that chunk. The *\_\_errno()* function then looks at the stack  
 128741 pointer, determines the chunk number, and uses that as an index into the chunk table to find its  
 128742 thread object and thus its private value of *errno*. On most architectures, this can be done in four

128743 to five instructions. Some compilers may wish to implement `__errno()` inline to improve  
128744 performance.

#### 128745 **Disallowing Return of the [EINTR] Error Code**

128746 Many blocking interfaces defined by POSIX.1-2024 may return [EINTR] if interrupted during  
128747 their execution by a signal handler. Blocking interfaces introduced under the threads  
128748 functionality do not have this property. Instead, they require that the interface appear to be  
128749 atomic with respect to interruption. In particular, applications calling blocking interfaces need  
128750 not handle any possible [EINTR] return as a special case since it will never occur. In the case of  
128751 threads functions in `<threads.h>`, the requirement is stated in terms of the call not being affected  
128752 if the calling thread executes a signal handler during the call, since these functions return errors  
128753 in a different way and cannot distinguish an [EINTR] condition from other error conditions. If it  
128754 is necessary to restart operations or complete incomplete operations following the execution of a  
128755 signal handler, this is handled by the implementation, rather than by the application.

128756 Requiring applications to handle [EINTR] errors on blocking interfaces has been shown to be a  
128757 frequent source of often unreproducible bugs, and it adds no compelling value to the available  
128758 functionality. Thus, blocking interfaces introduced for use by multi-threaded programs do not  
128759 use this paradigm. In particular, in none of the functions `flockfile()`, `pthread_cond_timedwait()`,  
128760 `pthread_cond_wait()`, `pthread_join()`, `pthread_mutex_lock()`, and `sigwait()` did providing [EINTR]  
128761 returns add value, or even particularly make sense. Thus, these functions do not provide for an  
128762 [EINTR] return, even when interrupted by a signal handler. The same arguments can be applied  
128763 to `sem_wait()`, `sem_trywait()`, `sigwaitinfo()`, and `sigtimedwait()`, but implementations are  
128764 permitted to return [EINTR] error codes for these functions for compatibility with earlier  
128765 versions of this standard. Applications cannot rely on calls to these functions returning [EINTR]  
128766 error codes when signals are delivered to the calling thread, but they should allow for the  
128767 possibility.

128768 Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and  
128769 [EOPNOTSUPP] to be the same values.

#### 128770 **B.2.3.1 Additional Error Numbers**

128771 The ISO C standard defines the name space for implementations to add additional error  
128772 numbers.

### 128773 **B.2.4 Signal Concepts**

128774 Historical implementations of signals, using the `signal()` function, have shortcomings that make  
128775 them unreliable for many application uses. Because of this, a new signal mechanism, based very  
128776 closely on the one of 4.2 BSD and 4.3 BSD, was added to POSIX.1.

#### 128777 **Signal Names**

128778 The restriction on the actual type used for `sigset_t` is intended to guarantee that these objects can  
128779 always be assigned, have their address taken, and be passed as parameters by value. It is not  
128780 intended that this type be a structure including pointers to other data structures, as that could  
128781 impact the portability of applications performing such operations. A reasonable implementation  
128782 could be a structure containing an array of some integer type.

128783 The signals described in POSIX.1-2024 must have unique values so that they may be named as  
128784 parameters of `case` statements in the body of a C-language `switch` clause. However,  
128785 implementation-defined signals may have values that overlap with each other or with signals

128786 specified in POSIX.1-2024. An example of this is SIGABRT, which traditionally overlaps some  
128787 other signal, such as SIGIOT.

128788 SIGKILL, SIGTERM, SIGUSR1, and SIGUSR2 are ordinarily generated only through the explicit  
128789 use of the *kill()* function, although some implementations generate SIGKILL under  
128790 extraordinary circumstances. SIGTERM is traditionally the default signal sent by the *kill*  
128791 command.

128792 The signals SIGBUS, SIGEMT, SIGIOT, SIGTRAP, and SIGSYS were omitted from POSIX.1  
128793 because their behavior is implementation-defined and could not be adequately categorized.  
128794 Conforming implementations may deliver these signals, but must document the circumstances  
128795 under which they are delivered and note any restrictions concerning their delivery. The signals  
128796 SIGFPE, SIGILL, and SIGSEGV are similar in that they also generally result only from  
128797 programming errors. They were included in POSIX.1 because they do indicate three relatively  
128798 well-categorized conditions. They are all defined by the ISO C standard and thus would have to  
128799 be defined by any system with an ISO C standard binding, even if not explicitly included in  
128800 POSIX.1.

128801 There is very little that a Conforming POSIX.1 Application can do by catching, ignoring, or  
128802 masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, SIGSEGV, SIGSYS, or  
128803 SIGFPE. They will generally be generated by the system only in cases of programming errors.  
128804 While it may be desirable for some robust code (for example, a library routine) to be able to  
128805 detect and recover from programming errors in other code, these signals are not nearly sufficient  
128806 for that purpose. One portable use that does exist for these signals is that a command interpreter  
128807 can recognize them as the cause of termination of a process (with *wait()*) and print an  
128808 appropriate message. The mnemonic tags for these signals are derived from their PDP-11 origin.

128809 The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job control  
128810 and are unchanged from 4.2 BSD. The signal SIGCHLD is also typically used by job control  
128811 shells to detect children that have terminated or, as in 4.2 BSD, stopped.

128812 Some implementations, including System V, have a signal named SIGCLD, which is similar to  
128813 SIGCHLD in 4.2 BSD. POSIX.1 permits implementations to have a single signal with both  
128814 names. POSIX.1 carefully specifies ways in which conforming applications can avoid the  
128815 semantic differences between the two different implementations. The name SIGCHLD was  
128816 chosen for POSIX.1 because most current application usages of it can remain unchanged in  
128817 conforming applications. SIGCLD in System V has more cases of semantics that POSIX.1 does  
128818 not specify, and thus applications using it are more likely to require changes in addition to the  
128819 name change.

128820 The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of  
128821 exceptional behavior and are described as “reserved as application-defined” so that such use is  
128822 not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except when  
128823 explicitly requested by *kill()*. It is recommended that libraries not use these two signals, as such  
128824 use in libraries could interfere with their use by applications calling the libraries. If such use is  
128825 unavoidable, it should be documented. It is prudent for non-portable libraries to use non-  
128826 standard signals to avoid conflicts with use of standard signals by portable libraries.

128827 There is no portable way for an application to catch or ignore non-standard signals. Some  
128828 implementations define the range of signal numbers, so applications can install signal-catching  
128829 functions for all of them. Unfortunately, implementation-defined signals often cause problems  
128830 when caught or ignored by applications that do not understand the reason for the signal. While  
128831 the desire exists for an application to be more robust by handling all possible signals (even those  
128832 only generated by *kill()*), no existing mechanism was found to be sufficiently portable to include  
128833 in POSIX.1. The value of such a mechanism, if included, would be diminished given that  
128834 SIGKILL would still not be catchable.

128835 A number of new signal numbers are reserved for applications because the two user signals  
128836 defined by POSIX.1 are insufficient for many realtime applications. A range of signal numbers is  
128837 specified, rather than an enumeration of additional reserved signal names, because different  
128838 applications and application profiles will require a different number of application signals. It is  
128839 not desirable to burden all application domains and therefore all implementations with the  
128840 maximum number of signals required by all possible applications. Note that in this context,  
128841 signal numbers are essentially different signal priorities.

128842 The relatively small number of required additional signals, `{_POSIX_RTSIG_MAX}`, was chosen  
128843 so as not to require an unreasonably large signal mask/set. While this number of signals  
128844 defined in POSIX.1 will fit in a single 32-bit word signal mask, it is recognized that most existing  
128845 implementations define many more signals than are specified in POSIX.1 and, in fact, many  
128846 implementations have already exceeded 32 signals (including the “null signal”). Support of  
128847 `{_POSIX_RTSIG_MAX}` additional signals may push some implementation over the single 32-bit  
128848 word line, but is unlikely to push any implementations that are already over that line beyond  
128849 the 64-signal line.

#### 128850 B.2.4.1 Signal Generation and Delivery

128851 The terms defined in this section are not used consistently in documentation of historical  
128852 systems. Each signal can be considered to have a lifetime beginning with generation and ending  
128853 with delivery or acceptance. The POSIX.1 definition of “delivery” does not exclude ignored  
128854 signals; this is considered a more consistent definition. This revised text in several parts of  
128855 POSIX.1-2024 clarifies the distinct semantics of asynchronous signal delivery and synchronous  
128856 signal acceptance. The previous wording attempted to categorize both under the term  
128857 “delivery”, which led to conflicts over whether the effects of asynchronous signal delivery  
128858 applied to synchronous signal acceptance.

128859 Signals generated for a process are delivered to only one thread. Thus, if more than one thread  
128860 is eligible to receive a signal, one has to be chosen. The choice of threads is left entirely up to the  
128861 implementation both to allow the widest possible range of conforming implementations and to  
128862 give implementations the freedom to deliver the signal to the “easiest possible” thread should  
128863 there be differences in ease of delivery between different threads.

128864 Note that should multiple delivery among cooperating threads be required by an application,  
128865 this can be trivially constructed out of the provided single-delivery semantics. The construction  
128866 of a `sigwait_multiple()` function that accomplishes this goal is presented with the rationale for  
128867 `sigwaitinfo()`.

128868 Implementations should deliver unblocked signals as soon after they are generated as possible.  
128869 However, it is difficult for POSIX.1 to make specific requirements about this, beyond those in  
128870 `kill()` and `sigprocmask()`. Even on systems with prompt delivery, scheduling of higher priority  
128871 processes is always likely to cause delays.

128872 In general, the interval between the generation and delivery of unblocked signals cannot be  
128873 detected by an application. Thus, references to pending signals generally apply to blocked,  
128874 pending signals. An implementation registers a signal as pending on the process when no  
128875 thread has the signal unblocked and there are no threads blocked in a `sigwait()` function for that  
128876 signal. Thereafter, the implementation delivers the signal to the first thread that unblocks the  
128877 signal or calls a `sigwait()` function on a signal set containing this signal rather than choosing the  
128878 recipient thread at the time the signal is sent.

128879 In the 4.3 BSD system, signals that are blocked and set to `SIG_IGN` are discarded immediately  
128880 upon generation. For a signal that is ignored as its default action, if the action is `SIG_DFL` and  
128881 the signal is blocked, a generated signal remains pending. In the 4.1 BSD system and in  
128882 System V Release 3 (two other implementations that support a somewhat similar signal

128883 mechanism), all ignored blocked signals remain pending if generated. Because it is not normally  
 128884 useful for an application to simultaneously ignore and block the same signal, it was unnecessary  
 128885 for POSIX.1 to specify behavior that would invalidate any of the historical implementations.

128886 There is one case in some historical implementations where an unblocked, pending signal does  
 128887 not remain pending until it is delivered. In the System V implementation of *signal()*, pending  
 128888 signals are discarded when the action is set to SIG\_DFL or a signal-catching routine (as well as  
 128889 to SIG\_IGN). Except in the case of setting SIGCHLD to SIG\_DFL, implementations that do this  
 128890 do not conform completely to POSIX.1. Some earlier proposals for POSIX.1 explicitly stated this,  
 128891 but these statements were redundant due to the requirement that functions defined by POSIX.1  
 128892 not change attributes of processes defined by POSIX.1 except as explicitly stated.

128893 POSIX.1 specifically states that the order in which multiple, simultaneously pending signals are  
 128894 delivered is unspecified. This order has not been explicitly specified in historical  
 128895 implementations, but has remained quite consistent and been known to those familiar with the  
 128896 implementations. Thus, there have been cases where applications (usually system utilities) have  
 128897 been written with explicit or implicit dependencies on this order. Implementors and others  
 128898 porting existing applications may need to be aware of such dependencies.

128899 When there are multiple pending signals that are not blocked, implementations should arrange  
 128900 for the delivery of all signals at once, if possible. Some implementations stack calls to all pending  
 128901 signal-catching routines, making it appear that each signal-catcher was interrupted by the next  
 128902 signal. In this case, the implementation should ensure that this stacking of signals does not  
 128903 violate the semantics of the signal masks established by *sigaction()*. Other implementations  
 128904 process at most one signal when the operating system is entered, with remaining signals saved  
 128905 for later delivery. Although this practice is widespread, this behavior is neither standardized  
 128906 nor endorsed. In either case, implementations should attempt to deliver signals associated with  
 128907 the current state of the process (for example, SIGFPE) before other signals, if possible.

128908 In 4.2 BSD and 4.3 BSD, it is not permissible to ignore or explicitly block SIGCONT, because if  
 128909 blocking or ignoring this signal prevented it from continuing a stopped process, such a process  
 128910 could never be continued (only killed by SIGKILL). However, 4.2 BSD and 4.3 BSD do block  
 128911 SIGCONT during execution of its signal-catching function when it is caught, creating exactly  
 128912 this problem. A proposal was considered to disallow catching SIGCONT in addition to ignoring  
 128913 and blocking it, but this limitation led to objections. The consensus was to require that  
 128914 SIGCONT always continue a stopped process when generated. This removed the need to  
 128915 disallow ignoring or explicit blocking of the signal; note that SIG\_IGN and SIG\_DFL are  
 128916 equivalent for SIGCONT.

#### 128917 B.2.4.2 Realtime Signal Generation and Delivery

128918 The realtime signals functionality is required in this version of the standard for the following  
 128919 reasons:

- 128920 • The **sigevent** structure is used by other POSIX.1 functions that result in asynchronous  
 128921 event notifications to specify the notification mechanism to use and other information  
 128922 needed by the notification mechanism. POSIX.1-2024 defines only three symbolic values  
 128923 for the notification mechanism:
  - 128924 — SIGEV\_NONE is used to indicate that no notification is required when the event  
 128925 occurs. This is useful for applications that use asynchronous I/O with polling for  
 128926 completion.
  - 128927 — SIGEV\_SIGNAL indicates that a signal is generated when the event occurs.

128928 — SIGEV\_THREAD provides for “callback functions” for asynchronous notifications  
 128929 done by a function call within the context of a new thread. This provides a multi-  
 128930 threaded process with a more natural means of notification than signals.

128931 The primary difficulty with previous notification approaches has been to specify the  
 128932 environment of the notification routine.

128933 — One approach is to limit the notification routine to call only functions permitted in a  
 128934 signal handler. While the list of permissible functions is clearly stated, this is overly  
 128935 restrictive.

128936 — A second approach is to define a new list of functions or classes of functions that are  
 128937 explicitly permitted or not permitted. This would give a programmer more lists to  
 128938 deal with, which would be awkward.

128939 — The third approach is to define completely the environment for execution of the  
 128940 notification function. A clear definition of an execution environment for notification  
 128941 is provided by executing the notification function in the environment of a newly  
 128942 created thread.

128943 Implementations may support additional notification mechanisms by defining new values  
 128944 for *sigev\_notify*.

128945 For a notification type of SIGEV\_SIGNAL, the other members of the **sigevent** structure  
 128946 defined by POSIX.1-2024 specify the realtime signal—that is, the signal number and  
 128947 application-defined value that differentiates between occurrences of signals with the same  
 128948 number—that will be generated when the event occurs. The structure is defined in  
 128949 **<signal.h>**, even though the structure is not directly used by any of the signal functions,  
 128950 because it is part of the signals interface used by the POSIX.1b “client functions”. When the  
 128951 client functions include **<signal.h>** to define the signal names, the **sigevent** structure will  
 128952 also be defined.

128953 An application-defined value passed to the signal handler is used to differentiate between  
 128954 different “events” instead of requiring that the application use different signal numbers for  
 128955 several reasons:

128956 — Realtime applications potentially handle a very large number of different events.  
 128957 Requiring that implementations support a correspondingly large number of distinct  
 128958 signal numbers will adversely impact the performance of signal delivery because the  
 128959 signal masks to be manipulated on entry and exit to the handlers will become large.

128960 — Event notifications are prioritized by signal number (the rationale for this is  
 128961 explained in the following paragraphs) and the use of different signal numbers to  
 128962 differentiate between the different event notifications overloads the signal number  
 128963 more than has already been done. It also requires that the application developer  
 128964 make arbitrary assignments of priority to events that are logically of equal priority.

128965 A union is defined for the application-defined value so that either an integer constant or a  
 128966 pointer can be portably passed to the signal-catching function. On some architectures a  
 128967 pointer cannot be cast to an **int** and *vice versa*.

128968 Use of a structure here with an explicit notification type discriminant rather than explicit  
 128969 parameters to realtime functions, or embedded in other realtime structures, provides for  
 128970 future extensions to POSIX.1-2024. Additional, perhaps more efficient, notification  
 128971 mechanisms can be supported for existing realtime function interfaces, such as timers and  
 128972 asynchronous I/O, by extending the **sigevent** structure appropriately. The existing  
 128973 realtime function interfaces will not have to be modified to use any such new notification  
 128974 mechanism. The revised text concerning the SIGEV\_SIGNAL value makes consistent the

128975 semantics of the members of the **sigevent** structure, particularly in the definitions of  
 128976 *lio\_listio()* and *aio\_fsync()*. For uniformity, other revisions cause this specification to be  
 128977 referred to rather than inaccurately duplicated in the descriptions of functions and  
 128978 structures using the **sigevent** structure. The revised wording does not relax the  
 128979 requirement that the signal number be in the range SIGRTMIN to SIGRTMAX to guarantee  
 128980 queuing and passing of the application value, since that requirement is still implied by the  
 128981 signal names.

128982 • POSIX.1-2024 is intentionally vague on whether “non-realtime” signal-generating  
 128983 mechanisms can result in a **siginfo\_t** being supplied to the handler on delivery. In one  
 128984 existing implementation, a **siginfo\_t** is posted on signal generation, even though the  
 128985 implementation does not support queuing of multiple occurrences of a signal. It is not the  
 128986 intent of POSIX.1-2024 to preclude this, independent of the mandate to define signals that  
 128987 do support queuing. Any interpretation that appears to preclude this is a mistake in the  
 128988 reading or writing of the standard.

128989 • Signals handled by realtime signal handlers might be generated by functions or conditions  
 128990 that do not allow the specification of an application-defined value and do not queue.  
 128991 POSIX.1-2024 specifies the *si\_code* member of the **siginfo\_t** structure used in existing  
 128992 practice and defines additional codes so that applications can detect whether an  
 128993 application-defined value is present or not. The code SI\_USER for *kill()*-generated signals  
 128994 is adopted from existing practice.

128995 • The *sigaction()* *sa\_flags* value SA\_SIGINFO tells the implementation that the signal-  
 128996 catching function expects two additional arguments. When the flag is not set, a single  
 128997 argument, the signal number, is passed as specified by POSIX.1-2024. Although  
 128998 POSIX.1-2024 does not explicitly allow the *info* argument to the handler function to be  
 128999 NULL, this is existing practice. This provides for compatibility with programs whose  
 129000 signal-catching functions are not prepared to accept the additional arguments.  
 129001 POSIX.1-2024 is explicitly unspecified as to whether signals actually queue when  
 129002 SA\_SIGINFO is not set for a signal, as there appear to be no benefits to applications in  
 129003 specifying one behavior or another. One existing implementation queues a **siginfo\_t** on  
 129004 each signal generation, unless the signal is already pending, in which case the  
 129005 implementation discards the new **siginfo\_t**; that is, the queue length is never greater than  
 129006 one. This implementation only examines SA\_SIGINFO on signal delivery, discarding the  
 129007 queued **siginfo\_t** if its delivery was not requested.

129008 The third argument to the signal-catching function, *context*, is left undefined by  
 129009 POSIX.1-2024, but is specified in the interface because it matches existing practice for the  
 129010 SA\_SIGINFO flag. It was considered undesirable to require a separate implementation for  
 129011 SA\_SIGINFO for POSIX conformance on implementations that already support the two  
 129012 additional parameters.

129013 • The requirement to deliver lower numbered signals in the range SIGRTMIN to SIGRTMAX  
 129014 first, when multiple unblocked signals are pending, results from several considerations:

129015 — A method is required to prioritize event notifications. The signal number was chosen  
 129016 instead of, for instance, associating a separate priority with each request, because an  
 129017 implementation has to check pending signals at various points and select one for  
 129018 delivery when more than one is pending. Specifying a selection order is the minimal  
 129019 additional semantic that will achieve prioritized delivery. If a separate priority were  
 129020 to be associated with queued signals, it would be necessary for an implementation to  
 129021 search all non-empty, non-blocked signal queues and select from among them the  
 129022 pending signal with the highest priority. This would significantly increase the cost of  
 129023 and decrease the determinism of signal delivery.

129024 — Given the specified selection of the lowest numeric unblocked pending signal,  
 129025 preemptive priority signal delivery can be achieved using signal numbers and signal  
 129026 masks by ensuring that the *sa\_mask* for each signal number blocks all signals with a  
 129027 higher numeric value.

129028 For realtime applications that want to use only the newly defined realtime signal  
 129029 numbers without interference from the standard signals, this can be achieved by  
 129030 blocking all of the standard signals in the thread signal mask and in the *sa\_mask*  
 129031 installed by the signal action for the realtime signal handlers.

129032 POSIX.1-2024 explicitly leaves unspecified the ordering of signals outside of the range of  
 129033 realtime signals and the ordering of signals within this range with respect to those outside  
 129034 the range. It was believed that this would unduly constrain implementations or standards  
 129035 in the future definition of new signals.

129036 Austin Group Defect 633 is applied, reducing to two the allowed behaviors for the signal mask  
 129037 of the thread that is created to handle a SIGEV\_THREAD notification.

129038 Austin Group Defect 1116 is applied, removing a reference to the Realtime Signals Extension  
 129039 option that existed in earlier versions of this standard.

#### 129040 B.2.4.3 Signal Actions

129041 Early proposals mentioned SIGCONT as a second exception to the rule that signals are not  
 129042 delivered to stopped processes until continued. Because POSIX.1-2024 now specifies that  
 129043 SIGCONT causes the stopped process to continue when it is generated, delivery of SIGCONT is  
 129044 not prevented because a process is stopped, even without an explicit exception to this rule.

129045 Ignoring a signal by setting the action to SIG\_IGN (or SIG\_DFL for signals whose default action  
 129046 is to ignore) is not the same as installing a signal-catching function that simply returns. Invoking  
 129047 such a function will interrupt certain system functions that block processes (for example, *wait()*,  
 129048 *sigsuspend()*, *pause()*, *read()*, *write()*) while ignoring a signal has no such effect on the process.

129049 Historical implementations discard pending signals when the action is set to SIG\_IGN.  
 129050 However, they do not always do the same when the action is set to SIG\_DFL and the default  
 129051 action is to ignore the signal. POSIX.1-2024 requires this for the sake of consistency and also for  
 129052 completeness, since the only signal this applies to is SIGCHLD, and POSIX.1-2024 disallows  
 129053 setting its action to SIG\_IGN.

129054 Some implementations (System V, for example) assign different semantics for SIGCHLD  
 129055 depending on whether the action is set to SIG\_IGN or SIG\_DFL. Since POSIX.1 requires that the  
 129056 default action for SIGCHLD be to ignore the signal, applications should always set the action to  
 129057 SIG\_DFL in order to avoid SIGCHLD.

129058 Whether or not an implementation allows SIG\_IGN as a SIGCHLD disposition to be inherited  
 129059 across a call to one of the *exec* family of functions or *posix\_spawn()* is explicitly left as  
 129060 unspecified. This change was made as a result of IEEE PASC Interpretation 1003.1 #132, and  
 129061 permits the implementation to decide between the following alternatives:

- 129062 • Unconditionally leave SIGCHLD set to SIG\_IGN, in which case the implementation would  
 129063 not allow applications that assume inheritance of SIG\_DFL to conform to POSIX.1-2024  
 129064 without change. The implementation would, however, retain an ability to control  
 129065 applications that create child processes but never call on the *wait* family of functions,  
 129066 potentially filling up the process table.
- 129067 • Unconditionally reset SIGCHLD to SIG\_DFL, in which case the implementation would  
 129068 allow applications that assume inheritance of SIG\_DFL to conform. The implementation  
 129069 would, however, lose an ability to control applications that spawn child processes but



129070 never reap them.

- 129071 • Provide some mechanism, not specified in POSIX.1-2024, to control inherited SIGCHLD
- 129072 dispositions.

129073 Some implementations (System V, for example) will deliver a SIGCLD signal immediately when  
 129074 a process establishes a signal-catching function for SIGCLD when that process has a child that  
 129075 has already terminated. Other implementations, such as 4.3 BSD, do not generate a new  
 129076 SIGCHLD signal in this way. In general, a process should not attempt to alter the signal action  
 129077 for the SIGCHLD signal while it has any outstanding children. However, it is not always  
 129078 possible for a process to avoid this; for example, shells sometimes start up processes in pipelines  
 129079 with other processes from the pipeline as children. Processes that cannot ensure that they have  
 129080 no children when altering the signal action for SIGCHLD thus need to be prepared for, but not  
 129081 depend on, generation of an immediate SIGCHLD signal.

129082 The default action of the stop signals (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is to stop a  
 129083 process that is executing. If a stop signal is delivered to a process that is already stopped, it has  
 129084 no effect. In fact, if a stop signal is generated for a stopped process whose signal mask blocks the  
 129085 signal, the signal will never be delivered to the process since the process must receive a  
 129086 SIGCONT, which discards all pending stop signals, in order to continue executing.

129087 The SIGCONT signal continues a stopped process even if SIGCONT is blocked (or ignored).  
 129088 However, if a signal-catching routine has been established for SIGCONT, it will not be entered  
 129089 until SIGCONT is unblocked.

129090 If a process in an orphaned process group stops, it is no longer under the control of a job control  
 129091 shell and hence would not normally ever be continued. Because of this, orphaned processes that  
 129092 receive terminal-related stop signals (SIGTSTP, SIGTTIN, SIGTTOU, but not SIGSTOP) must not  
 129093 be allowed to stop. The goal is to prevent stopped processes from languishing forever. (As  
 129094 SIGSTOP is sent only via *kill()*, it is assumed that the process or user sending a SIGSTOP can  
 129095 send a SIGCONT when desired.) Instead, the system must discard the stop signal. As an  
 129096 extension, it may also deliver another signal in its place. 4.3 BSD sends a SIGKILL, which is  
 129097 overly effective because SIGKILL is not catchable. Another possible choice is SIGHUP. 4.3 BSD  
 129098 also does this for orphaned processes (processes whose parent has terminated) rather than for  
 129099 members of orphaned process groups; this is less desirable because job control shells manage  
 129100 process groups. POSIX.1 also prevents SIGTTIN and SIGTTOU signals from being generated for  
 129101 processes in orphaned process groups as a direct result of activity on a terminal, preventing  
 129102 infinite loops when *read()* and *write()* calls generate signals that are discarded; see [Section](#)  
 129103 [A.11.1.4](#) (on page 3720). A similar restriction on the generation of SIGTSTP was considered, but  
 129104 that would be unnecessary and more difficult to implement due to its asynchronous nature.

129105 Although POSIX.1 requires that signal-catching functions be called with only one argument,  
 129106 there is nothing to prevent conforming implementations from extending POSIX.1 to pass  
 129107 additional arguments, as long as Strictly Conforming POSIX.1 Applications continue to compile  
 129108 and execute correctly. Most historical implementations do, in fact, pass additional, signal-  
 129109 specific arguments to certain signal-catching routines.

129110 There was a proposal to change the declared type of the signal handler to:

```
129111 void func (int sig, ...);
```

129112 The usage of ellipses ("*...*") is ISO C standard syntax to indicate a variable number of  
 129113 arguments. Its use was intended to allow the implementation to pass additional information to  
 129114 the signal handler in a standard manner.

129115 Unfortunately, this construct would require all signal handlers to be defined with this syntax  
 129116 because the ISO C standard allows implementations to use a different parameter passing  
 129117 mechanism for variable parameter lists than for non-variable parameter lists. Thus, all existing

129118 signal handlers in all existing applications would have to be changed to use the variable syntax  
129119 in order to be standard and portable. This is in conflict with the goal of Minimal Changes to  
129120 Existing Application Code.

129121 When terminating a process from a signal-catching function, processes should be aware of any  
129122 interpretation that their parent may make of the status returned by *wait()*, *waitid()*, or *waitpid()*.  
129123 In particular, a signal-catching function should not call *exit(0)* or *\_exit(0)* unless it wants to  
129124 indicate successful termination. A non-zero argument to *exit()* or *\_exit()* can be used to indicate  
129125 unsuccessful termination. Alternatively, the process can use *kill()* to send itself a fatal signal  
129126 (first ensuring that the signal is set to the default action and not blocked). See also the  
129127 RATIONALE section of the *\_exit()* function.

129128 The behavior of *unsafe* functions, as defined by this section, is undefined when they are called  
129129 from (or after a *longjmp()* or *siglongjmp()* out of) signal-catching functions in certain  
129130 circumstances. The behavior of *async-signal-safe* functions, as defined by this section, is as  
129131 specified by POSIX.1, regardless of invocation from a signal-catching function. This is the only  
129132 intended meaning of the statement that *async-signal-safe* functions may be used in signal-  
129133 catching functions without restriction. Applications must still consider all effects of such  
129134 functions on such things as data structures, files, and process state. In particular, application  
129135 developers need to consider the restrictions on interactions when interrupting *sleep()* (see  
129136 *sleep()*) and interactions among multiple handles for a file description. The fact that any specific  
129137 function is listed as *async-signal-safe* does not necessarily mean that invocation of that function  
129138 from a signal-catching function is recommended.

129139 In order to prevent errors arising from interrupting non-*async-signal-safe* function calls,  
129140 applications should protect calls to these functions either by blocking the appropriate signals or  
129141 through the use of some programmatic semaphore. POSIX.1 does not address the more general  
129142 problem of synchronizing access to shared data structures. Note in particular that even the  
129143 “safe” functions may modify the global variable *errno*; the signal-catching function may want to  
129144 save and restore its value. The same principles apply to the *async-signal-safety* of application  
129145 routines and asynchronous data access.

129146 Note that although *longjmp()* and *siglongjmp()* are in the list of *async-signal-safe* functions, there  
129147 are restrictions on subsequent behavior after the function is called from a signal-catching  
129148 function. This is because the code executing after *longjmp()* or *siglongjmp()* can call any *unsafe*  
129149 functions with the same danger as calling those *unsafe* functions directly from the signal  
129150 handler. Applications that use *longjmp()* or *siglongjmp()* out of signal handlers require rigorous  
129151 protection in order to be portable. Many of the other functions that are excluded from the list are  
129152 traditionally implemented using either the C language *malloc()* or *free()* functions or the ISO C  
129153 standard I/O library, both of which traditionally use data structures in a non-*async-signal-safe*  
129154 manner. Because any combination of different functions using a common data structure can  
129155 cause *async-signal-safety* problems, POSIX.1 does not define the behavior when any *unsafe*  
129156 function is called in (or after a *longjmp()* or *siglongjmp()* out of) a signal handler that interrupts  
129157 any *unsafe* function or the non-*async-signal-safe* processing equivalent to *exit()* that is  
129158 performed after return from the initial call to *main()*.

129159 The only realtime extension to signal actions is the addition of the additional parameters to the  
129160 signal-catching function. This extension has been explained and motivated in the previous  
129161 section. In making this extension, though, developers of POSIX.1b ran into issues relating to  
129162 function prototypes. In response to input from the POSIX.1 standard developers, members were  
129163 added to the **sigaction** structure to specify function prototypes for the newer signal-catching  
129164 function specified by POSIX.1b. These members follow changes that are being made to POSIX.1.  
129165 Note that POSIX.1-2024 explicitly states that these fields may overlap so that a union can be  
129166 defined. This enabled existing implementations of POSIX.1 to maintain binary-compatibility  
129167 when these extensions were added.

129168 The **siginfo\_t** structure was adopted for passing the application-defined value to match existing  
 129169 practice, but the existing practice has no provision for an application-defined value, so this was  
 129170 added. Note that POSIX normally reserves the “\_t” type designation for opaque types. The  
 129171 **siginfo\_t** structure breaks with this convention to follow existing practice and thus promote  
 129172 portability.

129173 POSIX.1-2024 specifies several values for the *si\_code* member of the **siginfo\_t** structure. Some  
 129174 were introduced in POSIX.1b; others were XSI functionality in the Single UNIX Specification,  
 129175 Version 2 and Version 3, that has now become Base functionality. Historically, an *si\_code* value of  
 129176 less than or equal to zero indicated that the signal was generated by a process via the *kill()*  
 129177 function, and values of *si\_code* that provided additional information for implementation-  
 129178 generated signals, such as SIGFPE or SIGSEGV, were all positive. This functionality is partially  
 129179 specified for XSI systems in that if *si\_code* is less than or equal to zero, the signal was generated  
 129180 by a process. However, since POSIX.1b did not specify that SI\_USER (or SI\_QUEUE) had a value  
 129181 less than or equal to zero, it is not true that when the signal is generated by a process, the value  
 129182 of *si\_code* will always be less than or equal to zero. XSI applications should check whether *si\_code*  
 129183 is SI\_USER or SI\_QUEUE in addition to checking whether it is less than or equal to zero.  
 129184 Applications on systems that do not support the XSI option should just check for SI\_USER and  
 129185 SI\_QUEUE.

129186 If an implementation chooses to define additional values for *si\_code*, these values have to be  
 129187 different from the values of the non-signal-specific symbols specified by POSIX.1-2024. This will  
 129188 allow conforming applications to differentiate between signals generated by standard events  
 129189 and those generated by other implementation events in a manner compatible with existing  
 129190 practice.

129191 The unique values of *si\_code* for the POSIX.1b asynchronous events have implications for  
 129192 implementations of, for example, asynchronous I/O or message passing in user space library  
 129193 code. Such an implementation will be required to provide a hidden interface to the signal  
 129194 generation mechanism that allows the library to specify the standard values of *si\_code*.

129195 POSIX.1-2024 also specifies additional members of **siginfo\_t**, beyond those that were in  
 129196 POSIX.1b. Like the *si\_code* values mentioned above, these were XSI functionality in the Single  
 129197 UNIX Specification, Version 2 and Version 3, that has now become Base functionality. They  
 129198 provide additional information when *si\_code* has one of the values that moved from XSI to Base.

129199 Although it is not explicitly visible to applications, there are additional semantics for signal  
 129200 actions implied by queued signals and their interaction with other POSIX.1b realtime functions.  
 129201 Specifically:

- 129202 • It is not necessary to queue signals whose action is SIG\_IGN.
- 129203 • For implementations that support POSIX.1b timers, some interaction with the timer  
 129204 functions at signal delivery is implied to manage the timer overrun count.

129205 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/5 is applied, reordering the RTS shaded  
 129206 text under the third and fourth paragraphs of the SIG\_DFL description. This corrects an earlier  
 129207 editorial error in this section.

129208 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/6 is applied, adding the *abort()* function  
 129209 to the list of async-signal-safe functions.

129210 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/4 is applied, adding the *socketmark()*  
 129211 function to the list of async-signal-safe functions.

129212 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0011 [690], XSH/TC2-2008/0012 [516],  
 129213 XSH/TC2-2008/0013 [692], XSH/TC2-2008/0014 [615], XSH/TC2-2008/0015 [516], and  
 129214 XSH/TC2-2008/0016 [807] are applied.

- 129215 Austin Group Defect 62 is applied, adding the `_Fork()` function to, and removing the `fork()`  
129216 function from, the list of async-signal-safe functions.
- 129217 Austin Group Defect 162 is applied, adding functions from the `<endian.h>` header to the list of  
129218 async-signal-safe functions.
- 129219 Austin Group Defect 411 is applied, adding `accept4()`, `dup3()`, and `pipe2()` to the list of async-  
129220 signal-safe functions.
- 129221 Austin Group Defect 614 is applied, adding `posix_close()` to the list of async-signal-safe  
129222 functions.
- 129223 Austin Group Defect 699 is applied, adding `setegid()`, `seteuid()`, `setregid()`, and `setreuid()` to the  
129224 list of async-signal-safe functions.
- 129225 Austin Group Defect 711 is applied, adding `va_arg()`, `va_copy()`, `va_end()`, and `va_start()` to the  
129226 list of async-signal-safe functions and updating related text to apply to function-like macros.
- 129227 Austin Group Defect 728 is applied, reducing the set of circumstances in which undefined  
129228 behavior results when a signal handler refers to an object with static or thread storage duration.
- 129229 Austin Group Defect 841 is applied, adding `pthread_setcancelstate()` to the list of async-signal-safe  
129230 functions and making it implementation-defined which additional interfaces are also async-  
129231 signal-safe.
- 129232 Austin Group Defect 986 is applied, adding `strlcat()`, `strlcpy()`, `wcslcat()`, and `wcslcpy()` to the list  
129233 of async-signal-safe functions.
- 129234 Austin Group Defect 1138 is applied, adding the `sig2str()` function to the list of async-signal-safe  
129235 functions.
- 129236 Austin Group Defect 1141 is applied, changing “core file” to “core image”.
- 129237 Austin Group Defects 1142, 1455, and 1625 are applied, adding the `pread()`, `pwrite()`, `readv()`,  
129238 `waitid()`, and `writev()` functions to the list of async-signal-safe functions.
- 129239 Austin Group Defect 1151 is applied, adding the `tcgetwinsize()` and `tcsetwinsize()` functions to the  
129240 list of async-signal-safe functions.
- 129241 Austin Group Defect 1215 is applied, removing XSI shading from text relating to abnormal  
129242 process termination with additional actions.
- 129243 Austin Group Defect 1263 is applied, adding the `ppoll()` function to the list of async-signal-safe  
129244 functions.
- 129245 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.
- 129246 Austin Group Defect 1667 is applied, adding `getresgid()`, `getresuid()`, `setresgid()`, and `setresuid()` to  
129247 the list of async-signal-safe functions.
- 129248 Austin Group Defect 1744 is applied, adding `killpg()` to the list of async-signal-safe functions.

129249 **B.2.4.4** *Signal Effects on Other Functions*

- 129250 The most common behavior of an interrupted function after a signal-catching function returns is  
129251 for the interrupted function to give an [EINTR] error unless the SA\_RESTART flag is in effect for  
129252 the signal. However, there are a number of specific exceptions, including `sleep()` and certain  
129253 situations with `read()` and `write()`.
- 129254 The historical implementations of many functions defined by POSIX.1-2024 are not interruptible,  
129255 but delay delivery of signals generated during their execution until after they complete. This is  
129256 never a problem for functions that are guaranteed to complete in a short (imperceptible to a

129257 human) period of time. It is normally those functions that can suspend a process indefinitely or  
 129258 for long periods of time (for example, *wait()*, *pause()*, *sigsuspend()*, *sleep()*, or *read()/write()* on a  
 129259 slow device like a terminal) that are interruptible. This permits applications to respond to  
 129260 interactive signals or to set timeouts on calls to most such functions with *alarm()*. Therefore,  
 129261 implementations should generally make such functions (including ones defined as extensions)  
 129262 interruptible.

129263 Functions not mentioned explicitly as interruptible may be so on some implementations,  
 129264 possibly as an extension where the function gives an [EINTR] error. There are several functions  
 129265 (for example, *getpid()*, *getuid()*) that are specified as never returning an error, which can thus  
 129266 never be extended in this way.

129267 If a signal-catching function returns while the SA\_RESTART flag is in effect, an interrupted  
 129268 function is restarted at the point it was interrupted. Conforming applications cannot make  
 129269 assumptions about the internal behavior of interrupted functions, even if the functions are  
 129270 async-signal-safe. For example, suppose the *read()* function is interrupted with SA\_RESTART in  
 129271 effect, the signal-catching function closes the file descriptor being read from and returns, and the  
 129272 *read()* function is then restarted; in this case the application cannot assume that the *read()*  
 129273 function will give an [EBADF] error, since *read()* might have checked the file descriptor for  
 129274 validity before being interrupted.

129275 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0017 [807] is applied.

## 129276 B.2.5 Standard I/O Streams

129277 Although the ISO C standard guarantees that, at program start-up, *stdin* is open for reading and  
 129278 *stdout* and *stderr* are open for writing, this guarantee is contingent (as are all guarantees made by  
 129279 the ISO C and POSIX standards) on the program being executed in a conforming environment.  
 129280 Programs executed with file descriptor 0 not open for reading or with file descriptor 1 or 2 not  
 129281 open for writing are executed in a non-conforming environment. Application writers are warned  
 129282 (in *exec*, *posix\_spawn()*, and Section C.2.7, on page 3890) not to execute a standard utility or a  
 129283 conforming application with file descriptor 0 not open for reading or with file descriptor 1 or 2  
 129284 not open for writing.

129285 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0018 [608] is applied.

129286 Austin Group Defect 689 is applied, clarifying the handling of deadlock situations when locking  
 129287 a stream.

129288 Austin Group Defect 1144 is applied, clarifying the effect of *setvbuf()* on memory streams.

129289 Austin Group Defect 1153 is applied, clarifying that the behavior is undefined if a memory  
 129290 buffer associated with a standard I/O stream overlaps with the destination buffer of a call that  
 129291 reads from the stream or with the source buffer of a call that writes to the stream.

129292 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

129293 Austin Group Defect 1347 is applied, clarifying the requirements for how *stderr*, *stdin*, and *stdout*  
 129294 are opened at program start-up.

### 129295 B.2.5.1 Interaction of File Descriptors and Standard I/O Streams

129296 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0019 [480] is applied.

129297 Austin Group Defect 1183 is applied, changing “non-full” to “non-null”.

129298 Austin Group Defect 1318 is applied, changing the list of functions that close file descriptors.

129299 B.2.5.2 *Stream Orientation and Encoding Rules*

129300 Austin Group Defect 1040 is applied, clarifying that conversion to or from (possibly multi-byte)  
 129301 characters is not performed by wide character I/O functions when the stream was opened using  
 129302 *open\_wmemstream()*.

129303 **B.2.6 File Descriptor Allocation**

129304 Functions such as *pipe()* and *socketpair()* which allocate two file descriptors are permitted to  
 129305 perform the two allocations independently. This means that other threads or signal handlers  
 129306 may perform operations on file descriptors in between the two allocations and this can result in  
 129307 the two file descriptors not having adjacent values or in the second allocation producing a lower  
 129308 value than the first.

129309 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0032 [835] is applied.

129310 **B.2.7 XSI Interprocess Communication**

129311 There are two forms of IPC supported as options in POSIX.1-2024. The traditional System V IPC  
 129312 routines derived from the SVID—that is, the *msg\*()*, *sem\*()*, and *shm\*()* interfaces—are  
 129313 mandatory on XSI-conformant systems. Thus, all XSI-conformant systems provide the same  
 129314 mechanisms for manipulating messages, shared memory, and semaphores.

129315 In addition, the POSIX Realtime Extension provides an alternate set of routines for those systems  
 129316 supporting the appropriate options.

129317 The application developer is presented with a choice: the System V interfaces or the POSIX  
 129318 interfaces (loosely derived from the Berkeley interfaces). The XSI profile prefers the System V  
 129319 interfaces, but the POSIX interfaces may be more suitable for realtime or other performance-  
 129320 sensitive applications.

129321 B.2.7.1 *IPC General Description*

129322 General information that is shared by all three mechanisms is described in this section. The  
 129323 common permissions mechanism is briefly introduced, describing the mode bits, and how they  
 129324 are used to determine whether or not a process has access to read or write/alter the appropriate  
 129325 instance of one of the IPC mechanisms. All other relevant information is contained in the  
 129326 reference pages themselves.

129327 The semaphore type of IPC allows processes to communicate through the exchange of  
 129328 semaphore values. A semaphore is a positive integer. Since many applications require the use of  
 129329 more than one semaphore, XSI-conformant systems have the ability to create sets or arrays of  
 129330 semaphores.

129331 Calls to support semaphores include:

129332 *semctl()*, *semget()*, *semop()*

129333 Semaphore sets are created by using the *semget()* function.

129334 The message type of IPC allows processes to communicate through the exchange of data stored  
 129335 in buffers. This data is transmitted between processes in discrete portions known as messages.

129336 Calls to support message queues include:

129337 *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

129338 The shared memory type of IPC allows two or more processes to share memory and  
 129339 consequently the data contained therein. This is done by allowing processes to set up access to a  
 129340 common memory address space. This sharing of memory provides a fast means of exchange of  
 129341 data between processes.

129342 Calls to support shared memory include:

129343 `shmctl()`, `shmdt()`, `shmget()`

129344 The `ftok()` interface is also provided.

129345 Austin Group Defect 377 is applied, changing the table giving the values for the *mode* member of  
 129346 the `ipc_perm` structure.

## 129347 B.2.8 Realtime

### 129348 Advisory Information

129349 POSIX.1b contains an Informative Annex with proposed interfaces for “realtime files”. These  
 129350 interfaces could determine groups of the exact parameters required to do “direct I/O” or  
 129351 “extents”. These interfaces were objected to by a significant portion of the balloting group as too  
 129352 complex. A conforming application had little chance of correctly navigating the large parameter  
 129353 space to match its desires to the system. In addition, they only applied to a new type of file  
 129354 (realtime files) and they told the implementation exactly what to do as opposed to advising the  
 129355 implementation on application behavior and letting it optimize for the system the (portable)  
 129356 application was running on. For example, it was not clear how a system that had a disk array  
 129357 should set its parameters.

129358 There seemed to be several overall goals:

- 129359 • Optimizing sequential access
- 129360 • Optimizing caching behavior
- 129361 • Optimizing I/O data transfer
- 129362 • Preallocation

129363 The advisory interfaces, `posix_fadvise()` and `posix_madvise()`, satisfy the first two goals. The  
 129364 `POSIX_FADV_SEQUENTIAL` and `POSIX_MADV_SEQUENTIAL` advice tells the  
 129365 implementation to expect serial access. Typically the system will prefetch the next several serial  
 129366 accesses in order to overlap I/O. It may also free previously accessed serial data if memory is  
 129367 tight. If the application is not doing serial access it can use `POSIX_FADV_WILLNEED` and  
 129368 `POSIX_MADV_WILLNEED` to accomplish I/O overlap, as required. When the application  
 129369 advises `POSIX_FADV_RANDOM` or `POSIX_MADV_RANDOM` behavior, the implementation  
 129370 usually tries to fetch a minimum amount of data with each request and it does not expect much  
 129371 locality. `POSIX_FADV_DONTNEED` and `POSIX_MADV_DONTNEED` allow the system to free  
 129372 up caching resources as the data will not be required in the near future.

129373 `POSIX_FADV_NOREUSE` tells the system that caching the specified data is not optimal. For file  
 129374 I/O, the transfer should go directly to the user buffer instead of being cached internally by the  
 129375 implementation. To portably perform direct disk I/O on all systems, the application must  
 129376 perform its I/O transfers according to the following rules:

- 129377 1. The user buffer should be aligned according to the `{POSIX_REC_XFER_ALIGN}`  
 129378 `pathconf()` variable.

- 129379 2. The number of bytes transferred in an I/O operation should be a multiple of the  
129380 {POSIX\_ALLOC\_SIZE\_MIN} *pathconf()* variable.
- 129381 3. The offset into the file at the start of an I/O operation should be a multiple of the  
129382 {POSIX\_ALLOC\_SIZE\_MIN} *pathconf()* variable.
- 129383 4. The application should ensure that all threads which open a given file specify  
129384 POSIX\_FADV\_NOREUSE to be sure that there is no unexpected interaction between  
129385 threads using buffered I/O and threads using direct I/O to the same file.

129386 In some cases, a user buffer must be properly aligned in order to be transferred directly to/from  
129387 the device. The {POSIX\_REC\_XFER\_ALIGN} *pathconf()* variable tells the application the proper  
129388 alignment.

129389 The preallocation goal is met by the space control function, *posix\_fallocate()*. The application can  
129390 use *posix\_fallocate()* to guarantee no [ENOSPC] errors and to improve performance by prepaying  
129391 any overhead required for block allocation.

129392 Implementations may use information conveyed by a previous *posix\_fadvise()* call to influence  
129393 the manner in which allocation is performed. For example, if an application did the following  
129394 calls:

```
129395 fd = open("file");
129396 posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
129397 posix_fallocate(fd, len, size);
```

129398 an implementation might allocate the file contiguously on disk.

129399 Finally, the *pathconf()* variables {POSIX\_REC\_MIN\_XFER\_SIZE},  
129400 {POSIX\_REC\_MAX\_XFER\_SIZE}, and {POSIX\_REC\_INCR\_XFER\_SIZE} tell the application a  
129401 range of transfer sizes that are recommended for best I/O performance.

129402 Where bounded response time is required, the vendor can supply the appropriate settings of the  
129403 advisories to achieve a guaranteed performance level.

129404 The interfaces meet the goals while allowing applications using regular files to take advantage  
129405 of performance optimizations. The interfaces tell the implementation expected application  
129406 behavior which the implementation can use to optimize performance on a particular system  
129407 with a particular dynamic load.

129408 The *posix\_memalign()* function was added to allow for the allocation of specifically aligned  
129409 buffers; for example, for {POSIX\_REC\_XFER\_ALIGN}.

129410 The working group also considered the alternative of adding a function which would return an  
129411 aligned pointer to memory within a user-supplied buffer. This was not considered to be the best  
129412 method, because it potentially wastes large amounts of memory when buffers need to be aligned  
129413 on large alignment boundaries.

## 129414 Message Passing

129415 This section provides the rationale for the definition of the message passing interface in  
129416 POSIX.1-2024. This is presented in terms of the objectives, models, and requirements imposed  
129417 upon this interface.

- 129418 • Objectives

129419 Many applications, including both realtime and database applications, require a means of  
129420 passing arbitrary amounts of data between cooperating processes comprising the overall  
129421 application on one or more processors. Many conventional interfaces for interprocess  
129422 communication are insufficient for realtime applications in that efficient and deterministic



129423 data passing methods cannot be implemented. This has prompted the definition of  
129424 message passing interfaces providing these facilities:

- 129425 — Open a message queue.
- 129426 — Send a message to a message queue.
- 129427 — Receive a message from a queue, either synchronously or asynchronously.
- 129428 — Alter message queue attributes for flow and resource control.

129429 It is assumed that an application may consist of multiple cooperating processes and that  
129430 these processes may wish to communicate and coordinate their activities. The message  
129431 passing facility described in POSIX.1-2024 allows processes to communicate through  
129432 system-wide queues. These message queues are accessed through names that may be  
129433 pathnames. A message queue can be opened for use by multiple sending and/or multiple  
129434 receiving processes.

#### 129435 • Background on Embedded Applications

129436 Interprocess communication utilizing message passing is a key facility for the construction  
129437 of deterministic, high-performance realtime applications. The facility is present in all  
129438 realtime systems and is the framework upon which the application is constructed. The  
129439 performance of the facility is usually a direct indication of the performance of the resulting  
129440 application.

129441 Realtime applications, especially for embedded systems, are typically designed around the  
129442 performance constraints imposed by the message passing mechanisms. Applications for  
129443 embedded systems are typically very tightly constrained. Application developers expect to  
129444 design and control the entire system. In order to minimize system costs, the writer will  
129445 attempt to use all resources to their utmost and minimize the requirement to add  
129446 additional memory or processors.

129447 The embedded applications usually share address spaces and only a simple message  
129448 passing mechanism is required. The application can readily access common data incurring  
129449 only mutual-exclusion overheads. The models desired are the simplest possible with the  
129450 application building higher-level facilities only when needed.

#### 129451 • Requirements

129452 The following requirements determined the features of the message passing facilities  
129453 defined in POSIX.1-2024:

##### 129454 — Naming of Message Queues

129455 The mechanism for gaining access to a message queue is a pathname evaluated in a  
129456 context that is allowed to be a file system name space, or it can be independent of  
129457 any file system. This is a specific attempt to allow implementations based on either  
129458 method in order to address both embedded systems and to also allow  
129459 implementation in larger systems.

129460 The interface of `mq_open()` is defined to allow but not require the access control and  
129461 name conflicts resulting from utilizing a file system for name resolution. All required  
129462 behavior is specified for the access control case. Yet a conforming implementation,  
129463 such as an embedded system kernel, may define that there are no distinctions  
129464 between users and may define that all processes have all access privileges.

##### 129465 — Embedded System Naming

129466 Embedded systems need to be able to utilize independent name spaces for accessing  
129467 the various system objects. They typically do not have a file system, precluding its

- 129468 utilization as a common name resolution mechanism. The modularity of an  
 129469 embedded system limits the connections between separate mechanisms that can be  
 129470 allowed.
- 129471 Embedded systems typically do not have any access protection. Since the system  
 129472 does not support the mixing of applications from different areas, and usually does  
 129473 not even have the concept of an authorization entity, access control is not useful.
- 129474 — Large System Naming
- 129475 On systems with more functionality, the name resolution must support the ability to  
 129476 use the file system as the name resolution mechanism/object storage medium and to  
 129477 have control over access to the objects. Utilizing the pathname space can result in  
 129478 further errors when the names conflict with other objects.
- 129479 — Fixed Size of Messages
- 129480 The interfaces impose a fixed upper bound on the size of messages that can be sent to  
 129481 a specific message queue. The size is set on an individual queue basis and cannot be  
 129482 changed dynamically.
- 129483 The purpose of the fixed size is to increase the ability of the system to optimize the  
 129484 implementation of *mq\_send()* and *mq\_receive()*. With fixed sizes of messages and  
 129485 fixed numbers of messages, specific message blocks can be pre-allocated. This  
 129486 eliminates a significant amount of checking for errors and boundary conditions.  
 129487 Additionally, an implementation can optimize data copying to maximize  
 129488 performance. Finally, with a restricted range of message sizes, an implementation is  
 129489 better able to provide deterministic operations.
- 129490 — Prioritization of Messages
- 129491 Message prioritization allows the application to determine the order in which  
 129492 messages are received. Prioritization of messages is a key facility that is provided by  
 129493 most realtime kernels and is heavily utilized by the applications. The major purpose  
 129494 of having priorities in message queues is to avoid priority inversions in the message  
 129495 system, where a high-priority message is delayed behind one or more lower-priority  
 129496 messages. This allows the applications to be designed so that they do not need to be  
 129497 interrupted in order to change the flow of control when exceptional conditions occur.  
 129498 The prioritization does add additional overhead to the message operations, in those  
 129499 cases it is actually used, but a clever implementation can optimize for the FIFO case  
 129500 to make that more efficient.
- 129501 — Asynchronous Notification
- 129502 The interface supports the ability to have a task asynchronously notified of the  
 129503 availability of a message on the queue. The purpose of this facility is to allow the task  
 129504 to perform other functions and yet still be notified that a message has become  
 129505 available on the queue.
- 129506 To understand the requirement for this function, it is useful to understand two  
 129507 models of application design: a single task performing multiple functions and  
 129508 multiple tasks performing a single function. Each of these models has advantages.
- 129509 Asynchronous notification is required to build the model of a single task performing  
 129510 multiple operations. This model typically results from either the expectation that  
 129511 interruption is less expensive than utilizing a separate task or from the growth of the  
 129512 application to include additional functions.

**Semaphores**

129513  
129514 Semaphores are a high-performance process synchronization mechanism. Semaphores are  
129515 named by null-terminated strings of characters.

129516 A semaphore is created using the *sem\_init()* function or the *sem\_open()* function with the  
129517 *O\_CREAT* flag set in *oflag*.

129518 To use a semaphore, a process has to first initialize the semaphore or inherit an open descriptor  
129519 for the semaphore via *fork()*.

129520 A semaphore preserves its state when the last reference is closed. For example, if a semaphore  
129521 has a value of 13 when the last reference is closed, it will have a value of 13 when it is next  
129522 opened.

129523 When a semaphore is created, an initial state for the semaphore has to be provided. This value is  
129524 a non-negative integer. Negative values are not possible since they indicate the presence of  
129525 blocked processes. The persistence of any of these objects across a system crash or a system  
129526 reboot is undefined. Conforming applications must not depend on any sort of persistence across  
129527 a system reboot or a system crash.

129528 • Models and Requirements

129529 A realtime system requires synchronization and communication between the processes  
129530 comprising the overall application. An efficient and reliable synchronization mechanism  
129531 has to be provided in a realtime system that will allow more than one schedulable process  
129532 mutually-exclusive access to the same resource. This synchronization mechanism has to  
129533 allow for the optimal implementation of synchronization or systems implementors will  
129534 define other, more cost-effective methods.

129535 At issue are the methods whereby multiple processes (tasks) can be designed and  
129536 implemented to work together in order to perform a single function. This requires  
129537 interprocess communication and synchronization. A semaphore mechanism is the lowest  
129538 level of synchronization that can be provided by an operating system.

129539 A semaphore is defined as an object that has an integral value and a set of blocked  
129540 processes associated with it. If the value is positive or zero, then the set of blocked  
129541 processes is empty; otherwise, the size of the set is equal to the absolute value of the  
129542 semaphore value. The value of the semaphore can be incremented or decremented by any  
129543 process with access to the semaphore and must be done as an indivisible operation. When  
129544 a semaphore value is less than or equal to zero, any process that attempts to lock it again  
129545 will block or be informed that it is not possible to perform the operation.

129546 A semaphore may be used to guard access to any resource accessible by more than one  
129547 schedulable task in the system. It is a global entity and not associated with any particular  
129548 process. As such, a method of obtaining access to the semaphore has to be provided by the  
129549 operating system. A process that wants access to a critical resource (section) has to wait on  
129550 the semaphore that guards that resource. When the semaphore is locked on behalf of a  
129551 process, it knows that it can utilize the resource without interference by any other  
129552 cooperating process in the system. When the process finishes its operation on the resource,  
129553 leaving it in a well-defined state, it posts the semaphore, indicating that some other  
129554 process may now obtain the resource associated with that semaphore.

129555 In this section, mutexes and condition variables are specified as the synchronization  
129556 mechanisms between threads.

129557 These primitives are typically used for synchronizing threads that share memory in a  
129558 single process. However, this section provides an option allowing the use of these  
129559 synchronization interfaces and objects between processes that share memory, regardless of

- 129560 the method for sharing memory.
- 129561 Much experience with semaphores shows that there are two distinct uses of  
129562 synchronization: locking, which is typically of short duration; and waiting, which is  
129563 typically of long or unbounded duration. These distinct usages map directly onto mutexes  
129564 and condition variables, respectively.
- 129565 Semaphores are provided in POSIX.1-2024 primarily to provide a means of  
129566 synchronization for processes; these processes may or may not share memory. Mutexes  
129567 and condition variables are specified as synchronization mechanisms between threads;  
129568 these threads always share (some) memory. Both are synchronization paradigms that have  
129569 been in widespread use for a number of years. Each set of primitives is particularly well  
129570 matched to certain problems.
- 129571 With respect to binary semaphores, experience has shown that condition variables and  
129572 mutexes are easier to use for many synchronization problems than binary semaphores. The  
129573 primary reason for this is the explicit appearance of a Boolean predicate that specifies  
129574 when the condition wait is satisfied. This Boolean predicate terminates a loop, including  
129575 the call to `pthread_cond_wait()`. As a result, extra wakeups are benign since the predicate  
129576 governs whether the thread will actually proceed past the condition wait. With stateful  
129577 primitives, such as binary semaphores, the wakeup in itself typically means that the wait is  
129578 satisfied. The burden of ensuring correctness for such waits is thus placed on *all* signalers  
129579 of the semaphore rather than on an *explicitly coded* Boolean predicate located at the  
129580 condition wait. Experience has shown that the latter creates a major improvement in safety  
129581 and ease-of-use.
- 129582 Counting semaphores are well matched to dealing with producer/consumer problems,  
129583 including those that might exist between threads of different processes, or between a signal  
129584 handler and a thread. In the former case, there may be little or no memory shared by the  
129585 processes; in the latter case, one is not communicating between co-equal threads, but  
129586 between a thread and an interrupt-like entity. It is for these reasons that POSIX.1-2024  
129587 allows semaphores to be used by threads.
- 129588 Mutexes and condition variables have been effectively used with and without priority  
129589 inheritance, priority ceiling, and other attributes to synchronize threads that share  
129590 memory. The efficiency of their implementation is comparable to or better than that of  
129591 other synchronization primitives that are sometimes harder to use (for example, binary  
129592 semaphores). Furthermore, there is at least one known implementation of Ada tasking that  
129593 uses these primitives. Mutexes and condition variables together constitute an appropriate,  
129594 sufficient, and complete set of inter-thread synchronization primitives.
- 129595 Efficient multi-threaded applications require high-performance synchronization  
129596 primitives. Considerations of efficiency and generality require a small set of primitives  
129597 upon which more sophisticated synchronization functions can be built.
- 129598 • Standardization Issues
- 129599 It is possible to implement very high-performance semaphores using test-and-set  
129600 instructions on shared memory locations. The library routines that implement such a high-  
129601 performance interface have to properly ensure that a `sem_wait()` or `sem_trywait()` operation  
129602 that cannot be performed will issue a blocking semaphore system call or properly report  
129603 the condition to the application. The same interface to the application program would be  
129604 provided by a high-performance implementation.

129605 B.2.8.1 *Realtime Signals*

## 129606 **Realtime Signals Extension**

129607 This portion of the rationale presents models, requirements, and standardization issues relevant  
129608 to the Realtime Signals Extension. This extension provides the capability required to support  
129609 reliable, deterministic, asynchronous notification of events. While a new mechanism,  
129610 unencumbered by the historical usage and semantics of POSIX.1 signals, might allow for a more  
129611 efficient implementation, the application requirements for event notification can be met with a  
129612 small number of extensions to signals. Therefore, a minimal set of extensions to signals to  
129613 support the application requirements is specified.

129614 The realtime signal extensions specified in this section are used by other realtime functions  
129615 requiring asynchronous notification:

- 129616 • **Models**

129617 The model supported is one of multiple cooperating processes, each of which handles  
129618 multiple asynchronous external events. Events represent occurrences that are generated as  
129619 the result of some activity in the system. Examples of occurrences that can constitute an  
129620 event include:

- 129621 — Completion of an asynchronous I/O request
- 129622 — Expiration of a POSIX.1b timer
- 129623 — Arrival of an interprocess message
- 129624 — Generation of a user-defined event

129625 Processing of these events may occur synchronously via polling for event notifications or  
129626 asynchronously via a software interrupt mechanism. Existing practice for this model is  
129627 well established for traditional proprietary realtime operating systems, realtime  
129628 executives, and realtime extended POSIX-like systems.

129629 A contrasting model is that of “cooperating sequential processes” where each process  
129630 handles a single priority of events via polling. Each process blocks while waiting for  
129631 events, and each process depends on the preemptive, priority-based process scheduling  
129632 mechanism to arbitrate between events of different priority that need to be processed  
129633 concurrently. Existing practice for this model is also well established for small realtime  
129634 executives that typically execute in an unprotected physical address space, but it is just  
129635 emerging in the context of a fuller function operating system with multiple virtual address  
129636 spaces.

129637 It could be argued that the cooperating sequential process model, and the facilities  
129638 supported by the POSIX Threads Extension obviate a software interrupt model. But, even  
129639 with the cooperating sequential process model, the need has been recognized for a  
129640 software interrupt model to handle exceptional conditions and process aborting, so the  
129641 mechanism must be supported in any case. Furthermore, it is not the purview of  
129642 POSIX.1-2024 to attempt to convince realtime practitioners that their current application  
129643 models based on software interrupts are “broken” and should be replaced by the  
129644 cooperating sequential process model. Rather, it is the charter of POSIX.1-2024 to provide  
129645 standard extensions to mechanisms that support existing realtime practice.

- 129646 • **Requirements**

129647 This section discusses the following realtime application requirements for asynchronous  
129648 event notification:

- 129649 — Reliable delivery of asynchronous event notification
- 129650 The events notification mechanism guarantees delivery of an event notification.  
129651 Asynchronous operations (such as asynchronous I/O and timers) that complete  
129652 significantly after they are invoked have to guarantee that delivery of the event  
129653 notification can occur at the time of completion.
- 129654 — Prioritized handling of asynchronous event notifications
- 129655 The events notification mechanism supports the assigning of a user function as an  
129656 event notification handler. Furthermore, the mechanism supports the preemption of  
129657 an event handler function by a higher priority event notification and supports the  
129658 selection of the highest priority pending event notification when multiple  
129659 notifications (of different priority) are pending simultaneously.
- 129660 The model here is based on hardware interrupts. Asynchronous event handling  
129661 allows the application to ensure that time-critical events are immediately processed  
129662 when delivered, without the indeterminism of being at a random location within a  
129663 polling loop. Use of handler priority allows the specification of how handlers are  
129664 interrupted by other higher priority handlers.
- 129665 — Differentiation between multiple occurrences of event notifications of the same type
- 129666 The events notification mechanism passes an application-defined value to the event  
129667 handler function. This value can be used for a variety of purposes, such as enabling  
129668 the application to identify which of several possible events of the same type (for  
129669 example, timer expirations) has occurred.
- 129670 — Polled reception of asynchronous event notifications
- 129671 The events notification mechanism supports blocking and non-blocking polls for  
129672 asynchronous event notification.
- 129673 The polled mode of operation is often preferred over the interrupt mode by those  
129674 practitioners accustomed to this model. Providing support for this model facilitates  
129675 the porting of applications based on this model to POSIX.1b conforming systems.
- 129676 — Deterministic response to asynchronous event notifications
- 129677 The events notification mechanism does not preclude implementations that provide  
129678 deterministic event dispatch latency and minimizes the number of system calls  
129679 needed to use the event facilities during realtime processing.
- 129680 • Rationale for Extension
- 129681 POSIX.1 signals have many of the characteristics necessary to support the asynchronous  
129682 handling of event notifications, and the Realtime Signals Extension addresses the  
129683 following deficiencies in the POSIX.1 signal mechanism:
- 129684 — Signals do not support reliable delivery of event notification. Subsequent  
129685 occurrences of a pending signal are not guaranteed to be delivered.
- 129686 — Signals do not support prioritized delivery of event notifications. The order of signal  
129687 delivery when multiple unblocked signals are pending is undefined.
- 129688 — Signals do not support the differentiation between multiple signals of the same type.

## 129689 B.2.8.2 Asynchronous I/O

129690 Many applications need to interact with the I/O subsystem in an asynchronous manner. The  
129691 asynchronous I/O mechanism provides the ability to overlap application processing and I/O  
129692 operations initiated by the application. The asynchronous I/O mechanism allows a single  
129693 process to perform I/O simultaneously to a single file multiple times or to multiple files  
129694 multiple times.

129695 **Overview**

129696 Asynchronous I/O operations proceed in logical parallel with the processing done by the  
129697 application after the asynchronous I/O has been initiated. Other than this difference,  
129698 asynchronous I/O behaves similarly to normal I/O using *read()*, *write()*, *lseek()*, and *fsync()*.  
129699 The effect of issuing an asynchronous I/O request is as if a separate thread of execution were to  
129700 perform atomically the implied *lseek()* operation, if any, and then the requested I/O operation  
129701 (either *read()*, *write()*, or *fsync()*). There is no seek implied with a call to *aiio\_fsync()*. Concurrent  
129702 asynchronous operations and synchronous operations applied to the same file update the file as  
129703 if the I/O operations had proceeded serially.

129704 When asynchronous I/O completes, a signal can be delivered to the application to indicate the  
129705 completion of the I/O. This signal can be used to indicate that buffers and control blocks used  
129706 for asynchronous I/O can be reused. Signal delivery is not required for an asynchronous  
129707 operation and may be turned off on a per-operation basis by the application. Signals may also be  
129708 synchronously polled using *aiio\_suspend()*, *sigtimedwait()*, or *sigwaitinfo()*.

129709 Normal I/O has a return value and an error status associated with it. Asynchronous I/O  
129710 returns a value and an error status when the operation is first submitted, but that only relates to  
129711 whether the operation was successfully queued up for servicing. The I/O operation itself also  
129712 has a return status and an error value. To allow the application to retrieve the return status and  
129713 the error value, functions are provided that, given the address of an asynchronous I/O control  
129714 block, yield the return and error status associated with the operation. Until an asynchronous I/O  
129715 operation is done, its error status is [EINPROGRESS]. Thus, an application can poll for  
129716 completion of an asynchronous I/O operation by waiting for the error status to become equal to  
129717 a value other than [EINPROGRESS]. The return status of an asynchronous I/O operation is  
129718 undefined so long as the error status is equal to [EINPROGRESS].

129719 Storage for asynchronous operation return and error status may be limited. Submission of  
129720 asynchronous I/O operations may fail if this storage is exceeded. When an application retrieves  
129721 the return status of a given asynchronous operation, therefore, any system-maintained storage  
129722 used for this status and the error status may be reclaimed for use by other asynchronous  
129723 operations.

129724 Asynchronous I/O can be performed on file descriptors that have been enabled for POSIX.1b  
129725 synchronized I/O. In this case, the I/O operation still occurs asynchronously, as defined herein;  
129726 however, the asynchronous operation I/O in this case is not completed until the I/O has reached  
129727 either the state of synchronized I/O data integrity completion or synchronized I/O file integrity  
129728 completion, depending on the sort of synchronized I/O that is enabled on the file descriptor.

129729 **Models**

129730 Three models illustrate the use of asynchronous I/O: a journalization model, a data acquisition  
129731 model, and a model of the use of asynchronous I/O in supercomputing applications.

## 129732 • Journalization Model

129733 Many realtime applications perform low-priority journalizing functions. Journalizing  
129734 requires that logging records be queued for output without blocking the initiating process.

## 129735 • Data Acquisition Model

129736 A data acquisition process may also serve as a model. The process has two or more  
129737 channels delivering intermittent data that must be read within a certain time. The process  
129738 issues one asynchronous read on each channel. When one of the channels needs data  
129739 collection, the process reads the data and posts it through an asynchronous write to  
129740 secondary memory for future processing.

## 129741 • Supercomputing Model

129742 The supercomputing community has used asynchronous I/O much like that specified in  
129743 POSIX.1 for many years. This community requires the ability to perform multiple I/O  
129744 operations to multiple devices with a minimal number of entries to “the system”; each  
129745 entry to “the system” provokes a major delay in operations when compared to the normal  
129746 progress made by the application. This existing practice motivated the use of combined  
129747 *lseek()* and *read()* or *write()* calls, as well as the *lio\_listio()* call. Another common practice is  
129748 to disable signal notification for I/O completion, and simply poll for I/O completion at  
129749 some interval by which the I/O should be completed. Likewise, interfaces like *aio\_cancel()*  
129750 have been in successful commercial use for many years. Note also that an underlying  
129751 implementation of asynchronous I/O will require the ability, at least internally, to cancel  
129752 outstanding asynchronous I/O, at least when the process exits. (Consider an asynchronous  
129753 read from a terminal, when the process intends to exit immediately.)

129754 **Requirements**

129755 Asynchronous input and output for realtime implementations have these requirements:

- 129756 • The ability to queue multiple asynchronous read and write operations to a single open  
129757 instance. Both sequential and random access should be supported.
- 129758 • The ability to queue asynchronous read and write operations to multiple open instances.
- 129759 • The ability to obtain completion status information by polling and/or asynchronous event  
129760 notification.
- 129761 • Asynchronous event notification on asynchronous I/O completion is optional.
- 129762 • It has to be possible for the application to associate the event with the *aioctx* for the  
129763 operation that generated the event.
- 129764 • The ability to cancel queued requests.
- 129765 • The ability to wait upon asynchronous I/O completion in conjunction with other types of  
129766 events.
- 129767 • The ability to accept an *aio\_read()* and an *aio\_cancel()* for a device that accepts a *read()*, and  
129768 the ability to accept an *aio\_write()* and an *aio\_cancel()* for a device that accepts a *write()*.  
129769 This does not imply that the operation is asynchronous.



129770 **Standardization Issues**

129771 The following issues are addressed by the standardization of asynchronous I/O:

## 129772 • Rationale for New Interface

129773 Non-blocking I/O does not satisfy the needs of either realtime or high-performance  
 129774 computing models; these models require that a process overlap program execution and  
 129775 I/O processing. Realtime applications will often make use of direct I/O to or from the  
 129776 address space of the process, or require synchronized (unbuffered) I/O; they also require  
 129777 the ability to overlap this I/O with other computation. In addition, asynchronous I/O  
 129778 allows an application to keep a device busy at all times, possibly achieving greater  
 129779 throughput. Supercomputing and database architectures will often have specialized  
 129780 hardware that can provide true asynchrony underlying the logical asynchrony provided  
 129781 by this interface. In addition, asynchronous I/O should be supported by all types of files  
 129782 and devices in the same manner.

## 129783 • Effect of Buffering

129784 If asynchronous I/O is performed on a file that is buffered prior to being actually written  
 129785 to the device, it is possible that asynchronous I/O will offer no performance advantage  
 129786 over normal I/O; the cycles *stolen* to perform the asynchronous I/O will be taken away  
 129787 from the running process and the I/O will occur at interrupt time. This potential lack of  
 129788 gain in performance in no way obviates the need for asynchronous I/O by realtime  
 129789 applications, which very often will use specialized hardware support, multiple processors,  
 129790 and/or unbuffered, synchronized I/O.

129791 *B.2.8.3 Memory Management*129792 All memory management and shared memory definitions are located in the `<sys/mman.h>`  
129793 header. This is for alignment with historical practice.129794 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/7 is applied, correcting the shading and  
129795 margin markers in the introduction to Section 2.8.3.1.129796 **Memory Locking Functions**129797 This portion of the rationale presents models, requirements, and standardization issues relevant  
129798 to process memory locking.

## 129799 • Models

129800 Realtime systems that conform to POSIX.1-2024 are expected (and desired) to be supported  
 129801 on systems with demand-paged virtual memory management, non-paged swapping  
 129802 memory management, and physical memory systems with no memory management  
 129803 hardware. The general case, however, is the demand-paged, virtual memory system with  
 129804 each POSIX process running in a virtual address space. Note that this includes  
 129805 architectures where each process resides in its own virtual address space and architectures  
 129806 where the address space of each process is only a portion of a larger global virtual address  
 129807 space.

129808 The concept of memory locking is introduced to eliminate the indeterminacy introduced  
 129809 by paging and swapping, and to support an upper bound on the time required to access  
 129810 the memory mapped into the address space of a process. Ideally, this upper bound will be  
 129811 the same as the time required for the processor to access “main memory”, including any  
 129812 address translation and cache miss overheads. But some implementations—primarily on  
 129813 mainframes—will not actually force locked pages to be loaded and held resident in main  
 129814 memory. Rather, they will handle locked pages so that accesses to these pages will meet the

129815 performance metrics for locked process memory in the implementation. Also, although it  
 129816 is not, for example, the intention that this interface, as specified, be used to lock process  
 129817 memory into “cache”, it is conceivable that an implementation could support a large static  
 129818 RAM memory and define this as “main memory” and use a large[r] dynamic RAM as  
 129819 “backing store”. These interfaces could then be interpreted as supporting the locking of  
 129820 process memory into the static RAM. Support for multiple levels of backing store would  
 129821 require extensions to these interfaces.

129822 Implementations may also use memory locking to guarantee a fixed translation between  
 129823 virtual and physical addresses where such is beneficial to improving determinacy for  
 129824 direct-to/from-process input/output. POSIX.1-2024 does not guarantee to the application  
 129825 that the virtual-to-physical address translations, if such exist, are fixed, because such  
 129826 behavior would not be implementable on all architectures on which implementations of  
 129827 POSIX.1-2024 are expected. But POSIX.1-2024 does mandate that an implementation  
 129828 define, for the benefit of potential users, whether or not locking guarantees fixed  
 129829 translations.

129830 Memory locking is defined with respect to the address space of a process. Only the pages  
 129831 mapped into the address space of a process may be locked by the process, and when the  
 129832 pages are no longer mapped into the address space—for whatever reason—the locks  
 129833 established with respect to that address space are removed. Shared memory areas warrant  
 129834 special mention, as they may be mapped into more than one address space or mapped  
 129835 more than once into the address space of a process; locks may be established on pages  
 129836 within these areas with respect to several of these mappings. In such a case, the lock state  
 129837 of the underlying physical pages is the logical OR of the lock state with respect to each of  
 129838 the mappings. Only when all such locks have been removed are the shared pages  
 129839 considered unlocked.

129840 In recognition of the page granularity of Memory Management Units (MMU), and in order  
 129841 to support locking of ranges of address space, memory locking is defined in terms of  
 129842 “page” granularity. That is, for the interfaces that support an address and size specification  
 129843 for the region to be locked, the address must be on a page boundary, and all pages mapped  
 129844 by the specified range are locked, if valid. This means that the length is implicitly rounded  
 129845 up to a multiple of the page size. The page size is implementation-defined and is available  
 129846 to applications as a compile-time symbolic constant or at runtime via *sysconf()*.

129847 A “real memory” POSIX.1b implementation that has no MMU could elect not to support  
 129848 these interfaces, returning [ENOSYS]. But an application could easily interpret this as  
 129849 meaning that the implementation would unconditionally page or swap the application  
 129850 when such is not the case. It is the intention of POSIX.1-2024 that such a system could  
 129851 define these interfaces as “NO-OPs”, returning success without actually performing any  
 129852 function except for mandated argument checking.

129853 • Requirements

129854 For realtime applications, memory locking is generally considered to be required as part of  
 129855 application initialization. This locking is performed after an application has been loaded  
 129856 (that is, *exec'd*) and the program remains locked for its entire lifetime. But to support  
 129857 applications that undergo major mode changes where, in one mode, locking is required,  
 129858 but in another it is not, the specified interfaces allow repeated locking and unlocking of  
 129859 memory within the lifetime of a process.

129860 When a realtime application locks its address space, it should not be necessary for the  
 129861 application to then “touch” all of the pages in the address space to guarantee that they are  
 129862 resident or else suffer potential paging delays the first time the page is referenced. Thus,  
 129863 POSIX.1-2024 requires that the pages locked by the specified interfaces be resident when

129864 the locking functions return successfully.

129865 Many architectures support system-managed stacks that grow automatically when the  
129866 current extent of the stack is exceeded. A realtime application has a requirement to be able  
129867 to “preallocate” sufficient stack space and lock it down so that it will not suffer page faults  
129868 to grow the stack during critical realtime operation. There was no consensus on a portable  
129869 way to specify how much stack space is needed, so POSIX.1-2024 supports no specific  
129870 interface for preallocating stack space. But an application can portably lock down a specific  
129871 amount of stack space by specifying `MCL_FUTURE` in a call to `mlockall()` and then calling  
129872 a dummy function that declares an automatic array of the desired size.

129873 Memory locking for realtime applications is also generally considered to be an “all or  
129874 nothing” proposition. That is, the entire process, or none, is locked down. But, for  
129875 applications that have well-defined sections that need to be locked and others that do not,  
129876 POSIX.1-2024 supports an optional set of interfaces to lock or unlock a range of process  
129877 addresses. Reasons for locking down a specific range include:

- 129878 — An asynchronous event handler function that must respond to external events in a  
129879 deterministic manner such that page faults cannot be tolerated
- 129880 — An input/output “buffer” area that is the target for direct-to-process I/O, and the  
129881 overhead of implicit locking and unlocking for each I/O call cannot be tolerated

129882 Finally, locking is generally viewed as an “application-wide” function. That is, the  
129883 application is globally aware of which regions are locked and which are not over time. This  
129884 is in contrast to a function that is used temporarily within a “third party” library routine  
129885 whose function is unknown to the application, and therefore must have no “side-effects”.  
129886 The specified interfaces, therefore, do not support “lock stacking” or “lock nesting” within  
129887 a process. But, for pages that are shared between processes or mapped more than once  
129888 into a process address space, “lock stacking” is essentially mandated by the requirement  
129889 that unlocking of pages that are mapped by more than one process or more than once by  
129890 the same process does not affect locks established on the other mappings.

129891 There was some support for “lock stacking” so that locking could be transparently used in  
129892 functions or opaque modules. But the consensus was not to burden all implementations  
129893 with lock stacking (and reference counting), and an implementation option was proposed.  
129894 There were strong objections to the option because applications would have to support  
129895 both options in order to remain portable. The consensus was to eliminate lock stacking  
129896 altogether, primarily through overwhelming support for the System V “`m[un]lock[all]`”  
129897 interface on which POSIX.1-2024 is now based.

129898 Locks are not inherited across `fork()`s because some implementations implement `fork()` by  
129899 creating new address spaces for the child. In such an implementation, requiring locks to be  
129900 inherited would lead to new situations in which a fork would fail due to the inability of  
129901 the system to lock sufficient memory to lock both the parent and the child. The consensus  
129902 was that there was no benefit to such inheritance. Note that this does not mean that locks  
129903 are removed when, for instance, a thread is created in the same address space.

129904 Similarly, locks are not inherited across `exec` because some implementations implement `exec`  
129905 by unmapping all of the pages in the address space (which, by definition, removes the  
129906 locks on these pages), and maps in pages of the `exec`’d image. In such an implementation,  
129907 requiring locks to be inherited would lead to new situations in which `exec` would fail.  
129908 Reporting this failure would be very cumbersome to detect in time to report to the calling  
129909 process, and no appropriate mechanism exists for informing the `exec`’d process of its status.

129910 It was determined that, if the newly loaded application required locking, it was the  
129911 responsibility of that application to establish the locks. This is also in keeping with the

129912 general view that it is the responsibility of the application to be aware of all locks that are  
129913 established.

129914 There was one request to allow (not mandate) locks to be inherited across *fork()*, and a  
129915 request for a flag, *MCL\_INHERIT*, that would specify inheritance of memory locks across  
129916 *execs*. Given the difficulties raised by this and the general lack of support for the feature in  
129917 POSIX.1-2024, it was not added. POSIX.1-2024 does not preclude an implementation from  
129918 providing this feature for administrative purposes, such as a “run” command that will  
129919 lock down and execute a specified application. Additionally, the rationale for the objection  
129920 equated *fork()* with creating a thread in the address space. POSIX.1-2024 does not mandate  
129921 releasing locks when creating additional threads in an existing process.

129922 • Standardization Issues

129923 One goal of POSIX.1-2024 is to define a set of primitives that provide the necessary  
129924 functionality for realtime applications, with consideration for the needs of other  
129925 application domains where such were identified, which is based to the extent possible on  
129926 existing industry practice.

129927 The Memory Locking option is required by many realtime applications to tune  
129928 performance. Such a facility is accomplished by placing constraints on the virtual memory  
129929 system to limit paging of time of the process or of critical sections of the process. This  
129930 facility should not be used by most non-realtime applications.

129931 Optional features provided in POSIX.1-2024 allow applications to lock selected address  
129932 ranges with the caveat that the process is responsible for being aware of the page  
129933 granularity of locking and the unnested nature of the locks.

129934 **Mapped Files Functions**

129935 The memory mapped files functionality provides a mechanism that allows a process to access  
129936 files by directly incorporating file data into its address space. Once a file is “mapped” into a  
129937 process address space, the data can be manipulated by instructions as memory. The use of  
129938 mapped files can significantly reduce I/O data movement since file data does not have to be  
129939 copied into process data buffers as in *read()* and *write()*. If more than one process maps a file, its  
129940 contents are shared among them. This provides a low overhead mechanism by which processes  
129941 can synchronize and communicate.

129942 • Historical Perspective

129943 Realtime applications have historically been implemented using a collection of cooperating  
129944 processes or tasks. In early systems, these processes ran on bare hardware (that is, without  
129945 an operating system) with no memory relocation or protection. The application paradigms  
129946 that arose from this environment involve the sharing of data between the processes.

129947 When realtime systems were implemented on top of vendor-supplied operating systems,  
129948 the paradigm or performance benefits of direct access to data by multiple processes was  
129949 still deemed necessary. As a result, operating systems that claim to support realtime  
129950 applications must support the shared memory paradigm.

129951 Additionally, a number of realtime systems provide the ability to map specific sections of  
129952 the physical address space into the address space of a process. This ability is required if an  
129953 application is to obtain direct access to memory locations that have specific properties (for  
129954 example, refresh buffers or display devices, dual ported memory locations, DMA target  
129955 locations). The use of this ability is common enough to warrant some degree of  
129956 standardization of its interface. This ability overlaps the general paradigm of shared  
129957 memory in that, in both instances, common global objects are made addressable by  
129958 individual processes or tasks.

129959 Finally, a number of systems also provide the ability to map process addresses to files. This  
 129960 provides both a general means of sharing persistent objects, and using files in a manner  
 129961 that optimizes memory and swapping space usage.

129962 Simple shared memory is clearly a special case of the more general file mapping capability.  
 129963 In addition, there is relatively widespread agreement and implementation of the file  
 129964 mapping interface. In these systems, many different types of objects can be mapped (for  
 129965 example, files, memory, devices, and so on) using the same mapping interfaces. This  
 129966 approach both minimizes interface proliferation and maximizes the generality of programs  
 129967 using the mapping interfaces.

129968 • Memory Mapped Files Usage

129969 A memory object can be concurrently mapped into the address space of one or more  
 129970 processes. The *mmap()* and *munmap()* functions allow a process to manipulate their  
 129971 address space by mapping portions of memory objects into it and removing them from it.  
 129972 When multiple processes map the same memory object, they can share access to the  
 129973 underlying data. Implementations may restrict the size and alignment of mappings to be  
 129974 on *page*-size boundaries. The page size, in bytes, is the value of the system-configurable  
 129975 variable {PAGESIZE}, typically accessed by calling *sysconf()* with a *name* argument of  
 129976 *\_SC\_PAGESIZE*. If an implementation has no restrictions on size or alignment, it may  
 129977 specify a 1-byte page size.

129978 To map memory, a process first opens a memory object. The *ftruncate()* function can be  
 129979 used to contract or extend the size of the memory object even when the object is currently  
 129980 mapped. If the memory object is extended, the contents of the extended areas are zeros.

129981 After opening a memory object, the application maps the object into its address space  
 129982 using the *mmap()* function call. Once a mapping has been established, it remains mapped  
 129983 until unmapped with *munmap()*, even if the memory object is closed. The *mprotect()*  
 129984 function can be used to change the memory protections initially established by *mmap()*.

129985 A *close()* of the file descriptor, while invalidating the file descriptor itself, does not unmap  
 129986 any mappings established for the memory object. The address space, including all mapped  
 129987 regions, is inherited on *fork()*. The entire address space is unmapped on process  
 129988 termination or by successful calls to any of the *exec* family of functions.

129989 The *msync()* function is used to force mapped file data to permanent storage.

129990 • Effects on Other Functions

129991 With memory mapped files, the operation of the *open()*, *creat()*, and *unlink()* functions are  
 129992 a natural result of using the file system name space to map the global names for memory  
 129993 objects.

129994 The *ftruncate()* function can be used to set the length of a sharable memory object.

129995 The meaning of *stat()* fields other than the size and protection information is undefined on  
 129996 implementations where memory objects are not implemented using regular files. When  
 129997 regular files are used, the times reflect when the implementation updated the file image of  
 129998 the data, not when a process updated the data in memory.

129999 The operations of *fdopen()*, *write()*, *read()*, and *lseek()* were made unspecified for objects  
 130000 opened with *shm\_open()*, so that implementations that did not implement memory objects  
 130001 as regular files would not have to support the operation of these functions on shared  
 130002 memory objects.

130003 The behavior of memory objects with respect to *close()*, *dup()*, *dup2()*, *dup3()*, *open()*,  
 130004 *close()*, *fork()*, *\_exit()*, and the *exec* family of functions is the same as the behavior of the

130005 existing practice of the `mmap()` function.

130006 A memory object can still be referenced after a close. That is, any mappings made to the  
130007 file are still in effect, and reads and writes that are made to those mappings are still valid  
130008 and are shared with other processes that have the same mapping. Likewise, the memory  
130009 object can still be used if any references remain after its name(s) have been deleted. Any  
130010 references that remain after a close must not appear to the application as file descriptors.

130011 This is existing practice for `mmap()` and `close()`. In addition, there are already mappings  
130012 present (text, data, stack) that do not have open file descriptors. The text mapping in  
130013 particular is considered a reference to the file containing the text. The desire was to treat all  
130014 mappings by the process uniformly. Also, many modern implementations use `mmap()` to  
130015 implement shared libraries, and it would not be desirable to keep file descriptors for each  
130016 of the many libraries an application can use. It was felt there were many other existing  
130017 programs that used this behavior to free a file descriptor, and thus POSIX.1-2024 could not  
130018 forbid it and still claim to be using existing practice.

130019 For implementations that implement memory objects using memory only, memory objects  
130020 will retain the memory allocated to the file after the last close and will use that same  
130021 memory on the next open. Note that closing the memory object is not the same as deleting  
130022 the name, since the memory object is still defined in the memory object name space.

130023 The locks of `fcntl()` do not block any read or write operation, including read or write access  
130024 to shared memory or mapped files. In addition, implementations that only support shared  
130025 memory objects should not be required to implement record locks. The reference to `fcntl()`  
130026 is added to make this point explicitly. The other `fcntl()` commands are useful with shared  
130027 memory objects.

130028 The size of pages that mapping hardware may be able to support may be a configurable  
130029 value, or it may change based on hardware implementations. The addition of the  
130030 `_SC_PAGESIZE` parameter to the `sysconf()` function is provided for determining the  
130031 mapping page size at runtime.

### 130032 **Shared Memory Functions**

130033 Implementations may support the Shared Memory Objects option independently of memory  
130034 mapped files. Shared memory objects are named regions of storage that may be independent of  
130035 the file system and can be mapped into the address space of one or more processes to allow  
130036 them to share the associated memory.

- 130037 • Requirements

130038 Shared memory is used to share data among several processes, each potentially running at  
130039 different priority levels, responding to different inputs, or performing separate tasks.  
130040 Shared memory is not just simply providing common access to data, it is providing the  
130041 fastest possible communication between the processes. With one memory write operation,  
130042 a process can pass information to as many processes as have the memory region mapped.

130043 As a result, shared memory provides a mechanism that can be used for all other  
130044 interprocess communication facilities. It may also be used by an application for  
130045 implementing more sophisticated mechanisms than semaphores and message queues.

130046 The need for a shared memory interface is obvious for virtual memory systems, where the  
130047 operating system is directly preventing processes from accessing each other's data.  
130048 However, in unprotected systems, such as those found in some embedded controllers, a  
130049 shared memory interface is needed to provide a portable mechanism to allocate a region of  
130050 memory to be shared and then to communicate the address of that region to other  
130051 processes.

130052 This, then, provides the minimum functionality that a shared memory interface must have  
 130053 in order to support realtime applications: to allocate and name an object to be mapped into  
 130054 memory for potential sharing (*open()* or *shm\_open()*), and to make the memory object  
 130055 available within the address space of a process (*mmap()*). To complete the interface, a  
 130056 mechanism to release the claim of a process on a shared memory object (*munmap()*) is also  
 130057 needed, as well as a mechanism for deleting the name of a sharable object that was  
 130058 previously created (*unlink()* or *shm\_unlink()*).

130059 After a mapping has been established, an implementation should not have to provide  
 130060 services to maintain that mapping. All memory writes into that area will appear  
 130061 immediately in the memory mapping of that region by any other processes.

130062 Thus, requirements include:

- 130063 — Support creation of sharable memory objects and the mapping of these objects into  
 130064 the address space of a process.
- 130065 — Sharable memory objects should be accessed by global names accessible from all  
 130066 processes.
- 130067 — Support the mapping of specific sections of physical address space (such as a  
 130068 memory mapped device) into the address space of a process. This should not be  
 130069 done by the process specifying the actual address, but again by an implementation-  
 130070 defined global name (such as a special device name) dedicated to this purpose.
- 130071 — Support the mapping of discrete portions of these memory objects.
- 130072 — Support for minimum hardware configurations that contain no physical media on  
 130073 which to store shared memory contents permanently.
- 130074 — The ability to preallocate the entire shared memory region so that minimum  
 130075 hardware configurations without virtual memory support can guarantee contiguous  
 130076 space.
- 130077 — The maximizing of performance by not requiring functionality that would require  
 130078 implementation interaction above creating the shared memory area and returning  
 130079 the mapping.

130080 Note that the above requirements do not preclude:

- 130081 — The sharable memory object from being implemented using actual files on an actual  
 130082 file system.
- 130083 — The global name that is accessible from all processes being restricted to a file system  
 130084 area that is dedicated to handling shared memory.
- 130085 — An implementation not providing implementation-defined global names for the  
 130086 purpose of physical address mapping.

#### 130087 • Shared Memory Objects Usage

130088 If the Shared Memory Objects option is supported, a shared memory object may be  
 130089 created, or opened if it already exists, with the *shm\_open()* function. If the shared memory  
 130090 object is created, it has a length of zero. The *ftruncate()* function can be used to set the size  
 130091 of the shared memory object after creation. The *shm\_unlink()* function removes the name  
 130092 for a shared memory object created by *shm\_open()*.

#### 130093 • Shared Memory Overview

130094 The shared memory facility defined by POSIX.1-2024 usually results in memory locations  
 130095 being added to the address space of the process. The implementation returns the address

130096 of the new space to the application by means of a pointer. This works well in languages  
130097 like C. However, in languages without pointer types it will not work. In the bindings for  
130098 such a language, either a special COMMON section will need to be defined (which is  
130099 unlikely), or the binding will have to allow existing structures to be mapped. The  
130100 implementation will likely have to place restrictions on the size and alignment of such  
130101 structures or will have to map a suitable region of the address space of the process into the  
130102 memory object, and thus into other processes. These are issues for that particular language  
130103 binding. For POSIX.1-2024, however, the practice will not be forbidden, merely undefined.

130104 Two potentially different name spaces are used for naming objects that may be mapped  
130105 into process address spaces. When using memory mapped files, files may be accessed via  
130106 *open()*. When the Shared Memory Objects option is supported, sharable memory objects  
130107 that might not be files may be accessed via the *shm\_open()* function. These operations are  
130108 not mutually-exclusive.

130109 Some implementations supporting the Shared Memory Objects option may choose to  
130110 implement the shared memory object name space as part of the file system name space.  
130111 There are several reasons for this:

- 130112 — It allows applications to prevent name conflicts by use of the directory structure.
- 130113 — It uses an existing mechanism for accessing global objects and prevents the creation  
130114 of a new mechanism for naming global objects.

130115 In such implementations, memory objects can be implemented using regular files, if that is  
130116 what the implementation chooses. The *shm\_open()* function can be implemented as an  
130117 *open()* call in a fixed directory with the O\_CLOEXEC flag set. The *shm\_unlink()* function  
130118 can be implemented as an *unlink()* call.

130119 On the other hand, it is also expected that small embedded systems that support the  
130120 Shared Memory Objects option may wish to implement shared memory without having  
130121 any file systems present. In this case, the implementations may choose to use a simple  
130122 string valued name space for shared memory regions. The *shm\_open()* function permits  
130123 either type of implementation.

130124 Some implementations have hardware that supports protection of mapped data from  
130125 certain classes of access and some do not. Systems that supply this functionality support  
130126 the memory protection functionality.

130127 Some implementations restrict size, alignment, and protections to be on *page-size*  
130128 boundaries. If an implementation has no restrictions on size or alignment, it may specify a  
130129 1-byte page size. Applications on implementations that do support larger pages must be  
130130 cognizant of the page size since this is the alignment and protection boundary.

130131 Simple embedded implementations may have a 1-byte page size and only support the  
130132 Shared Memory Objects option. This provides simple shared memory between processes  
130133 without requiring mapping hardware.

130134 POSIX.1-2024 specifically allows a memory object to remain referenced after a close  
130135 because that is existing practice for the *mmap()* function.



130136 **Typed Memory Functions**

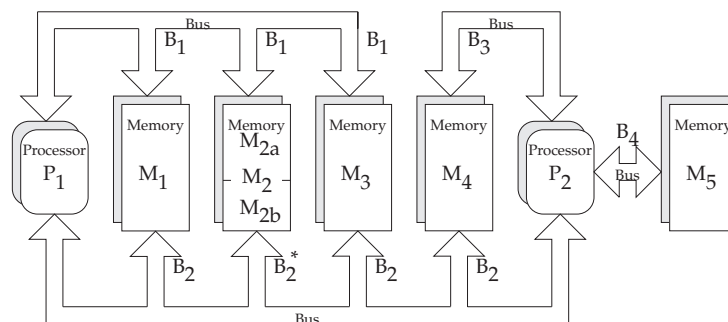
130137 Implementations may support the Typed Memory Objects option without supporting either the  
 130138 Shared Memory option or memory mapped files. Types memory objects are pools of specialized  
 130139 storage, different from the main memory resource normally used by a processor to hold code  
 130140 and data, that can be mapped into the address space of one or more processes.

## 130141 • Model

130142 Realtime systems conforming to one of the POSIX.13 realtime profiles are expected (and  
 130143 desired) to be supported on systems with more than one type or pool of memory (for  
 130144 example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory  
 130145 may be accessible by one or more processors via one or more buses (ports). Memory  
 130146 mapped files, shared memory objects, and the language-specific storage allocation  
 130147 operators (*malloc()* for the ISO C standard, *new* for ISO Ada) fail to provide application  
 130148 program interfaces versatile enough to allow applications to control their utilization of  
 130149 such diverse memory resources. The typed memory interfaces *posix\_typed\_mem\_open()*,  
 130150 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *mmap()*, and *munmap()* defined herein  
 130151 support the model of typed memory described below.

130152 For purposes of this model, a system comprises several processors (for example,  $P_1$  and  
 130153  $P_2$ ), several physical memory pools (for example,  $M_1, M_2, M_{2a}, M_{2b}, M_3, M_4,$  and  $M_5$ ), and  
 130154 several buses or "ports" (for example,  $B_1, B_2, B_3,$  and  $B_4$ ) interconnecting the various  
 130155 processors and memory pools in some system-specific way. Notice that some memory  
 130156 pools may be contained in others (for example,  $M_{2a}$  and  $M_{2b}$  are contained in  $M_2$ ).

130157 **Figure B-1** shows an example of such a model. In a system like this, an application should  
 130158 be able to perform the following operations:



- \* All addresses in pool  $M_2$  (comprising pools  $M_{2a}$  and  $M_{2b}$ ) accessible via port  $B_1$ .
- Addresses in pool  $M_{2b}$  are also accessible via port  $B_2$ .
- Addresses in pool  $M_{2a}$  are *not* accessible via port  $B_2$ .

130159 **Figure B-1** Example of a System with Typed Memory

## 130160 — Typed Memory Allocation

130161 An application should be able to allocate memory dynamically from the desired pool  
 130162 using the desired bus, and map it into the address space of a process. For example,  
 130163 processor  $P_1$  can allocate some portion of memory pool  $M_1$  through port  $B_1$ , treating  
 130164 all unmapped subareas of  $M_1$  as a heap-storage resource from which memory may be  
 130165 allocated. This portion of memory is mapped into address space of the process, and  
 130166 subsequently deallocated when unmapped from all processes.

- 130167 — Using the Same Storage Region from Different Buses
- 130168 An application process with a mapped region of storage that is accessed from one  
130169 bus should be able to map that same storage area at another address (subject to page  
130170 size restrictions detailed in *mmap()*), to allow it to be accessed from another bus. For  
130171 example, processor  $P_1$  may wish to access the same region of memory pool  $M_{2b}$  both  
130172 through ports  $B_1$  and  $B_2$ .
- 130173 — Sharing Typed Memory Regions
- 130174 Several application processes running on the same or different processors may wish  
130175 to share a particular region of a typed memory pool. Each process or processor may  
130176 wish to access this region through different buses. For example, processor  $P_1$  may  
130177 want to share a region of memory pool  $M_4$  with processor  $P_2$ , and they may be  
130178 required to use buses  $B_2$  and  $B_3$ , respectively, to minimize bus contention. A problem  
130179 arises here when a process allocates and maps a portion of fragmented memory and  
130180 then wants to share this region of memory with another process, either in the same  
130181 processor or different processors. The solution adopted is to allow the first process to  
130182 find out the memory map (offsets and lengths) of all the different fragments of  
130183 memory that were mapped into its address space, by repeatedly calling  
130184 *posix\_mem\_offset()*. Then, this process can pass the offsets and lengths obtained to  
130185 the second process, which can then map the same memory fragments into its address  
130186 space.
- 130187 — Contiguous Allocation
- 130188 The problem of finding the memory map of the different fragments of the memory  
130189 pool that were mapped into logically contiguous addresses of a given process can be  
130190 solved by requesting contiguous allocation. For example, a process in  $P_1$  can allocate  
130191 10 Kbytes of physically contiguous memory from  $M_3-B_1$ , and obtain the offset (within  
130192 pool  $M_3$ ) of this block of memory. Then, it can pass this offset (and the length) to a  
130193 process in  $P_2$  using some interprocess communication mechanism. The second  
130194 process can map the same block of memory by using the offset transferred and  
130195 specifying  $M_3-B_2$ .
- 130196 — Unallocated Mapping
- 130197 Any subarea of a memory pool that is mapped to a process, either as the result of an  
130198 allocation request or an explicit mapping, is normally unavailable for allocation.  
130199 Special processes such as debuggers, however, may need to map large areas of a  
130200 typed memory pool, yet leave those areas available for allocation.
- 130201 Typed memory allocation and mapping has to coexist with storage allocation operators  
130202 like *malloc()*, but systems are free to choose how to implement this coexistence. For  
130203 example, it may be system configuration-dependent if all available system memory is  
130204 made part of one of the typed memory pools or if some part will be restricted to  
130205 conventional allocation operators. Equally system configuration-dependent may be the  
130206 availability of operators like *malloc()* to allocate storage from certain typed memory pools.  
130207 It is not excluded to configure a system such that a given named pool,  $P_1$ , is in turn split  
130208 into non-overlapping named subpools. For example,  $M_1-B_1$ ,  $M_2-B_1$ , and  $M_3-B_1$  could also be  
130209 accessed as one common pool  $M_{123}-B_1$ . A call to *malloc()* on  $P_1$  could work on such a larger  
130210 pool while full optimization of memory usage by  $P_1$  would require typed memory  
130211 allocation at the subpool level.
- 130212 • Existing Practice
- 130213 OS-9 provides for the naming (numbering) and prioritization of memory types by a system  
130214 administrator. It then provides APIs to request memory allocation of typed (colored)

130215 memory by number, and to generate a bus address from a mapped memory address  
 130216 (translate). When requesting colored memory, the user can specify type 0 to signify  
 130217 allocation from the first available type in priority order.

130218 HP-RT presents interfaces to map different kinds of storage regions that are visible through  
 130219 a VME bus, although it does not provide allocation operations. It also provides functions  
 130220 to perform address translation between VME addresses and virtual addresses. It represents  
 130221 a VME-bus unique solution to the general problem.

130222 The PSOS approach is similar (that is, based on a pre-established mapping of bus address  
 130223 ranges to specific memories) with a concept of segments and regions (regions dynamically  
 130224 allocated from a heap which is a special segment). Therefore, PSOS does not fully address  
 130225 the general allocation problem either. PSOS does not have a "process"-based model, but  
 130226 more of a "thread"-only-based model of multi-tasking. So mapping to a process address  
 130227 space is not an issue.

130228 QNX uses the System V approach of opening specially named devices (shared memory  
 130229 segments) and using *mmap()* to then gain access from the process. They do not address  
 130230 allocation directly, but once typed shared memory can be mapped, an "allocation  
 130231 manager" process could be written to handle requests for allocation.

130232 The System V approach also included allocation, implemented by opening yet other  
 130233 special "devices" which allocate, rather than appearing as a whole memory object.

130234 The Orkid realtime kernel interface definition has operations to manage memory "regions"  
 130235 and "pools", which are areas of memory that may reflect the differing physical nature of  
 130236 the memory. Operations to allocate memory from these regions and pools are also  
 130237 provided.

#### 130238 • Requirements

130239 Existing practice in SVID-derived UNIX systems relies on functionality similar to *mmap()*  
 130240 and its related interfaces to achieve mapping and allocation of typed memory. However,  
 130241 the issue of sharing typed memory (allocated or mapped) and the complication of multiple  
 130242 ports are not addressed in any consistent way by existing UNIX system practice. Part of  
 130243 this functionality is existing practice in specialized realtime operating systems. In order to  
 130244 solidify the capabilities implied by the model above, the following requirements are  
 130245 imposed on the interface:

#### 130246 — Identification of Typed Memory Pools and Ports

130247 All processes (running in all processors) in the system are able to identify a particular  
 130248 (system configured) typed memory pool accessed through a particular (system  
 130249 configured) port by a name. That name is a member of a name space common to all  
 130250 these processes, but need not be the same name space as that containing ordinary  
 130251 pathnames. The association between memory pools/ports and corresponding names  
 130252 is typically established when the system is configured. The "open" operation for  
 130253 typed memory objects should be distinct from the *open()* function, for consistency  
 130254 with other similar services, but implementable on top of *open()*. This implies that the  
 130255 handle for a typed memory object will be a file descriptor.

#### 130256 — Allocation and Mapping of Typed Memory

130257 Once a typed memory object has been identified by a process, it is possible to both  
 130258 map user-selected subareas of that object into process address space and to map  
 130259 system-selected (that is, dynamically allocated) subareas of that object, with user-  
 130260 specified length, into process address space. It is also possible to determine the  
 130261 maximum length of memory allocation that may be requested from a given typed

- 130262 memory object.
- 130263 — Sharing Typed Memory
- 130264 Two or more processes are able to share portions of typed memory, either user-
- 130265 selected or dynamically allocated. This requirement applies also to dynamically
- 130266 allocated regions of memory that are composed of several non-contiguous pieces.
- 130267 — Contiguous Allocation
- 130268 For dynamic allocation, it is the user's option whether the system is required to
- 130269 allocate a contiguous subarea within the typed memory object, or whether it is
- 130270 permitted to allocate discontinuous fragments which appear contiguous in the
- 130271 process mapping. Contiguous allocation simplifies the process of sharing allocated
- 130272 typed memory, while discontinuous allocation allows for potentially better recovery
- 130273 of deallocated typed memory.
- 130274 — Accessing Typed Memory Through Different Ports
- 130275 Once a subarea of a typed memory object has been mapped, it is possible to
- 130276 determine the location and length corresponding to a user-selected portion of that
- 130277 object within the memory pool. This location and length can then be used to remap
- 130278 that portion of memory for access from another port. If the referenced portion of
- 130279 typed memory was allocated discontinuously, the length thus determined may be
- 130280 shorter than anticipated, and the user code must adapt to the value returned.
- 130281 — Deallocation
- 130282 When a previously mapped subarea of typed memory is no longer mapped by any
- 130283 process in the system—as a result of a call or calls to *munmap()*—that subarea
- 130284 becomes potentially reusable for dynamic allocation; actual reuse of the subarea is a
- 130285 function of the dynamic typed memory allocation policy.
- 130286 — Unallocated Mapping
- 130287 It must be possible to map user-selected subareas of a typed memory object without
- 130288 marking that subarea as unavailable for allocation. This option is not the default
- 130289 behavior, and requires appropriate privileges.
- 130290 • Scenario
- 130291 The following scenario will serve to clarify the use of the typed memory interfaces.
- 130292 Process A running on  $P_1$  (see [Figure B-1](#), on page 3777) wants to allocate some memory
- 130293 from memory pool  $M_2$ , and it wants to share this portion of memory with process B
- 130294 running on  $P_2$ . Since  $P_2$  only has access to the lower part of  $M_2$ , both processes will use the
- 130295 memory pool named  $M_{2b}$  which is the part of  $M_2$  that is accessible both from  $P_1$  and  $P_2$ . The
- 130296 operations that both processes need to perform are shown below:
- 130297 — Allocating Typed Memory
- 130298 Process A calls *posix\_typed\_mem\_open()* with the name **/typed.m2b-b1** and a *tflag* of
- 130299 POSIX\_TYPED\_MEM\_ALLOCATE to get a file descriptor usable for allocating from
- 130300 pool  $M_{2b}$  accessed through port  $B_1$ . It then calls *mmap()* with this file descriptor
- 130301 requesting a length of 4 096 bytes. The system allocates two discontinuous blocks of
- 130302 sizes 1 024 and 3 072 bytes within  $M_{2b}$ . The *mmap()* function returns a pointer to a
- 130303 4 096-byte array in process A's logical address space, mapping the allocated blocks
- 130304 contiguously. Process A can then utilize the array, and store data in it.

## 130305 — Determining the Location of the Allocated Blocks

130306 Process A can determine the lengths and offsets (relative to  $M_{2b}$ ) of the two blocks  
 130307 allocated, by using the following procedure: First, process A calls `posix_mem_offset()`  
 130308 with the address of the first element of the array and length 4096. Upon return, the  
 130309 offset and length (1024 bytes) of the first block are returned. A second call to  
 130310 `posix_mem_offset()` is then made using the address of the first element of the array  
 130311 plus 1024 (the length of the first block), and a new length of 4096–1024. If there were  
 130312 more fragments allocated, this procedure could have been continued within a loop  
 130313 until the offsets and lengths of all the blocks were obtained. Notice that this relatively  
 130314 complex procedure can be avoided if contiguous allocation is requested (by opening  
 130315 the typed memory object with the `tflag`  
 130316 `POSIX_TYPED_MEM_ALLOCATE_CONTIG`).

## 130317 — Sharing Data Across Processes

130318 Process A passes the two offset values and lengths obtained from the  
 130319 `posix_mem_offset()` calls to process B running on  $P_2$ , via some form of interprocess  
 130320 communication. Process B can gain access to process A's data by calling  
 130321 `posix_typed_mem_open()` with the name `/typed.m2b-b2` and a `tflag` of zero, then using  
 130322 two `mmap()` calls on the resulting file descriptor to map the two subareas of that  
 130323 typed memory object to its own address space.

130324 • Rationale for no `mem_alloc()` and `mem_free()`

130325 The standard developers had originally proposed a pair of new flags to `mmap()` which,  
 130326 when applied to a typed memory object descriptor, would cause `mmap()` to allocate  
 130327 dynamically from an unallocated and unmapped area of the typed memory object.  
 130328 Deallocation was similarly accomplished through the use of `munmap()`. This was rejected  
 130329 by the ballot group because it excessively complicated the (already rather complex)  
 130330 `mmap()` interface and introduced semantics useful only for typed memory, to a function  
 130331 which must also map shared memory and files. They felt that a memory allocator should  
 130332 be built on top of `mmap()` instead of being incorporated within the same interface, much as  
 130333 the ISO C standard libraries build `malloc()` on top of the virtual memory mapping  
 130334 functions `brk()` and `sbrk()`. This would eliminate the complicated semantics involved with  
 130335 unmapping only part of an allocated block of typed memory.

130336 To attempt to achieve ballot group consensus, typed memory allocation and deallocation  
 130337 was first migrated from `mmap()` and `munmap()` to a pair of complementary functions  
 130338 modeled on the ISO C standard `malloc()` and `free()`. The `mem_alloc()` function specified  
 130339 explicitly the typed memory object (typed memory pool/access port) from which  
 130340 allocation takes place, unlike `malloc()` where the memory pool and port are unspecified.  
 130341 The `mem_free()` function handled deallocation. These new semantics still met all of the  
 130342 requirements detailed above without modifying the behavior of `mmap()` except to allow it  
 130343 to map specified areas of typed memory objects. An implementation would have been free  
 130344 to implement `mem_alloc()` and `mem_free()` over `mmap()`, through `mmap()`, or independently  
 130345 but cooperating with `mmap()`.

130346 The ballot group was queried to see if this was an acceptable alternative, and while there  
 130347 was some agreement that it achieved the goal of removing the complicated semantics of  
 130348 allocation from the `mmap()` interface, several balloters realized that it just created two  
 130349 additional functions that behaved, in great part, like `mmap()`. These balloters proposed an  
 130350 alternative which has been implemented here in place of a separate `mem_alloc()` and  
 130351 `mem_free()`. This alternative is based on four specific suggestions:

- 130352 1. The `posix_typed_mem_open()` function should provide a flag which specifies  
 130353 ``allocate on `mmap()`'' (otherwise, `mmap()` just maps the underlying object). This  
 130354 allows things roughly similar to `/dev/zero` versus `/dev/swap`. Two such flags have  
 130355 been implemented, one of which forces contiguous allocation.
- 130356 2. The `posix_mem_offset()` function is acceptable because it can be applied usefully to  
 130357 mapped objects in general. It should return the file descriptor of the underlying  
 130358 object.
- 130359 3. The `mem_get_info()` function in an earlier draft should be renamed  
 130360 `posix_typed_mem_get_info()` because it is not generally applicable to memory objects.  
 130361 It should probably return the file descriptor's allocation attribute. The renaming of  
 130362 the function has been implemented, but having it return a piece of information  
 130363 which is readily known by an application without this function has been rejected.  
 130364 Its whole purpose is to query the typed memory object for attributes that are not  
 130365 user-specified, but determined by the implementation.
- 130366 4. There should be no separate `mem_alloc()` or `mem_free()` functions. Instead, using  
 130367 `mmap()` on a typed memory object opened with an ``allocate on `mmap()`'' flag  
 130368 should be used to force allocation. These are precisely the semantics defined in the  
 130369 current draft.

130370 • Rationale for no Typed Memory Access Management

130371 The working group had originally defined an additional interface (and an additional kind  
 130372 of object: typed memory manager) to establish and dissolve mappings to typed memory  
 130373 on behalf of devices or processors which were independent of the operating system and  
 130374 had no inherent capability to directly establish mappings on their own. This was to have  
 130375 provided functionality similar to device driver interfaces such as `physio()` and their  
 130376 underlying bus-specific interfaces (for example, `mballoc()`) which serve to set up and break  
 130377 down DMA pathways, and derive mapped addresses for use by hardware devices and  
 130378 processor cards.

130379 The ballot group felt that this was beyond the scope of POSIX.1 and its amendments.  
 130380 Furthermore, the removal of interrupt handling interfaces from a preceding amendment  
 130381 (the IEEE Std 1003.1d-1999) during its balloting process renders these typed memory  
 130382 access management interfaces an incomplete solution to portable device management from  
 130383 a user process; it would be possible to initiate a device transfer to/from typed memory, but  
 130384 impossible to handle the transfer-complete interrupt in a portable way.

130385 To achieve ballot group consensus, all references to typed memory access management  
 130386 capabilities were removed. The concept of portable interfaces from a device driver to both  
 130387 operating system and hardware is being addressed by the Uniform Driver Interface (UDI)  
 130388 industry forum, with formal standardization deferred until proof of concept and industry-  
 130389 wide acceptance and implementation.

130390 B.2.8.4 Process Scheduling

130391 IEEE PASC Interpretation 1003.1 #96 has been applied, adding the `pthread_setschedprio()`  
 130392 function. This was added since previously there was no way for a thread to lower its own  
 130393 priority without going to the tail of the threads list for its new priority. This capability is  
 130394 necessary to bound the duration of priority inversion encountered by a thread.

130395 The following portion of the rationale presents models, requirements, and standardization  
 130396 issues relevant to process and thread scheduling; see [Section B.2.9.4](#) (on page 3824) for  
 130397 additional rationale relevant to thread scheduling.

130398 In an operating system supporting multiple concurrent processes or threads, the system  
130399 determines the order in which processes or threads execute to meet implementation-defined  
130400 goals. For time-sharing systems, the goal is to enhance system throughput and promote fairness;  
130401 the application is provided with little or no control over this sequencing function. While this is  
130402 acceptable and desirable behavior in a time-sharing system, it is inappropriate in a realtime  
130403 system; realtime applications must specifically control the execution sequence of their  
130404 concurrent processes or threads in order to meet externally defined response requirements.

130405 In POSIX.1-2024, the control over process and thread sequencing is provided using a concept of  
130406 scheduling policies. These policies, described in detail in this section, define the behavior of the  
130407 system whenever processor resources are to be allocated to competing processes or threads.  
130408 Only the behavior of the policy is defined; conforming implementations are free to use any  
130409 mechanism desired to achieve the described behavior.

130410 • Models

130411 In an operating system supporting multiple concurrent processes or threads, the system  
130412 determines the order in which threads (including those that are the only thread in a single-  
130413 threaded process) execute and might force long-running threads to yield to other threads  
130414 at certain intervals. Typically, the scheduling code is executed whenever an event occurs  
130415 that might alter the thread to be executed next.

130416 The simplest scheduling strategy is a “first-in, first-out” (FIFO) dispatcher. Whenever a  
130417 thread becomes runnable, it is placed on the end of a ready list. When processing resources  
130418 become available, the thread at the front of the ready list starts or resumes execution and is  
130419 removed from the list. This thread is executed until it exits or becomes blocked, at which  
130420 point the processing resources used to execute it become available to execute another  
130421 runnable thread. This scheduling technique is also known as “run-to-completion” or “run-  
130422 to-block”.

130423 A natural extension to this scheduling technique is the assignment of a “non-migrating  
130424 priority” to each thread. This policy differs from strict FIFO scheduling in only one respect:  
130425 whenever a thread becomes runnable, it is placed at the end of the list of threads runnable  
130426 at that priority level. When selecting a thread to run, the system always selects the first  
130427 thread from the highest priority queue with a runnable thread. Thus, when a thread  
130428 becomes unblocked, it will preempt a running thread of lower priority without otherwise  
130429 altering the ready list. Further, if a running or runnable thread’s priority is altered, it is  
130430 removed from the ready list for its old priority (if present in the list; that is, not running)  
130431 and is inserted into the ready list for its new priority, according to the policy above, except  
130432 that threads executing at a temporarily elevated priority as a consequence of owning a  
130433 mutex initialized with the PTHREAD\_PRIO\_INHERIT or PTHREAD\_PRIO\_PROTECT  
130434 protocol are exempted from this in order to ensure that a thread can lock and unlock such  
130435 as mutex without the implicit yield that any resulting priority changes would normally  
130436 cause.

130437 While the above policy might be considered unfriendly in a time-sharing environment in  
130438 which multiple users require more balanced resource allocation, it could be ideal in a  
130439 realtime environment for several reasons. The most important of these is that it is  
130440 deterministic: the highest-priority thread is always run and, among threads of equal  
130441 priority, the thread that has been runnable for the longest time is executed first. Because of  
130442 this determinism, cooperating threads can implement more complex scheduling simply by  
130443 altering their priority. For instance, if threads at a single priority were to reschedule  
130444 themselves at fixed time intervals, a time-slice policy would result.

130445 In a dedicated operating system in which all threads belong to well-behaved realtime  
130446 applications, non-migrating priority scheduling is sufficient. However, many existing

- 130447 implementations provide for more complex scheduling policies.
- 130448 For process scheduling, POSIX.1-2024 specifies a linear scheduling model. In this model,  
130449 every process in the system has a priority. The system scheduler always dispatches a  
130450 process that has the highest (generally the most time-critical) priority among all runnable  
130451 processes in the system. As long as there is only one such process, the dispatching policy is  
130452 trivial. When multiple processes of equal priority are eligible to run, they are ordered  
130453 according to a strict run-to-completion (FIFO) policy. Thread scheduling is similar, except  
130454 that the scheduling policy can be applied just to the threads within one process  
130455 (PTHREAD\_SCOPE\_PROCESS scheduling contention scope) or to all threads system-wide  
130456 (PTHREAD\_SCOPE\_SYSTEM scheduling contention scope). This and other considerations  
130457 specific to thread scheduling are the subject of [Section B.2.9.4](#) (on page 3824); the  
130458 remainder of this section is described in terms of process scheduling but is also relevant to  
130459 thread scheduling when read in conjunction with [Section B.2.9.4](#).
- 130460 The priority is represented as a positive integer and is inherited from the parent process.  
130461 For processes running under a fixed priority scheduling policy, the priority is never altered  
130462 except by an explicit function call.
- 130463 It was determined arbitrarily that larger integers correspond to “higher priorities”.
- 130464 Certain implementations might impose restrictions on the priority ranges to which  
130465 processes can be assigned. There also can be restrictions on the set of policies to which  
130466 processes can be set.
- 130467 • Requirements
- 130468 Realtime processes require that scheduling be fast and deterministic, and that it guarantees  
130469 to preempt lower priority processes.
- 130470 Thus, given the linear scheduling model, realtime processes require that they be run at a  
130471 priority that is higher than other processes. Within this framework, realtime processes are  
130472 free to yield execution resources to each other in a completely portable and  
130473 implementation-defined manner.
- 130474 As there is a generally perceived requirement for processes at the same priority level to  
130475 share processor resources more equitably, provisions are made by providing a scheduling  
130476 policy (that is, SCHED\_RR) intended to provide a timeslice-like facility.
- 130477 **Note:** The following topics assume that low numeric priority implies low scheduling criticality  
130478 and *vice versa*.
- 130479 • Rationale for New Interface
- 130480 Realtime applications need to be able to determine when processes will run in relation to  
130481 each other. It must be possible to guarantee that a critical process will run whenever it is  
130482 runnable; that is, whenever it wants to for as long as it needs. SCHED\_FIFO satisfies this  
130483 requirement. Additionally, SCHED\_RR was defined to meet a realtime requirement for a  
130484 well-defined time-sharing policy for processes at the same priority.
- 130485 It would be possible to use the BSD *setpriority()* and *getpriority()* functions by redefining  
130486 the meaning of the “nice” parameter according to the scheduling policy currently in use by  
130487 the process. The System V *nice()* interface was felt to be undesirable for realtime because it  
130488 specifies an adjustment to the “nice” value, rather than setting it to an explicit value.  
130489 Realtime applications will usually want to set priority to an explicit value. Also, System V  
130490 *nice()* does not allow for changing the priority of another process.
- 130491 With the POSIX.1b interfaces, the traditional “nice” value does not affect the SCHED\_FIFO  
130492 or SCHED\_RR scheduling policies. If a “nice” value is supported, it is implementation-  
130493 defined whether it affects the SCHED\_OTHER policy.



130494 An important aspect of POSIX.1-2024 is the explicit description of the queuing and  
130495 preemption rules. It is critical, to achieve deterministic scheduling, that such rules be  
130496 stated clearly in POSIX.1-2024.

130497 POSIX.1-2024 does not address the interaction between priority and swapping. The issues  
130498 involved with swapping and virtual memory paging are extremely implementation-  
130499 defined and would be nearly impossible to standardize at this point. The proposed  
130500 scheduling paradigm, however, fully describes the scheduling behavior of runnable  
130501 processes, of which one criterion is that the working set be resident in memory. Assuming  
130502 the existence of a portable interface for locking portions of a process in memory, paging  
130503 behavior need not affect the scheduling of realtime processes.

130504 POSIX.1-2024 also does not address the priorities of “system” processes. In general, these  
130505 processes should always execute in low-priority ranges to avoid conflict with other  
130506 realtime processes. Implementations should document the priority ranges in which system  
130507 processes run.

130508 The default scheduling policy is not defined. The effect of I/O interrupts and other system  
130509 processing activities is not defined. The temporary lending of priority from one process to  
130510 another (such as for the purposes of affecting freeing resources) by the system is not  
130511 addressed. Preemption of resources is not addressed. Restrictions on the ability of a  
130512 process to affect other processes beyond a certain level (influence levels) is not addressed.

130513 The rationale used to justify the simple time-quantum scheduler is that it is common  
130514 practice to depend upon this type of scheduling to ensure “fair” distribution of processor  
130515 resources among portions of the application that must interoperate in a serial fashion. Note  
130516 that POSIX.1-2024 is silent with respect to the setting of this time quantum, or whether it is  
130517 a system-wide value or a per-process value, although it appears that the prevailing  
130518 realtime practice is for it to be a system-wide value.

130519 In a system with  $N$  processes at a given priority, all processor-bound, in which the time  
130520 quantum is equal for all processes at a specific priority level, the following assumptions  
130521 are made of such a scheduling policy:

- 130522 1. A time quantum  $Q$  exists and the current process will own control of the processor  
130523 for at least a duration of  $Q$  and will have the processor for a duration of  $Q$ .
- 130524 2. The  $N$ th process at that priority will control a processor within a duration of  $(N-1)$   
130525  $\times Q$ .

130526 These assumptions are necessary to provide equal access to the processor and bounded  
130527 response from the application.

130528 The assumptions hold for the described scheduling policy only if no system overhead,  
130529 such as interrupt servicing, is present. If the interrupt servicing load is non-zero, then one  
130530 of the two assumptions becomes fallacious, based upon how  $Q$  is measured by the system.

130531 If  $Q$  is measured by clock time, then the assumption that the process obtains a duration  $Q$   
130532 processor time is false if interrupt overhead exists. Indeed, a scenario can be constructed  
130533 with  $N$  processes in which a single process undergoes complete processor starvation if a  
130534 peripheral device, such as an analog-to-digital converter, generates significant interrupt  
130535 activity periodically with a period of  $N \times Q$ .

130536 If  $Q$  is measured as actual processor time, then the assumption that the  $N$ th process runs in  
130537 within the duration  $(N-1) \times Q$  is false.

130538 It should be noted that SCHED\_FIFO suffers from interrupt-based delay as well. However,  
130539 for SCHED\_FIFO, the implied response of the system is “as soon as possible”, so that the  
130540 interrupt load for this case is a vendor selection and not a compliance issue.

130541 With this in mind, it is necessary either to complete the definition by including bounds on  
 130542 the interrupt load, or to modify the assumptions that can be made about the scheduling  
 130543 policy.

130544 Since the motivation of inclusion of the policy is common usage, and since current  
 130545 applications do not enjoy the luxury of bounded interrupt load, item (2) above is sufficient  
 130546 to express existing application needs and is less restrictive in the standard definition. No  
 130547 difference in interface is necessary.

130548 In an implementation in which the time quantum is equal for all processes at a specific  
 130549 priority, our assumptions can then be restated as:

- 130550 — A time quantum  $Q$  exists, and a processor-bound process will be rescheduled after a  
 130551 duration of, at most,  $Q$ . Time quantum  $Q$  may be defined in either wall clock time or  
 130552 execution time.

- 130553 — In general, the  $N$ th process of a priority level should wait no longer than  $(N-1) \times Q$   
 130554 time to execute, assuming no processes exist at higher priority levels.

- 130555 — No process should wait indefinitely.

130556 For implementations supporting per-process time quanta, these assumptions can be  
 130557 readily extended.

130558 Austin Group Defect 1302 is applied, making requirements on `sched_yield()` also apply to  
 130559 `thrd_yield()`.

130560 Austin Group Defect 1610 is applied, clarifying the effects of `PTHREAD_PRIO_INHERIT` and  
 130561 `PTHREAD_PRIO_PROTECT` on scheduling queues.

### 130562 Sporadic Server Scheduling Policy

130563 The sporadic server is a mechanism defined for scheduling aperiodic activities in time-critical  
 130564 realtime systems. This mechanism reserves a certain bounded amount of execution capacity for  
 130565 processing aperiodic events at a high priority level. Any aperiodic events that cannot be  
 130566 processed within the bounded amount of execution capacity are executed in the background at a  
 130567 low priority level. Thus, a certain amount of execution capacity can be guaranteed to be  
 130568 available for processing periodic tasks, even under burst conditions in the arrival of aperiodic  
 130569 processing requests (that is, a large number of requests in a short time interval). The sporadic  
 130570 server also simplifies the schedulability analysis of the realtime system, because it allows  
 130571 aperiodic processes or threads to be treated as if they were periodic. The sporadic server was  
 130572 first described by Sprunt, et al.

130573 The key concept of the sporadic server is to provide and limit a certain amount of computation  
 130574 capacity for processing aperiodic events at their assigned normal priority, during a time interval  
 130575 called the “replenishment period”. Once the entity controlled by the sporadic server mechanism  
 130576 is initialized with its period and execution-time budget attributes, it preserves its execution  
 130577 capacity until an aperiodic request arrives. The request will be serviced (if there are no higher  
 130578 priority activities pending) as long as there is execution capacity left. If the request is completed,  
 130579 the actual execution time used to service it is subtracted from the capacity, and a replenishment  
 130580 of this amount of execution time is scheduled to happen one replenishment period after the  
 130581 arrival of the aperiodic request. If the request is not completed, because there is no execution  
 130582 capacity left, then the aperiodic process or thread is assigned a lower background priority. For  
 130583 each portion of consumed execution capacity the execution time used is replenished after one  
 130584 replenishment period. At the time of replenishment, if the sporadic server was executing at a  
 130585 background priority level, its priority is elevated to the normal level. Other similar  
 130586 replenishment policies have been defined, but the one presented here represents a compromise  
 130587 between efficiency and implementation complexity.

130588 The interface that appears in this section defines a new scheduling policy for threads and  
130589 processes that behaves according to the rules of the sporadic server mechanism. Scheduling  
130590 attributes are defined and functions are provided to allow the user to set and get the parameters  
130591 that control the scheduling behavior of this mechanism, namely the normal and low priority, the  
130592 replenishment period, the maximum number of pending replenishment operations, and the  
130593 initial execution-time budget.

130594 • Scheduling Aperiodic Activities

130595 Virtually all realtime applications are required to process aperiodic activities. In many  
130596 cases, there are tight timing constraints that the response to the aperiodic events must  
130597 meet. Usual timing requirements imposed on the response to these events are:

130598 — The effects of an aperiodic activity on the response time of lower priority activities  
130599 must be controllable and predictable.

130600 — The system must provide the fastest possible response time to aperiodic events.

130601 — It must be possible to take advantage of all the available processing bandwidth not  
130602 needed by time-critical activities to enhance average-case response times to aperiodic  
130603 events.

130604 Traditional methods for scheduling aperiodic activities are background processing, polling  
130605 tasks, and direct event execution:

130606 — Background processing consists of assigning a very low priority to the processing of  
130607 aperiodic events. It utilizes all the available bandwidth in the system that has not  
130608 been consumed by higher priority threads. However, it is very difficult, or  
130609 impossible, to meet requirements on average-case response time, because the  
130610 aperiodic entity has to wait for the execution of all other entities which have higher  
130611 priority.

130612 — Polling consists of creating a periodic process or thread for servicing aperiodic  
130613 requests. At regular intervals, the polling entity is started and its services  
130614 accumulated pending aperiodic requests. If no aperiodic requests are pending, the  
130615 polling entity suspends itself until its next period. Polling allows the aperiodic  
130616 requests to be processed at a higher priority level. However, worst and average-case  
130617 response times of polling entities are a direct function of the polling period, and there  
130618 is execution overhead for each polling period, even if no event has arrived. If the  
130619 deadline of the aperiodic activity is short compared to the inter-arrival time, the  
130620 polling frequency must be increased to guarantee meeting the deadline. For this case,  
130621 the increase in frequency can dramatically reduce the efficiency of the system and,  
130622 therefore, its capacity to meet all deadlines. Yet, polling represents a good way to  
130623 handle a large class of practical problems because it preserves system predictability,  
130624 and because the amortized overhead drops as load increases.

130625 — Direct event execution consists of executing the aperiodic events at a high fixed-  
130626 priority level. Typically, the aperiodic event is processed by an interrupt service  
130627 routine as soon as it arrives. This technique provides predictable response times for  
130628 aperiodic events, but makes the response times of all lower priority activities  
130629 completely unpredictable under burst arrival conditions. Therefore, if the density of  
130630 aperiodic event arrivals is unbounded, it may be a dangerous technique for time-  
130631 critical systems. Yet, for those cases in which the physics of the system imposes a  
130632 bound on the event arrival rate, it is probably the most efficient technique.

130633 — The sporadic server scheduling algorithm combines the predictability of the polling  
130634 approach with the short response times of the direct event execution. Thus, it allows  
130635 systems to meet an important class of application requirements that cannot be met by

130636 using the traditional approaches. Multiple sporadic servers with different attributes  
130637 can be applied to the scheduling of multiple classes of aperiodic events, each with  
130638 different kinds of timing requirements, such as individual deadlines, average  
130639 response times, and so on. It also has many other interesting applications for  
130640 realtime, such as scheduling producer/consumer tasks in time-critical systems,  
130641 limiting the effects of faults on the estimation of task execution-time requirements,  
130642 and so on.

130643 • Existing Practice

130644 The sporadic server has been used in different kinds of applications, including military  
130645 avionics, robot control systems, industrial automation systems, and so on. There are  
130646 examples of many systems that cannot be successfully scheduled using the classic  
130647 approaches, such as direct event execution, or polling, and are schedulable using a  
130648 sporadic server scheduler. The sporadic server algorithm itself can successfully schedule  
130649 all systems scheduled with direct event execution or polling.

130650 The sporadic server scheduling policy has been implemented as a commercial product in  
130651 the run-time system of the Verdex Ada compiler. There are also many applications that  
130652 have used a much less efficient application-level sporadic server. These realtime  
130653 applications would benefit from a sporadic server scheduler implemented at the scheduler  
130654 level.

130655 • Library-Level *versus* Kernel-Level Implementation

130656 The sporadic server interface described in this section requires the sporadic server policy  
130657 to be implemented at the same level as the scheduler. This means that the process sporadic  
130658 server must be implemented at the kernel level and the thread sporadic server policy  
130659 implemented at the same level as the thread scheduler; that is, kernel or library level.

130660 In an earlier interface for the sporadic server, this mechanism was implementable at a  
130661 different level than the scheduler. This feature allowed the implementor to choose between  
130662 an efficient scheduler-level implementation, or a simpler user or library-level  
130663 implementation. However, the working group considered that this interface made the use  
130664 of sporadic servers more complex, and that library-level implementations would lack some  
130665 of the important functionality of the sporadic server, namely the limitation of the actual  
130666 execution time of aperiodic activities. The working group also felt that the interface  
130667 described in this chapter does not preclude library-level implementations of threads  
130668 intended to provide efficient low-overhead scheduling for those threads that are not  
130669 scheduled under the sporadic server policy.

130670 • Range of Scheduling Priorities

130671 Each of the scheduling policies supported in POSIX.1-2024 has an associated range of  
130672 priorities. The priority ranges for each policy might or might not overlap with the priority  
130673 ranges of other policies. For time-critical realtime applications it is usual for periodic and  
130674 aperiodic activities to be scheduled together in the same processor. Periodic activities will  
130675 usually be scheduled using the SCHED\_FIFO scheduling policy, while aperiodic activities  
130676 may be scheduled using SCHED\_SPORADIC. Since the application developer will require  
130677 complete control over the relative priorities of these activities in order to meet his timing  
130678 requirements, it would be desirable for the priority ranges of SCHED\_FIFO and  
130679 SCHED\_SPORADIC to overlap completely. Therefore, although POSIX.1-2024 does not  
130680 require any particular relationship between the different priority ranges, it is  
130681 recommended that these two ranges should coincide.

- 130682
- Dynamically Setting the Sporadic Server Policy
- 130683 Several members of the working group requested that implementations should not be  
 130684 required to support dynamically setting the sporadic server scheduling policy for a thread.  
 130685 The reason is that this policy may have a high overhead for library-level implementations  
 130686 of threads, and if threads are allowed to dynamically set this policy, this overhead can be  
 130687 experienced even if the thread does not use that policy. By disallowing the dynamic setting  
 130688 of the sporadic server scheduling policy, these implementations can accomplish efficient  
 130689 scheduling for threads using other policies. If a strictly conforming application needs to  
 130690 use the sporadic server policy, and is therefore willing to pay the overhead, it must set this  
 130691 policy at the time of thread creation.
- 130692
- Limitation of the Number of Pending Replenishments
- 130693 The number of simultaneously pending replenishment operations must be limited for each  
 130694 sporadic server for two reasons: an unlimited number of replenishment operations would  
 130695 need an unlimited number of system resources to store all the pending replenishment  
 130696 operations; on the other hand, in some implementations each replenishment operation will  
 130697 represent a source of priority inversion (just for the duration of the replenishment  
 130698 operation) and thus, the maximum amount of replenishments must be bounded to  
 130699 guarantee bounded response times. The way in which the number of replenishments is  
 130700 bounded is by lowering the priority of the sporadic server to *sched\_ss\_low\_priority* when  
 130701 the number of pending replenishments has reached its limit. In this way, no new  
 130702 replenishments are scheduled until the number of pending replenishments decreases.
- 130703 In the sporadic server scheduling policy defined in POSIX.1-2024, the application can  
 130704 specify the maximum number of pending replenishment operations for a single sporadic  
 130705 server, by setting the value of the *sched\_ss\_max\_repl* scheduling parameter. This value must  
 130706 be between one and {SS\_REPL\_MAX}, which is a maximum limit imposed by the  
 130707 implementation. The limit {SS\_REPL\_MAX} must be greater than or equal to  
 130708 {\_POSIX\_SS\_REPL\_MAX}, which is defined to be four in POSIX.1-2024. The minimum  
 130709 limit of four was chosen so that an application can at least guarantee that four different  
 130710 aperiodic events can be processed during each interval of length equal to the  
 130711 replenishment period.
- 130712 *B.2.8.5 Clocks and Timers*
- Clocks
- 130714 POSIX.1-2024 and the ISO C standard both define functions for obtaining system time.  
 130715 Implicit behind these functions is a mechanism for measuring passage of time. This  
 130716 specification makes this mechanism explicit and calls it a clock. The CLOCK\_REALTIME  
 130717 clock required by POSIX.1-2024 is a higher resolution version of the clock that maintains  
 130718 POSIX.1 system time. This is a “system-wide” clock, in that it is visible to all processes  
 130719 and, were it possible for multiple processes to all read the clock at the same time, they  
 130720 would see the same value.
- 130721 An extensible interface was defined, with the ability for implementations to define  
 130722 additional clocks. This was done because of the observation that many realtime platforms  
 130723 support multiple clocks, and it was desired to fit this model within the standard interface.  
 130724 But implementation-defined clocks need not represent actual hardware devices, nor are  
 130725 they necessarily system-wide.
- Timers
- 130726 Two timer types are required for a system to support realtime applications:  
 130727

130728 1. One-shot

130729 A one-shot timer is a timer that is armed with an initial expiration time, either  
 130730 relative to the current time or at an absolute time (based on some timing base, such  
 130731 as time in seconds and nanoseconds since the Epoch). The timer expires once and  
 130732 then is disarmed. With the specified facilities, this is accomplished by setting the  
 130733 *it\_value* member of the *value* argument to the desired expiration time and the  
 130734 *it\_interval* member to zero.

130735 2. Periodic

130736 A periodic timer is a timer that is armed with an initial expiration time, again either  
 130737 relative or absolute, and a repetition interval. When the initial expiration occurs,  
 130738 the timer is reloaded with the repetition interval and continues counting. With the  
 130739 specified facilities, this is accomplished by setting the *it\_value* member of the *value*  
 130740 argument to the desired initial expiration time and the *it\_interval* member to the  
 130741 desired repetition interval.

130742 For both of these types of timers, the time of the initial timer expiration can be specified in  
 130743 two ways:

130744 1. Relative (to the current time)

130745 2. Absolute

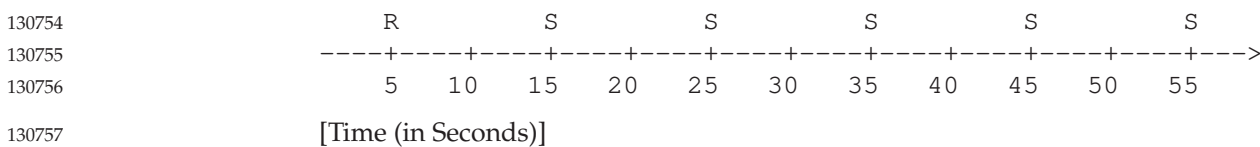
130746 • Examples of Using Realtime Timers

130747 In the diagrams below, *S* indicates a program schedule, *R* shows a schedule method  
 130748 request, and *E* suggests an internal operating system event.

130749 — Periodic Timer: Data Logging

130750 During an experiment, it might be necessary to log realtime data periodically to an  
 130751 internal buffer or to a mass storage device. With a periodic scheduling method, a  
 130752 logging module can be started automatically at fixed time intervals to log the data.

130753 Program schedule is requested every 10 seconds.

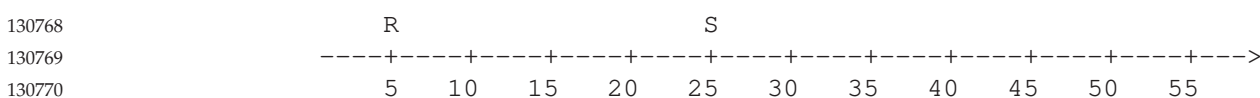


130758 To achieve this type of scheduling using the specified facilities, one would allocate a  
 130759 per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 130760 armed via a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an  
 130761 initial expiration value and a repetition interval of 10 seconds.

130762 — One-shot Timer (Relative Time): Device Initialization

130763 In an emission test environment, large sample bags are used to capture the exhaust  
 130764 from a vehicle. The exhaust is purged from these bags before each and every test.  
 130765 With a one-shot timer, a module could initiate the purge function and then suspend  
 130766 itself for a predetermined period of time while the sample bags are prepared.

130767 Program schedule requested 20 seconds after call is issued.



130771 [Time (in Seconds)]

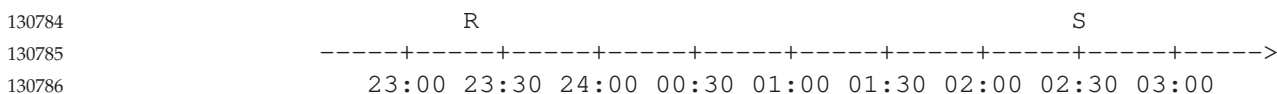
130772 To achieve this type of scheduling using the specified facilities, one would allocate a  
 130773 per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 130774 armed via a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an  
 130775 initial expiration value of 20 seconds and a repetition interval of zero.

130776 Note that if the program wishes merely to suspend itself for the specified interval, it  
 130777 could more easily use `nanosleep()`.

130778 — One-shot Timer (Absolute Time): Data Transmission

130779 The results from an experiment are often moved to a different system within a  
 130780 network for post-processing or archiving. With an absolute one-shot timer, a module  
 130781 that moves data from a test-cell computer to a host computer can be automatically  
 130782 scheduled on a daily basis.

130783 Program schedule requested for 2:30 a.m.



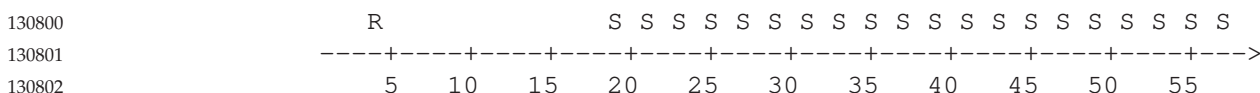
130787 [Time of Day]

130788 To achieve this type of scheduling using the specified facilities, a per-process timer  
 130789 would be allocated based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 130790 armed via a call to `timer_settime()` with the `TIMER_ABSTIME` flag set, and an initial  
 130791 expiration value equal to 2:30 a.m. of the next day.

130792 — Periodic Timer (Relative Time): Signal Stabilization

130793 Some measurement devices, such as emission analyzers, do not respond  
 130794 instantaneously to an introduced sample. With a periodic timer with a relative initial  
 130795 expiration time, a module that introduces a sample and records the average response  
 130796 could suspend itself for a predetermined period of time while the signal is stabilized  
 130797 and then sample at a fixed rate.

130798 Program schedule requested 15 seconds after call is issued and every 2 seconds  
 130799 thereafter.



130803 [Time (in Seconds)]

130804 To achieve this type of scheduling using the specified facilities, one would allocate a  
 130805 per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 130806 armed via a call to `timer_settime()` with `TIMER_ABSTIME` flag reset, and with an  
 130807 initial expiration value of 15 seconds and a repetition interval of 2 seconds.

130808 — Periodic Timer (Absolute Time): Work Shift-related Processing

130809 Resource utilization data is useful when time to perform experiments is being  
 130810 scheduled at a facility. With a periodic timer with an absolute initial expiration time,  
 130811 a module can be scheduled at the beginning of a work shift to gather resource  
 130812 utilization data throughout the shift. This data can be used to allocate resources  
 130813 effectively to minimize bottlenecks and delays and maximize facility throughput.

130814 Program schedule requested for 2:00 a.m. and every 15 minutes thereafter.

```

130815                R                                S S S S S S
130816          -----+-----+-----+-----+-----+-----+-----+----->
130817                23:00 23:30 24:00 00:30 01:00 01:30 02:00 02:30 03:00

```

130818 [Time of Day]

130819 To achieve this type of scheduling using the specified facilities, one would allocate a  
 130820 per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 130821 armed via a call to `timer_settime()` with `TIMER_ABSTIME` flag set, and with an initial  
 130822 expiration value equal to 2:00 a.m. and a repetition interval equal to 15 minutes.

130823 • Relationship of Timers to Clocks

130824 The relationship between clocks and timers armed with an absolute time is  
 130825 straightforward: a timer expiration signal is requested when the associated clock reaches  
 130826 or exceeds the specified time. The relationship between clocks and timers armed with a  
 130827 relative time (an interval) is less obvious, but not unintuitive. In this case, a timer  
 130828 expiration signal is requested when the specified interval, *as measured by the associated clock*,  
 130829 has passed. For the required `CLOCK_REALTIME` clock, this allows timer expiration  
 130830 signals to be requested at specified “wall clock” times (absolute), or when a specified  
 130831 interval of “realtime” has passed (relative). For an implementation-defined clock—say, a  
 130832 process virtual time clock—timer expirations could be requested when the process has  
 130833 used a specified total amount of virtual time (absolute), or when it has used a specified  
 130834 *additional* amount of virtual time (relative).

130835 The interfaces also allow flexibility in the implementation of the functions. For example, an  
 130836 implementation could convert all absolute times to intervals by subtracting the clock value  
 130837 at the time of the call from the requested expiration time and “counting down” at the  
 130838 supported resolution. Or it could convert all relative times to absolute expiration time by  
 130839 adding in the clock value at the time of the call and comparing the clock value to the  
 130840 expiration time at the supported resolution. Or it might even choose to maintain absolute  
 130841 times as absolute and compare them to the clock value at the supported resolution for  
 130842 absolute timers, and maintain relative times as intervals and count them down at the  
 130843 resolution supported for relative timers. The choice will be driven by efficiency  
 130844 considerations and the underlying hardware or software clock implementation.

130845 • Data Definitions for Clocks and Timers

130846 POSIX.1-2024 uses a time representation capable of supporting nanosecond resolution  
 130847 timers for the following reasons:

- 130848 — To enable POSIX.1-2024 to represent those computer systems already using  
 130849 nanosecond or submicrosecond resolution clocks.
- 130850 — To accommodate those per-process timers that might need nanoseconds to specify an  
 130851 absolute value of system-wide clocks, even though the resolution of the per-process  
 130852 timer may only be milliseconds, or *vice versa*.
- 130853 — Because the number of nanoseconds in a second can be represented in 32 bits.

130854 Time values are represented in the `timespec` structure. The `tv_sec` member is of type `time_t`  
 130855 so that this member is compatible with time values used by POSIX.1 functions and the  
 130856 ISO C standard. The `tv_nsec` member is a **signed long** in order to simplify and clarify code  
 130857 that decrements or finds differences of time values. Note that because 1 billion (number of  
 130858 nanoseconds per second) is less than half of the value representable by a signed 32-bit  
 130859 value, it is always possible to add two valid fractional seconds represented as integral  
 130860 nanoseconds without overflowing the signed 32-bit value.

130861 A maximum allowable resolution for the `CLOCK_REALTIME` clock of 20 ms (1/50



130862 seconds) was chosen to allow line frequency clocks in European countries to be  
130863 conforming. 60 Hz clocks in the US will also be conforming, as will finer granularity  
130864 clocks, although a Strictly Conforming Application cannot assume a granularity of less  
130865 than 20 ms (1/50 seconds).

130866 The minimum allowable maximum time allowed for the CLOCK\_REALTIME clock and  
130867 the function *nanosleep()*, and timers created with *clock\_id=CLOCK\_REALTIME*, is  
130868 determined by the fact that the *tv\_sec* member is of type **time\_t**.

130869 POSIX.1-2024 specifies that timer expirations must not be delivered early, and *nanosleep()*  
130870 must not return early due to quantization error. POSIX.1-2024 discusses the various  
130871 implementations of *alarm()* in the rationale and states that implementations that do not  
130872 allow alarm signals to occur early are the most appropriate, but refrained from mandating  
130873 this behavior. Because of the importance of predictability to realtime applications,  
130874 POSIX.1-2024 takes a stronger stance.

130875 The standard developers considered using a time representation that differs from  
130876 POSIX.1b in the second 32 bit of the 64-bit value. Whereas POSIX.1b defines this field as a  
130877 fractional second in nanoseconds, the other methodology defines this as a binary fraction  
130878 of one second, with the radix point assumed before the most significant bit.

130879 POSIX.1b is a software, source-level standard and most of the benefits of the alternate  
130880 representation are enjoyed by hardware implementations of clocks and algorithms. It was  
130881 felt that mandating this format for POSIX.1b clocks and timers would unnecessarily  
130882 burden the application developer with writing, possibly non-portable, multiple precision  
130883 arithmetic packages to perform conversion between binary fractions and integral units  
130884 such as nanoseconds, milliseconds, and so on.

#### 130885 **Rationale for the Monotonic Clock**

130886 For those applications that use time services to achieve realtime behavior, changing the value of  
130887 the clock on which these services rely may cause erroneous timing behavior. For these  
130888 applications, it is necessary to have a monotonic clock which cannot run backwards, and which  
130889 has a maximum clock jump that is required to be documented by the implementation.  
130890 Additionally, it is desirable (but not required by POSIX.1-2024) that the monotonic clock  
130891 increases its value uniformly. This clock should not be affected by changes to the system time;  
130892 for example, to synchronize the clock with an external source or to account for leap seconds.  
130893 Such changes would cause errors in the measurement of time intervals for those time services  
130894 that use the absolute value of the clock.

130895 One could argue that by defining the behavior of time services when the value of a clock is  
130896 changed, deterministic realtime behavior can be achieved. For example, one could specify that  
130897 relative time services should be unaffected by changes in the value of a clock. However, there are  
130898 time services that are based upon an absolute time, but that are essentially intended as relative  
130899 time services. For example, *pthread\_cond\_timedwait()* uses an absolute time to allow it to wake  
130900 up after the required interval despite spurious wakeups. Although sometimes the  
130901 *pthread\_cond\_timedwait()* timeouts are absolute in nature, there are many occasions in which they  
130902 are relative, and their absolute value is determined from the current time plus a relative time  
130903 interval. In this latter case, if the clock changes while the thread is waiting, the wait interval will  
130904 not be the expected length. If a *pthread\_cond\_timedwait()* function were created that would take a  
130905 relative time, it would not solve the problem because to retain the intended “deadline” a thread  
130906 would need to compensate for latency due to the spurious wakeup, and preemption between  
130907 wakeup and the next wait.

130908 The solution is to create a new monotonic clock, whose value does not change except for the  
130909 regular ticking of the clock, and use this clock for implementing the various relative timeouts

130910 that appear in the different POSIX interfaces, as well as allow `pthread_cond_timedwait()` to choose  
 130911 this new clock for its timeout. A new `clock_nanosleep()` function is created to allow an application  
 130912 to take advantage of this newly defined clock. Notice that the monotonic clock may be  
 130913 implemented using the same hardware clock as the system clock.

130914 Relative timeouts for `sigtimedwait()` and `aio_suspend()` have been redefined to use the monotonic  
 130915 clock, if present. The `alarm()` function has not been redefined, because the same effect but with  
 130916 better resolution can be achieved by creating a timer (for which the appropriate clock may be  
 130917 chosen).

130918 The `pthread_cond_timedwait()` function has been treated in a different way, compared to other  
 130919 functions with absolute timeouts, because it is used to wait for an event, and thus it may have a  
 130920 deadline, while the other timeouts are generally used as an error recovery mechanism, and for  
 130921 them the use of the monotonic clock is not so important. Since the desired timeout for the  
 130922 `pthread_cond_timedwait()` function may either be a relative interval or an absolute time of day  
 130923 deadline, a new initialization attribute has been created for condition variables to specify the  
 130924 clock that is used for measuring the timeout in a call to `pthread_cond_timedwait()`. In this way, if  
 130925 a relative timeout is desired, the monotonic clock will be used; if an absolute deadline is  
 130926 required instead, the `CLOCK_REALTIME` or another appropriate clock may be used. For  
 130927 condition variables, this capability is also available by passing `CLOCK_MONOTONIC` to the  
 130928 `pthread_cond_clockwait()` function. Similarly, `CLOCK_MONOTONIC` can be specified when  
 130929 calling `pthread_mutex_clocklock()`, `pthread_rwlock_clockrdlock()`, `pthread_rwlock_clockwrlock()`, and  
 130930 `sem_clockwait()`.

130931 It was later found necessary to add variants of almost all interfaces that accept absolute timeouts  
 130932 that allow the clock to be specified. This is because, despite the claim in the previous paragraph,  
 130933 it is not possible to safely use a `CLOCK_REALTIME` absolute timeout even to prevent errors  
 130934 when the system clock is warped by a potentially large amount. A “safety timeout” of a minute  
 130935 on a call to `pthread_mutex_timedlock()` could actually mean that the call would return  
 130936 `ETIMEDOUT` early without acquiring the lock if the system clock is warped forwards  
 130937 immediately prior to or during the call. On the other hand, a short timeout could end up being  
 130938 arbitrarily long if the system clock is warped backwards immediately prior to or during the call.  
 130939 These problems are solved by the new `clockwait` and `clocklock` variants of the existing `timedwait`  
 130940 and `timedlock` functions. These variants accept an extra `clockid_t` parameter to indicate the clock  
 130941 to be used for the wait. The clock ID is passed rather than using attributes as previously for  
 130942 `pthread_cond_timedwait()` in order to allow the ISO/IEC 14882:2011 standard (C++11) and later to  
 130943 be implemented correctly. C++ requires that the clock to use for the wait is not known until the  
 130944 time of the wait call, so it cannot be supplied during creation. The new functions are  
 130945 `pthread_cond_clockwait()`, `pthread_mutex_clocklock()`, `pthread_mutex_clockrdlock()`,  
 130946 `pthread_mutex_clockwrlock()`, and `sem_clockwait()`. It is expected that `mq_clockreceive()` and  
 130947 `mq_clocksend()` functions will be added in a future version of this standard.

130948 The `nanosleep()` function has not been modified with the introduction of the monotonic clock.  
 130949 Instead, a new `clock_nanosleep()` function has been created, in which the desired clock may be  
 130950 specified in the function call.

#### 130951 • History of Resolution Issues

130952 Due to the shift from relative to absolute timeouts in IEEE Std 1003.1d-1999, the  
 130953 amendments to the `sem_timedwait()`, `pthread_mutex_timedlock()`, `mq_timedreceive()`, and  
 130954 `mq_timedsend()` functions of that standard have been removed. Those amendments  
 130955 specified that `CLOCK_MONOTONIC` would be used for the (relative) timeouts if the  
 130956 (optional at the time) Monotonic Clock was supported.

130957 Having these functions continue to be tied solely to `CLOCK_MONOTONIC` would not  
 130958 work. Since the absolute value of a time value obtained from `CLOCK_MONOTONIC` is

130959 unspecified, under the absolute timeouts interface, applications would behave differently  
 130960 depending on whether the Monotonic Clock was supported or not (because the absolute  
 130961 value of the clock would have different meanings in either case).

130962 Two options were considered:

- 130963 1. Leave the current behavior unchanged, which specifies the CLOCK\_REALTIME  
 130964 clock for these (absolute) timeouts, to allow portability of applications between  
 130965 implementations supporting or not the Monotonic Clock.
- 130966 2. Modify these functions in the way that *pthread\_cond\_timedwait()* was modified to  
 130967 allow a choice of clock, so that an application could use CLOCK\_REALTIME when  
 130968 it is trying to achieve an absolute timeout and CLOCK\_MONOTONIC when it is  
 130969 trying to achieve a relative timeout.

130970 It was decided that the features of CLOCK\_MONOTONIC are not as critical to these  
 130971 functions as they are to *pthread\_cond\_timedwait()*. The *pthread\_cond\_timedwait()* function is  
 130972 given an absolute timeout; the timeout may represent a deadline for an event. When other  
 130973 functions are given relative timeouts, the timeouts are typically for error recovery  
 130974 purposes and need not be so precise.

130975 Therefore, it was decided that these functions should be tied to CLOCK\_REALTIME and  
 130976 not complicated by being given a choice of clock.

130977 Austin Group Defect 1346 is applied, requiring support for Monotonic Clock.

## 130978 Execution Time Monitoring

### 130979 • Introduction

130980 The main goals of the execution time monitoring facilities defined in this chapter are to  
 130981 measure the execution time of processes and threads and to allow an application to  
 130982 establish CPU time limits for these entities.

130983 The analysis phase of time-critical realtime systems often relies on the measurement of  
 130984 execution times of individual threads or processes to determine whether the timing  
 130985 requirements will be met. Also, performance analysis techniques for soft deadline realtime  
 130986 systems rely heavily on the determination of these execution times. The execution time  
 130987 monitoring functions provide application developers with the ability to measure these  
 130988 execution times online and open the possibility of dynamic execution-time analysis and  
 130989 system reconfiguration, if required.

130990 The second goal of allowing an application to establish execution time limits for individual  
 130991 processes or threads and detecting when they overrun allows program robustness to be  
 130992 increased by enabling online checking of the execution times.

130993 If errors are detected—possibly because of erroneous program constructs, the existence of  
 130994 errors in the analysis phase, or a burst of event arrivals—online detection and recovery is  
 130995 possible in a portable way. This feature can be extremely important for many time-critical  
 130996 applications. Other applications require trapping CPU-time errors as a normal way to exit  
 130997 an algorithm; for instance, some realtime artificial intelligence applications trigger a  
 130998 number of independent inference processes of varying accuracy and speed, limit how long  
 130999 they can run, and pick the best answer available when time runs out. In many periodic  
 131000 systems, overrun processes are simply restarted in the next resource period, after necessary  
 131001 end-of-period actions have been taken. This allows algorithms that are inherently data-  
 131002 dependent to be made predictable.

131003 The interface that appears in this chapter defines a new type of clock, the CPU-time clock,  
 131004 which measures execution time. Each process or thread can invoke the clock and timer

131005 functions defined in POSIX.1 to use them. Functions are also provided to access the CPU-  
 131006 time clock of other processes or threads to enable remote monitoring of these clocks.  
 131007 Monitoring of threads of other processes is not supported, since these threads are not  
 131008 visible from outside of their own process with the interfaces defined in POSIX.1.

131009 • Execution Time Monitoring Interface

131010 The clock and timer interface defined in POSIX.1 historically only defined one clock, which  
 131011 measures wall-clock time. The requirements for measuring execution time of processes and  
 131012 threads, and setting limits to their execution time by detecting when they overrun, can be  
 131013 accomplished with that interface if a new kind of clock is defined. These new clocks  
 131014 measure execution time, and one is associated with each process and with each thread. The  
 131015 clock functions currently defined in POSIX.1 can be used to read and set these CPU-time  
 131016 clocks, and timers can be created using these clocks as their timing base. These timers can  
 131017 then be used to send a signal when some specified execution time has been exceeded. The  
 131018 CPU-time clocks of each process or thread can be accessed by using the symbols  
 131019 `CLOCK_PROCESS_CPUTIME_ID` or `CLOCK_THREAD_CPUTIME_ID`.

131020 The clock and timer interface defined in POSIX.1 and extended with the new kind of CPU-  
 131021 time clock would only allow processes or threads to access their own CPU-time clocks.  
 131022 However, many realtime systems require the possibility of monitoring the execution time  
 131023 of processes or threads from independent monitoring entities. In order to allow  
 131024 applications to construct independent monitoring entities that do not require cooperation  
 131025 from or modification of the monitored entities, two functions have been added:  
 131026 `clock_getcpuclockid()`, for accessing CPU-time clocks of other processes, and  
 131027 `pthread_getcpuclockid()`, for accessing CPU-time clocks of other threads. These functions  
 131028 return the clock identifier associated with the process or thread specified in the call. These  
 131029 clock IDs can then be used in the rest of the clock function calls.

131030 The clocks accessed through these functions could also be used as a timing base for the  
 131031 creation of timers, thereby allowing independent monitoring entities to limit the CPU time  
 131032 consumed by other entities. However, this possibility would imply additional complexity  
 131033 and overhead because of the need to maintain a timer queue for each process or thread, to  
 131034 store the different expiration times associated with timers created by different processes or  
 131035 threads. The working group decided this additional overhead was not justified by  
 131036 application requirements. Therefore, creation of timers attached to the CPU-time clocks of  
 131037 other processes or threads has been specified as implementation-defined.

131038 • Overhead Considerations

131039 The measurement of execution time may introduce additional overhead in the thread  
 131040 scheduling, because of the need to keep track of the time consumed by each of these  
 131041 entities. In library-level implementations of threads, the efficiency of scheduling could be  
 131042 somehow compromised because of the need to make a kernel call, at each context switch,  
 131043 to read the process CPU-time clock. Consequently, a thread creation attribute called *cpu-  
 131044 clock-requirement* was defined, to allow threads to disconnect their respective CPU-time  
 131045 clocks. However, the Ballot Group considered that this attribute itself introduced some  
 131046 overhead, and that in current implementations it was not worth the effort. Therefore, the  
 131047 attribute was deleted, and thus thread CPU-time clocks are required for all threads if the  
 131048 Thread CPU-Time Clocks option is supported.

131049 • Accuracy of CPU-Time Clocks

131050 The mechanism used to measure the execution time of processes and threads is specified in  
 131051 POSIX.1-2024 as implementation-defined. The reason for this is that both the underlying  
 131052 hardware and the implementation architecture have a very strong influence on the  
 131053 accuracy achievable for measuring CPU time. For some implementations, the specification

131054 of strict accuracy requirements would represent very large overheads, or even the  
131055 impossibility of being implemented.

131056 Since the mechanism for measuring execution time is implementation-defined, realtime  
131057 applications will be able to take advantage of accurate implementations using a portable  
131058 interface. Of course, strictly conforming applications cannot rely on any particular degree  
131059 of accuracy, in the same way as they cannot rely on a very accurate measurement of wall  
131060 clock time. There will always exist applications whose accuracy or efficiency requirements  
131061 on the implementation are more rigid than the values defined in POSIX.1-2024 or any  
131062 other standard.

131063 In any case, there is a minimum set of characteristics that realtime applications would  
131064 expect from most implementations. One such characteristic is that the sum of all the  
131065 execution times of all the threads in a process equals the process execution time, when no  
131066 CPU-time clocks are disabled. This need not always be the case because implementations  
131067 may differ in how they account for time during context switches. Another characteristic is  
131068 that the sum of the execution times of all processes in a system equals the number of  
131069 processors, multiplied by the elapsed time, assuming that no processor is idle during that  
131070 elapsed time. However, in some implementations it might not be possible to relate CPU  
131071 time to elapsed time. For example, in a heterogeneous multi-processor system in which  
131072 each processor runs at a different speed, an implementation may choose to define each  
131073 “second” of CPU time to be a certain number of “cycles” that a CPU has executed.

- 131074 • Existing Practice

131075 Measuring and limiting the execution time of each concurrent activity are common  
131076 features of most industrial implementations of realtime systems. Almost all critical  
131077 realtime systems are currently built upon a cyclic executive. With this approach, a regular  
131078 timer interrupt kicks off the next sequence of computations. It also checks that the current  
131079 sequence has completed. If it has not, then some error recovery action can be undertaken  
131080 (or at least an overrun is avoided). Current software engineering principles and the  
131081 increasing complexity of software are driving application developers to implement these  
131082 systems on multi-threaded or multi-process operating systems. Therefore, if a POSIX  
131083 operating system is to be used for this type of application, then it must offer the same level  
131084 of protection.

131085 Execution time clocks are also common in most UNIX implementations, although these  
131086 clocks usually have requirements different from those of realtime applications. The  
131087 POSIX.1 *times()* function supports the measurement of the execution time of the calling  
131088 process, and its terminated child processes. This execution time is measured in clock ticks  
131089 and is supplied as two different values with the user and system execution times,  
131090 respectively. BSD supports the function *getrusage()*, which allows the calling process to get  
131091 information about the resources used by itself and/or all of its terminated child processes.  
131092 The resource usage includes user and system CPU time. Some UNIX systems have options  
131093 to specify high resolution (up to one microsecond) CPU-time clocks using the *times()* or  
131094 the *getrusage()* functions.

131095 The *times()* and *getrusage()* interfaces do not meet important realtime requirements, such  
131096 as the possibility of monitoring execution time from a different process or thread, or the  
131097 possibility of detecting an execution time overrun. The latter requirement is supported in  
131098 some UNIX implementations that are able to send a signal when the execution time of a  
131099 process has exceeded some specified value. For example, BSD defines the functions  
131100 *getitimer()* and *setitimer()*, which can operate either on a realtime clock (wall-clock), or on  
131101 virtual-time or profile-time clocks which measure CPU time in two different ways. These  
131102 functions do not support access to the execution time of other processes.

131103 At least one operating system supports per-process and per-thread execution time clocks,  
131104 and also supports limiting the execution time of a given process.

131105 Given all this existing practice, the working group considered that the POSIX.1 clocks and  
131106 timers interface was appropriate to meet most of the requirements that realtime  
131107 applications have for execution time clocks. Functions were added to get the CPU time  
131108 clock IDs, and to allow/disallow the thread CPU-time clocks (in order to preserve the  
131109 efficiency of some implementations of threads).

131110 • Clock Constants

131111 The definition of the manifest constants `CLOCK_PROCESS_CPUTIME_ID` and  
131112 `CLOCK_THREAD_CPUTIME_ID` allows processes or threads, respectively, to access their  
131113 own execution-time clocks. However, given a process or thread, access to its own  
131114 execution-time clock is also possible if the clock ID of this clock is obtained through a call  
131115 to `clock_getcpuclockid()` or `pthread_getcpuclockid()`. Therefore, these constants are not  
131116 necessary and could be deleted to make the interface simpler. Their existence saves one  
131117 system call in the first access to the CPU-time clock of each process or thread. The working  
131118 group considered this issue and decided to leave the constants in POSIX.1-2024 because  
131119 they are closer to the POSIX.1b use of clock identifiers.

131120 • Library Implementations of Threads

131121 In library implementations of threads, kernel entities and library threads can coexist. In  
131122 this case, if the CPU-time clocks are supported, most of the clock and timer functions will  
131123 need to have two implementations: one in the thread library, and one in the system calls  
131124 library. The main difference between these two implementations is that the thread library  
131125 implementation will have to deal with clocks and timers that reside in the thread space,  
131126 while the kernel implementation will operate on timers and clocks that reside in kernel  
131127 space. In the library implementation, if the clock ID refers to a clock that resides in the  
131128 kernel, a kernel call will have to be made. The correct version of the function can be chosen  
131129 by specifying the appropriate order for the libraries during the link process.

131130 • History of Resolution Issues: Deletion of the *enable* Attribute

131131 In early proposals, consideration was given to inclusion of an attribute called *enable* for  
131132 CPU-time clocks. This would allow implementations to avoid the overhead of measuring  
131133 execution time for those processes or threads for which this measurement was not  
131134 required. However, this is unnecessary since processes are already required to measure  
131135 execution time by the POSIX.1 `times()` function. Consequently, the *enable* attribute is not  
131136 present.

131137 **Rationale Relating to Timeouts**

131138 • Requirements for Timeouts

131139 Realtime systems which must operate reliably over extended periods without human  
131140 intervention are characteristic in embedded applications such as avionics, machine control,  
131141 and space exploration, as well as more mundane applications such as cable TV, security  
131142 systems, and plant automation. A multi-tasking paradigm, in which many independent  
131143 and/or cooperating software functions relinquish the processor(s) while waiting for a  
131144 specific stimulus, resource, condition, or operation completion, is very useful in producing  
131145 well engineered programs for such systems. For such systems to be robust and fault-  
131146 tolerant, expected occurrences that are unduly delayed or that never occur must be  
131147 detected so that appropriate recovery actions may be taken. This is difficult if there is no  
131148 way for a task to regain control of a processor once it has relinquished control (blocked)  
131149 awaiting an occurrence which, perhaps because of corrupted code, hardware malfunction,

131150 or latent software bugs, will not happen when expected. Therefore, the common practice  
 131151 in realtime operating systems is to provide a capability to time out such blocking services.  
 131152 Although there are several methods to achieve this already defined by POSIX, none are as  
 131153 reliable or efficient as initiating a timeout simultaneously with initiating a blocking service.  
 131154 This is especially critical in hard-realtime embedded systems because the processors  
 131155 typically have little time reserve, and allowed fault recovery times are measured in  
 131156 milliseconds rather than seconds.

131157 The working group largely agreed that such timeouts were necessary and ought to become  
 131158 part of POSIX.1-2024, particularly vendors of realtime operating systems whose customers  
 131159 had already expressed a strong need for timeouts. There was some resistance to inclusion  
 131160 of timeouts in POSIX.1-2024 because the desired effect, fault tolerance, could, in theory, be  
 131161 achieved using existing facilities and alternative software designs, but there was no  
 131162 compelling evidence that realtime system designers would embrace such designs at the  
 131163 sacrifice of performance and/or simplicity.

131164 • Which Services should be Timed Out?

131165 Originally, the working group considered the prospect of providing timeouts on all  
 131166 blocking services, including those currently existing in POSIX.1, POSIX.1b, and POSIX.1c,  
 131167 and future interfaces to be defined by other working groups, as sort of a general policy.  
 131168 This was rather quickly rejected because of the scope of such a change, and the fact that  
 131169 many of those services would not normally be used in a realtime context. More traditional  
 131170 timesharing solutions to timeout would suffice for most of the POSIX.1 interfaces, while  
 131171 others had asynchronous alternatives which, while more complex to utilize, would be  
 131172 adequate for some realtime and all non-realtime applications.

131173 The list of potential candidates for timeouts was narrowed to the following for further  
 131174 consideration:

- 131175 — POSIX.1b
  - 131176 — *sem\_wait()*
  - 131177 — *mq\_receive()*
  - 131178 — *mq\_send()*
  - 131179 — *lio\_listio()*
  - 131180 — *aio\_suspend()*
  - 131181 — *sigwait()* (timeout already implemented by *sigtimedwait()*)
- 131182 — POSIX.1c
  - 131183 — *pthread\_mutex\_lock()*
  - 131184 — *pthread\_join()*
  - 131185 — *pthread\_cond\_wait()*
  - 131186 (timeout already implemented by *pthread\_cond\_timedwait()*)
- 131187 — POSIX.1
  - 131188 — *read()*
  - 131189 — *write()*

131190 After further review by the working group, the *lio\_listio()*, *read()*, and *write()* functions (all  
 131191 forms of blocking synchronous I/O) were eliminated from the list because of the  
 131192 following:

- 131193 — Asynchronous alternatives exist
- 131194 — Timeouts can be implemented, albeit non-portably, in device drivers
- 131195 — A strong desire not to introduce modifications to POSIX.1 interfaces

131196 The working group ultimately rejected `pthread_join()` since both that interface and a timed  
 131197 variant of that interface are non-minimal and may be implemented as a function. See  
 131198 below for a library implementation of `pthread_join()`.

131199 Thus, there was a consensus among the working group members to add timeouts to 4 of  
 131200 the remaining 5 functions (the timeout for `aio_suspend()` was ultimately added directly to  
 131201 POSIX.1b, while the others were added by POSIX.1d). However, `pthread_mutex_lock()`  
 131202 remained contentious.

131203 Many feel that `pthread_mutex_lock()` falls into the same class as the other functions; that is,  
 131204 it is desirable to time out a mutex lock because a mutex may fail to be unlocked due to  
 131205 errant or corrupted code in a critical section (looping or branching outside of the unlock  
 131206 code), and therefore is equally in need of a reliable, simple, and efficient timeout. In fact,  
 131207 since mutexes are intended to guard small critical sections, most `pthread_mutex_lock()` calls  
 131208 would be expected to obtain the lock without blocking nor utilizing any kernel service,  
 131209 even in implementations of threads with global contention scope; the timeout alternative  
 131210 need only be considered after it is determined that the thread must block.

131211 Those opposed to timing out mutexes feel that the very simplicity of the mutex is  
 131212 compromised by adding a timeout semantic, and that to do so is senseless. They claim that  
 131213 if a timed mutex is really deemed useful by a particular application, then it can be  
 131214 constructed from the facilities already in POSIX.1b and POSIX.1c. The following two C-  
 131215 language library implementations of mutex locking with timeout represent the solutions  
 131216 offered (in both implementations, the timeout parameter is specified as absolute time, not  
 131217 relative time as in the proposed POSIX.1c interfaces).

#### 131218 • Spinlock Implementation

```

131219 #include <pthread.h>
131220 #include <time.h>
131221 #include <errno.h>

131222 int pthread_mutex_timedlock(pthread_mutex_t *mutex,
131223                             const struct timespec *timeout)
131224 {
131225     struct timespec timenow;

131226     while (pthread_mutex_trylock(mutex) == EBUSY)
131227     {
131228         clock_gettime(CLOCK_REALTIME, &timenow);
131229         if (timespec_cmp(&timenow, timeout) >= 0)
131230         {
131231             return ETIMEDOUT;
131232         }
131233         sched_yield();
131234     }
131235     return 0;
131236 }

```

131237 The Spinlock implementation is generally unsuitable for any application using priority-  
 131238 based thread scheduling policies such as SCHED\_FIFO or SCHED\_RR, since the mutex  
 131239 could currently be held by a thread of lower priority within the same allocation domain,



131240 but since the waiting thread never blocks, only threads of equal or higher priority will ever  
 131241 run, and the mutex cannot be unlocked. Setting priority inheritance or priority ceiling  
 131242 protocol on the mutex does not solve this problem, since the priority of a mutex owning  
 131243 thread is only boosted if higher priority threads are blocked waiting for the mutex; clearly  
 131244 not the case for this spinlock.

131245 • Condition Wait Implementation

```

131246 #include <pthread.h>
131247 #include <time.h>
131248 #include <errno.h>

131249 struct timed_mutex {
131250     int locked;
131251     pthread_mutex_t mutex;
131252     pthread_cond_t cond;
131253 };
131254 typedef struct timed_mutex timed_mutex_t;

131255 int timed_mutex_lock(timed_mutex_t *tm,
131256                    const struct timespec *timeout)
131257 {
131258     int timedout=FALSE;
131259     int error_status;

131260     pthread_mutex_lock(&tm->mutex);
131261     while (tm->locked && !timedout)
131262     {
131263         if ((error_status=pthread_cond_timedwait(&tm->cond,
131264         &tm->mutex, timeout))!=0)
131265         {
131266             if (error_status==ETIMEDOUT) timedout = TRUE;
131267         }
131268     }

131269     if(timedout)
131270     {
131271         pthread_mutex_unlock(&tm->mutex);
131272         return ETIMEDOUT;
131273     }
131274     else
131275     {
131276         tm->locked = TRUE;
131277         pthread_mutex_unlock(&tm->mutex);
131278         return 0;
131279     }
131280 }

131281 void timed_mutex_unlock(timed_mutex_t *tm)
131282 {
131283     pthread_mutex_lock(&tm->mutex); / for case assignment not atomic /
131284     tm->locked = FALSE;
131285     pthread_mutex_unlock(&tm->mutex);
131286     pthread_cond_signal(&tm->cond);
131287 }

```

131288 The Condition Wait implementation effectively substitutes the `pthread_cond_timedwait()`  
 131289 function (which is currently timed out) for the desired `pthread_mutex_timedlock()`. Since  
 131290 waits on condition variables currently do not include protocols which avoid priority  
 131291 inversion, this method is generally unsuitable for realtime applications because it does not  
 131292 provide the same priority inversion protection as the untimed `pthread_mutex_lock()`. Also,  
 131293 for any given implementations of the current mutex and condition variable primitives, this  
 131294 library implementation has a performance cost at least 2.5 times that of the untimed  
 131295 `pthread_mutex_lock()` even in the case where the timed mutex is readily locked without  
 131296 blocking (the interfaces required for this case are shown in bold). Even in uniprocessors or  
 131297 where assignment is atomic, at least an additional `pthread_cond_signal()` is required.  
 131298 `pthread_mutex_timedlock()` could be implemented at effectively no performance penalty in  
 131299 this case because the timeout parameters need only be considered after it is determined  
 131300 that the mutex cannot be locked immediately.

131301 Thus it has not yet been shown that the full semantics of mutex locking with timeout can  
 131302 be efficiently and reliably achieved using existing interfaces. Even if the existence of an  
 131303 acceptable library implementation were proven, it is difficult to justify why the interface  
 131304 itself should not be made portable, especially considering approval for the other four  
 131305 timeouts.

131306 • Rationale for Library Implementation of `pthread_timedjoin()`

131307 Library implementation of `pthread_timedjoin()`:

```
131308 /*
131309  * Construct a thread variety entirely from existing functions
131310  * with which a join can be done, allowing the join to time out.
131311  */
131312 #include <pthread.h>
131313 #include <time.h>
131314 struct timed_thread {
131315     pthread_t t;
131316     pthread_mutex_t m;
131317     int exiting;
131318     pthread_cond_t exit_c;
131319     void *(*start_routine)(void *arg);
131320     void *arg;
131321     void *status;
131322 };
131323 typedef struct timed_thread *timed_thread_t;
131324 static pthread_key_t timed_thread_key;
131325 static pthread_once_t timed_thread_once = PTHREAD_ONCE_INIT;
131326 static void timed_thread_init()
131327 {
131328     pthread_key_create(&timed_thread_key, NULL);
131329 }
131330 static void *timed_thread_start_routine(void *args)
131331 /*
131332  * Routine to establish thread-specific data value and run the actual
131333  * thread start routine which was supplied to timed_thread_create().
131334  */
131335 {
```

```

131336         timed_thread_t tt = (timed_thread_t) args;
131337         pthread_once(&timed_thread_once, timed_thread_init);
131338         pthread_setspecific(timed_thread_key, (void *)tt);
131339         timed_thread_exit((tt->start_routine)(tt->arg));
131340     }

131341     int timed_thread_create(timed_thread_t ttp, const pthread_attr_t *attr,
131342         void *(*start_routine)(void *), void *arg)

131343     /*
131344      * Allocate a thread which can be used with timed_thread_join().
131345      */
131346     {
131347         timed_thread_t tt;
131348         int result;

131349         tt = (timed_thread_t) malloc(sizeof(struct timed_thread));
131350         pthread_mutex_init(&tt->m, NULL);
131351         tt->exiting = FALSE;
131352         pthread_cond_init(&tt->exit_c, NULL);
131353         tt->start_routine = start_routine;
131354         tt->arg = arg;
131355         tt->status = NULL;

131356         if ((result = pthread_create(&tt->t, attr,
131357             timed_thread_start_routine, (void *)tt)) != 0) {
131358             free(tt);
131359             return result;
131360         }

131361         pthread_detach(tt->t);
131362         ttp = tt;
131363         return 0;
131364     }

131365     int timed_thread_join(timed_thread_t tt,
131366         struct timespec *timeout,
131367         void **status)
131368     {
131369         int result;

131370         pthread_mutex_lock(&tt->m);
131371         result = 0;
131372         /*
131373          * Wait until the thread announces that it is exiting,
131374          * or until timeout.
131375          */
131376         while (result == 0 && ! tt->exiting) {
131377             result = pthread_cond_timedwait(&tt->exit_c, &tt->m, timeout);
131378         }
131379         pthread_mutex_unlock(&tt->m);
131380         if (result == 0 && tt->exiting) {
131381             *status = tt->status;
131382             free((void *)tt);
131383             return result;
131384         }

```

```

131385         return result;
131386     }
131387 void timed_thread_exit(void *status)
131388 {
131389     timed_thread_t tt;
131390     void *specific;
131391
131392     if ((specific=pthread_getspecific(timed_thread_key)) == NULL) {
131393         /*
131394          * Handle cases which will not happen with correct usage.
131395          */
131396         pthread_exit( NULL);
131397     }
131398     tt = (timed_thread_t) specific;
131399     pthread_mutex_lock(&tt->m);
131400     /*
131401      * Tell a joiner that we are exiting.
131402      */
131403     tt->status = status;
131404     tt->exiting = TRUE;
131405     pthread_cond_signal(&tt->exit_c);
131406     pthread_mutex_unlock(&tt->m);
131407     /*
131408      * Call pthread exit() to call destructors and really
131409      * exit the thread.
131410      */
131411     pthread_exit(NULL);
131412 }

```

131412 The *pthread\_join()* C-language example shown above demonstrates that it is possible,  
131413 using existing pthread facilities, to construct a variety of thread which allows for joining  
131414 such a thread, but which allows the join operation to time out. It does this by using a  
131415 *pthread\_cond\_timedwait()* to wait for the thread to exit. A **timed\_thread\_t** descriptor  
131416 structure is used to pass parameters from the creating thread to the created thread, and  
131417 from the exiting thread to the joining thread. This implementation is roughly equivalent to  
131418 what a normal *pthread\_join()* implementation would do, with the single change being that  
131419 *pthread\_cond\_timedwait()* is used in place of a simple *pthread\_cond\_wait()*.

131420 Since it is possible to implement such a facility entirely from existing pthread interfaces,  
131421 and with roughly equal efficiency and complexity to an implementation which would be  
131422 provided directly by a pthreads implementation, it was the consensus of the working  
131423 group members that any *pthread\_timedjoin()* facility would be unnecessary, and should not  
131424 be provided.

#### 131425 • Form of the Timeout Interfaces

131426 The working group considered a number of alternative ways to add timeouts to blocking  
131427 services. At first, a system interface which would specify a one-shot or persistent timeout  
131428 to be applied to subsequent blocking services invoked by the calling process or thread was  
131429 considered because it allowed all blocking services to be timed out in a uniform manner  
131430 with a single additional interface; this was rather quickly rejected because it could easily  
131431 result in the wrong services being timed out.

131432 It was suggested that a timeout value might be specified as an attribute of the object  
131433 (semaphore, mutex, message queue, and so on), but there was no consensus on this, either

131434 on a case-by-case basis or for all timeouts.

131435 Looking at the two existing timeouts for blocking services indicates that the working  
131436 group members favor a separate interface for the timed version of a function. However,  
131437 `pthread_cond_timedwait()` utilizes an absolute timeout value while `sigtimedwait()` uses a  
131438 relative timeout value. The working group members agreed that relative timeout values  
131439 are appropriate where the timeout mechanism's primary use was to deal with an  
131440 unexpected or error situation, but they are inappropriate when the timeout must expire at  
131441 a particular time, or before a specific deadline. For the timeouts being introduced in  
131442 POSIX.1-2024, the working group considered allowing both relative and absolute timeouts  
131443 as is done with POSIX.1b timers, but ultimately favored the simpler absolute timeout form.

131444 An absolute time measure can be easily implemented on top of an interface that specifies  
131445 relative time, by reading the clock, calculating the difference between the current time and  
131446 the desired wakeup time, and issuing a relative timeout call. But there is a race condition  
131447 with this approach because the thread could be preempted after reading the clock, but  
131448 before making the timed-out call; in this case, the thread would be awakened later than it  
131449 should and, thus, if the wakeup time represented a deadline, it would miss it.

131450 There is also a race condition when trying to build a relative timeout on top of an interface  
131451 that specifies absolute timeouts. In this case, the clock would have to be read to calculate  
131452 the absolute wakeup time as the sum of the current time plus the relative timeout interval.  
131453 In this case, if the thread is preempted after reading the clock but before making the timed-  
131454 out call, the thread would be awakened earlier than desired.

131455 But the race condition with the absolute timeouts interface is not as bad as the one that  
131456 happens with the relative timeout interface, because there are simple workarounds. For the  
131457 absolute timeouts interface, if the timing requirement is a deadline, the deadline can still  
131458 be met because the thread woke up earlier than the deadline. If the timeout is just used as  
131459 an error recovery mechanism, the precision of timing is not really important. If the timing  
131460 requirement is that between actions A and B a minimum interval of time must elapse, the  
131461 absolute timeout interface can be safely used by reading the clock after action A has been  
131462 started. It could be argued that, since the call with the absolute timeout is atomic from the  
131463 application point of view, it is not possible to read the clock after action A, if this action is  
131464 part of the timed-out call. But looking at the nature of the calls for which timeouts are  
131465 specified (locking a mutex, waiting for a semaphore, waiting for a message, or waiting  
131466 until there is space in a message queue), the timeouts that an application would build on  
131467 these actions would not be triggered by these actions themselves, but by some other  
131468 external action. For example, if waiting for a message to arrive to a message queue, and  
131469 waiting for at least 20 milliseconds, this time interval would start to be counted from some  
131470 event that would trigger both the action that produces the message, as well as the action  
131471 that waits for the message to arrive, and not by the wait-for-message operation itself. In  
131472 this case, the workaround proposed above could be used.

131473 For these reasons, the absolute timeout is preferred over the relative timeout interface.

131474 **B.2.9 Threads**

131475 Threads will normally be more expensive than subroutines (or functions, routines, and so on) if  
131476 specialized hardware support is not provided. Nevertheless, threads should be sufficiently  
131477 efficient to encourage their use as a medium to fine-grained structuring mechanism for  
131478 parallelism in an application. Structuring an application using threads then allows it to take  
131479 immediate advantage of any underlying parallelism available in the host environment. This  
131480 means implementors are encouraged to optimize for fast execution at the possible expense of  
131481 efficient utilization of storage. For example, a common thread creation technique is to cache  
131482 appropriate thread data structures. That is, rather than releasing system resources, the  
131483 implementation retains these resources and reuses them when the program next asks to create a  
131484 new thread. If this reuse of thread resources is to be possible, there has to be very little unique  
131485 state associated with each thread, because any such state has to be reset when the thread is  
131486 reused.

131487 **Thread Creation Attributes**

131488 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
131489 support probable future standardization in these areas without requiring that the interface itself  
131490 be changed.

131491 Attributes objects provide clean isolation of the configurable aspects of threads. For example,  
131492 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When  
131493 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects  
131494 can help by allowing the changes to be isolated in a single place, rather than being spread across  
131495 every instance of thread creation.

131496 Attributes objects can be used to set up *classes* of threads with similar attributes; for example,  
131497 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
131498 can be defined in a single place and then referenced wherever threads need to be created.  
131499 Changes to “class” decisions become straightforward, and detailed analysis of each  
131500 *pthread\_create()* call is not required.

131501 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
131502 been specified as structures, adding new attributes would force recompilation of all multi-  
131503 threaded programs when the attributes objects are extended; this might not be possible if  
131504 different program components were supplied by different vendors.

131505 Additionally, opaque attributes objects present opportunities for improving performance.  
131506 Argument validity can be checked once when attributes are set, rather than each time a thread is  
131507 created. Implementations will often need to cache kernel objects that are expensive to create.  
131508 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
131509 invalid due to attribute changes.

131510 Because assignment is not necessarily defined on a given opaque type, implementation-defined  
131511 default values cannot be defined in a portable way. The solution to this problem is to allow  
131512 attribute objects to be initialized dynamically by attributes object initialization functions, so that  
131513 default values can be supplied automatically by the implementation.

131514 The following proposal was provided as a suggested alternative to the supplied attributes:

- 131515 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
131516 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
131517 parameter containing the flags should be an opaque type for extensibility. If no flags are  
131518 set in the parameter, then the objects are created with default characteristics. An  
131519 implementation may specify implementation-defined flag values and associated  
131520 behavior.

131521 2. If further specialization of mutexes and condition variables is necessary, implementations  
 131522 may specify additional procedures that operate on the `pthread_mutex_t` and  
 131523 `pthread_cond_t` objects (instead of on attributes objects).

131524 The difficulties with this solution are:

131525 1. A bitmask is not opaque if bits have to be set into bit-vector attributes objects using  
 131526 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an `int`,  
 131527 application programmers need to know the location of each bit. If bits are set or read by  
 131528 encapsulation (that is, `get*()` or `set*()` functions), then the bitmask is merely an  
 131529 implementation of attributes objects as currently defined and should not be exposed to  
 131530 the programmer.

131531 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
 131532 policy may be placed in 3 bits or 4 bits, but priority requires 5 bits or more, thereby taking  
 131533 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,  
 131534 the bitmask can only reasonably control whether particular attributes are set or not, and it  
 131535 cannot serve as the repository of the value itself. The value needs to be specified as a  
 131536 function parameter (which is non-extensible), or by setting a structure field (which is non-  
 131537 opaque), or by `get*()` and `set*()` functions (making the bitmask a redundant addition to  
 131538 the attributes objects).

131539 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
 131540 machine-dependent. Some implementations may not be able to change the size of the stack, for  
 131541 example, and others may not need to because stack pages may be discontinuous and can be  
 131542 allocated and released on demand.

131543 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
 131544 to the attribute mechanism or to any attributes object defined in POSIX.1-2024 have to be done  
 131545 with care so as not to affect binary-compatibility.

131546 Attribute objects, even if allocated by means of dynamic allocation functions such as `malloc()`,  
 131547 may have their size fixed at compile time. This means, for example, a `pthread_create()` in an  
 131548 implementation with extensions to the `pthread_attr_t` cannot look beyond the area that the  
 131549 binary application assumes is valid. This suggests that implementations should maintain a size  
 131550 field in the attributes object, as well as possibly version information, if extensions in different  
 131551 directions (possibly by different vendors) are to be accommodated.

## 131552 Thread Implementation Models

131553 There are various thread implementation models. At one end of the spectrum is the “library-  
 131554 thread model”. In such a model, the threads of a process are not visible to the operating system  
 131555 kernel, and the threads are not kernel-scheduled entities. The process is the only kernel-  
 131556 scheduled entity. The process is scheduled onto the processor by the kernel according to the  
 131557 scheduling attributes of the process. The threads are scheduled onto the single kernel-scheduled  
 131558 entity (the process) by the runtime library according to the scheduling attributes of the threads.  
 131559 A problem with this model is that it constrains concurrency. Since there is only one kernel-  
 131560 scheduled entity (namely, the process), only one thread per process can execute at a time. If the  
 131561 thread that is executing blocks on I/O, then the whole process blocks.

131562 At the other end of the spectrum is the “kernel-thread model”. In this model, all threads are  
 131563 visible to the operating system kernel. Thus, all threads are kernel-scheduled entities, and all  
 131564 threads can concurrently execute. The threads are scheduled onto processors by the kernel  
 131565 according to the scheduling attributes of the threads. The drawback to this model is that the  
 131566 creation and management of the threads entails operating system calls, as opposed to subroutine  
 131567 calls, which makes kernel threads heavier weight than library threads.

131568 Hybrids of these two models are common. A hybrid model offers the speed of library threads  
 131569 and the concurrency of kernel threads. In hybrid models, a process has some (relatively small)  
 131570 number of kernel scheduled entities associated with it. It also has a potentially much larger  
 131571 number of library threads associated with it. Some library threads may be bound to kernel-  
 131572 scheduled entities, while the other library threads are multiplexed onto the remaining kernel-  
 131573 scheduled entities. There are two levels of thread scheduling:

- 131574 1. The runtime library manages the scheduling of (unbound) library threads onto kernel-  
 131575 scheduled entities.
- 131576 2. The kernel manages the scheduling of kernel-scheduled entities onto processors.

131577 For this reason, a hybrid model is referred to as a two-level threads scheduling model. In this  
 131578 model, the process can have multiple concurrently executing threads; specifically, it can have as  
 131579 many concurrently executing threads as it has kernel-scheduled entities.

### 131580 Thread-Specific Data

131581 Many applications require that a certain amount of context be maintained on a per-thread basis  
 131582 across procedure calls. A common example is a multi-threaded library routine that allocates  
 131583 resources from a common pool and maintains an active resource list for each thread. The thread-  
 131584 specific data interface provided to meet these needs may be viewed as a two-dimensional array  
 131585 of values with keys serving as the row index and thread IDs as the column index (although the  
 131586 implementation need not work this way).

#### 131587 • Models

131588 Three possible thread-specific data models were considered:

#### 131589 1. No Explicit Support

131590 A standard thread-specific data interface is not strictly necessary to support  
 131591 applications that require per-thread context. One could, for example, provide a hash  
 131592 function that converted a **pthread\_t** into an integer value that could then be used to  
 131593 index into a global array of per-thread data pointers. This hash function, in  
 131594 conjunction with *pthread\_self()*, would be all the interface required to support a  
 131595 mechanism of this sort. Unfortunately, this technique is cumbersome. It can lead to  
 131596 duplicated code as each set of cooperating modules implements their own per-  
 131597 thread data management schemes. This technique would also require that **pthread\_t**  
 131598 not be an opaque type.

#### 131599 2. Single (**void \***) Pointer

131600 Another technique would be to provide a single word of per-thread storage and a  
 131601 pair of functions to fetch and store the value of this word. The word could then hold  
 131602 a pointer to a block of per-thread memory. The allocation, partitioning, and general  
 131603 use of this memory would be entirely up to the application. Although this method  
 131604 is not as problematic as technique 1, it suffers from interoperability problems. For  
 131605 example, all modules using the per-thread pointer would have to agree on a  
 131606 common usage protocol.

#### 131607 3. Key/Value Mechanism

131608 This method associates an opaque key (for example, stored in a variable of type  
 131609 **pthread\_key\_t**) with each per-thread datum. These keys play the role of identifiers  
 131610 for per-thread data. This technique is the most generic and avoids the problems  
 131611 noted above, albeit at the cost of some complexity.

131612 The primary advantage of the third model is its information hiding properties. Modules



131613 using this model are free to create and use their own key(s) independent of all other such  
 131614 usage, whereas the other models require that all modules that use thread-specific context  
 131615 explicitly cooperate with all other such modules. The data-independence provided by the  
 131616 third model is worth the additional interface. Therefore, the third model was chosen.

131617 • Requirements

131618 It is important that it be possible to implement the thread-specific data interface without  
 131619 the use of thread private memory. To do otherwise would increase the weight of each  
 131620 thread, thereby limiting the range of applications for which the threads interfaces provided  
 131621 by POSIX.1-2024 is appropriate.

131622 The values that one binds to the key via `pthread_setspecific()` may, in fact, be pointers to  
 131623 shared storage locations available to all threads. It is only the key/value bindings that are  
 131624 maintained on a per-thread basis, and these can be kept in any portion of the address space  
 131625 that is reserved for use by the calling thread (for example, on the stack). Thus, no per-  
 131626 thread MMU state is required to implement the interface. On the other hand, there is  
 131627 nothing in the interface specification to preclude the use of a per-thread MMU state if it is  
 131628 available (for example, the key values returned by `pthread_key_create()` could be thread  
 131629 private memory addresses).

131630 • Standardization Issues

131631 Thread-specific data is a requirement for a usable thread interface. The binding described  
 131632 in this section provides a portable thread-specific data mechanism for languages that do  
 131633 not directly support a thread-specific storage class. A binding to POSIX.1-2024 for a  
 131634 language that does include such a storage class need not provide this specific interface.

131635 If a language were to include the notion of thread-specific storage, it would be desirable  
 131636 (but *not* required) to provide an implementation of the pthreads thread-specific data  
 131637 interface based on the language feature. For example, assume that a compiler for a C-like  
 131638 language supports a *private* storage class that provides thread-specific storage. Something  
 131639 similar to the following macros might be used to effect a compatible implementation:

```
131640 #define pthread_key_t                private void *
131641 #define pthread_key_create(key)      /* no-op */
131642 #define pthread_setspecific(key,value) (key)=(value)
131643 #define pthread_getspecific(key)     (key)
```

131644 **Note:** For the sake of clarity, this example ignores destructor functions. A correct  
 131645 implementation would have to support them.

131646 **Barriers**

131647 • Background

131648 Barriers are typically used in parallel DO/FOR loops to ensure that all threads have  
 131649 reached a particular stage in a parallel computation before allowing any to proceed to the  
 131650 next stage. Highly efficient implementation is possible on machines which support a  
 131651 “Fetch and Add” operation as described in the referenced Almasi and Gottlieb (1989).

131652 The use of return value `PTHREAD_BARRIER_SERIAL_THREAD` is shown in the  
 131653 following example:

```
131654 if ( (status=pthread_barrier_wait(&barrier)) ==
131655     PTHREAD_BARRIER_SERIAL_THREAD) {
131656     ...serial section
131657 }
131658 else if (status != 0) {
```

```

131659         ...error processing
131660     }
131661     status=pthread_barrier_wait (&barrier);
131662     ...

```

131663 This behavior allows a serial section of code to be executed by one thread as soon as all  
 131664 threads reach the first barrier. The second barrier prevents the other threads from  
 131665 proceeding until the serial section being executed by the one thread has completed.

131666 Although barriers can be implemented with mutexes and condition variables, the  
 131667 referenced Almasi and Gottlieb (1989) provides ample illustration that such  
 131668 implementations are significantly less efficient than is possible. While the relative  
 131669 efficiency of barriers may well vary by implementation, it is important that they be  
 131670 recognized in the POSIX.1-2024 to facilitate applications portability while providing the  
 131671 necessary freedom to implementors.

131672 • Lack of Timeout Feature

131673 Alternate versions of most blocking routines have been provided to support watchdog  
 131674 timeouts. No alternate interface of this sort has been provided for barrier waits for the  
 131675 following reasons:

- 131676 • Multiple threads may use different timeout values, some of which may be indefinite.  
 131677 It is not clear which threads should break through the barrier with a timeout error if  
 131678 and when these timeouts expire.
- 131679 • The barrier may become unusable once a thread breaks out of a *pthread\_barrier\_wait()*  
 131680 with a timeout error. There is, in general, no way to guarantee the consistency of a  
 131681 barrier's internal data structures once a thread has timed out of a  
 131682 *pthread\_barrier\_wait()*. Even the inclusion of a special barrier reinitialization function  
 131683 would not help much since it is not clear how this function would affect the behavior  
 131684 of threads that reach the barrier between the original timeout and the call to the  
 131685 reinitialization function.

131686 **Spin Locks**

131687 • Background

131688 Spin locks represent an extremely low-level synchronization mechanism suitable primarily  
 131689 for use on shared memory multi-processors. It is typically an atomically modified Boolean  
 131690 value that is set to one when the lock is held and to zero when the lock is freed.

131691 When a caller requests a spin lock that is already held, it typically spins in a loop testing  
 131692 whether the lock has become available. Such spinning wastes processor cycles so the lock  
 131693 should only be held for short durations and not across sleep/block operations. Callers  
 131694 should unlock spin locks before calling sleep operations.

131695 Spin locks are available on a variety of systems. The functions included in POSIX.1-2024  
 131696 are an attempt to standardize that existing practice.

131697 • Lack of Timeout Feature

131698 Alternate versions of most blocking routines have been provided to support watchdog  
 131699 timeouts. No alternate interface of this sort has been provided for spin locks for the  
 131700 following reasons:

- 131701 • It is impossible to determine appropriate timeout intervals for spin locks in a  
 131702 portable manner. The amount of time one can expect to spend spin-waiting is  
 131703 inversely proportional to the degree of parallelism provided by the system.

131704 It can vary from a few cycles when each competing thread is running on its own  
 131705 processor, to an indefinite amount of time when all threads are multiplexed on a  
 131706 single processor (which is why spin locking is not advisable on uniprocessors).

131707 • When used properly, the amount of time the calling thread spends waiting on a spin  
 131708 lock should be considerably less than the time required to set up a corresponding  
 131709 watchdog timer. Since the primary purpose of spin locks is to provide a low-  
 131710 overhead synchronization mechanism for multi-processors, the overhead of a  
 131711 timeout mechanism was deemed unacceptable.

131712 It was also suggested that an additional *count* argument be provided (on the  
 131713 *pthread\_spin\_lock()* call) in lieu of a true timeout so that a spin lock call could fail gracefully  
 131714 if it was unable to apply the lock after *count* attempts. This idea was rejected because it is  
 131715 not existing practice. Furthermore, the same effect can be obtained with  
 131716 *pthread\_spin\_trylock()*, as illustrated below:

```
131717 int n = MAX_SPIN;
131718 while ( --n >= 0 )
131719 {
131720     if ( !pthread_spin_try_lock(...) )
131721         break;
131722 }
131723 if ( n >= 0 )
131724 {
131725     /* Successfully acquired the lock */
131726 }
131727 else
131728 {
131729     /* Unable to acquire the lock */
131730 }
```

131731 • *process-shared* Attribute

131732 The initialization functions associated with most POSIX synchronization objects (for  
 131733 example, mutexes, barriers, and read-write locks) take an attributes object with a *process-  
 131734 shared* attribute that specifies whether or not the object is to be shared across processes. In  
 131735 the draft corresponding to the first balloting round, two separate initialization functions  
 131736 are provided for spin locks, however: one for spin locks that were to be shared across  
 131737 processes (*spin\_init()*), and one for locks that were only used by multiple threads within a  
 131738 single process (*pthread\_spin\_init()*). This was done so as to keep the overhead associated  
 131739 with spin waiting to an absolute minimum. However, the balloting group requested that,  
 131740 since the overhead associated to a bit check was small, spin locks should be consistent with  
 131741 the rest of the synchronization primitives, and thus the *process-shared* attribute was  
 131742 introduced for spin locks.

131743 • Spin Locks *versus* Mutexes

131744 It has been suggested that mutexes are an adequate synchronization mechanism and spin  
 131745 locks are not necessary. Locking mechanisms typically must trade off the processor  
 131746 resources consumed while setting up to block the thread and the processor resources  
 131747 consumed by the thread while it is blocked. Spin locks require very little resources to set  
 131748 up the blocking of a thread. Existing practice is to simply loop, repeating the atomic  
 131749 locking operation until the lock is available. While the resources consumed to set up  
 131750 blocking of the thread are low, the thread continues to consume processor resources while  
 131751 it is waiting.

131752 On the other hand, mutexes may be implemented such that the processor resources  
131753 consumed to block the thread are large relative to a spin lock. After detecting that the  
131754 mutex lock is not available, the thread must alter its scheduling state, add itself to a set of  
131755 waiting threads, and, when the lock becomes available again, undo all of this before taking  
131756 over ownership of the mutex. However, while a thread is blocked by a mutex, no processor  
131757 resources are consumed.

131758 Therefore, spin locks and mutexes may be implemented to have different characteristics.  
131759 Spin locks may have lower overall overhead for very short-term blocking, and mutexes  
131760 may have lower overall overhead when a thread will be blocked for longer periods of time.  
131761 The presence of both interfaces allows implementations with these two different  
131762 characteristics, both of which may be useful to a particular application.

131763 It has also been suggested that applications can build their own spin locks from the  
131764 `pthread_mutex_trylock()` function:

```
131765 while (pthread_mutex_trylock(&mutex));
```

131766 The apparent simplicity of this construct is somewhat deceiving, however. While the actual  
131767 wait is quite efficient, various guarantees on the integrity of mutex objects (for example,  
131768 priority inheritance rules) may add overhead to the successful path of the trylock  
131769 operation that is not required of spin locks. One could, of course, add an attribute to the  
131770 mutex to bypass such overhead, but the very act of finding and testing this attribute  
131771 represents more overhead than is found in the typical spin lock.

131772 The need to hold spin lock overhead to an absolute minimum also makes it impossible to  
131773 provide guarantees against starvation similar to those provided for mutexes or read-write  
131774 locks. The overhead required to implement such guarantees (for example, disabling  
131775 preemption before spinning) may well exceed the overhead of the spin wait itself by many  
131776 orders of magnitude. If a "safe" spin wait seems desirable, it can always be provided  
131777 (albeit at some performance cost) via appropriate mutex attributes.

### 131778 **Robust Mutexes**

131779 Robust mutexes are intended to protect applications that use mutexes to protect data shared  
131780 between different processes. If a process is terminated by a signal while a thread is holding a  
131781 mutex, there is no chance for the process to clean up after it. Waiters for the locked mutex might  
131782 wait indefinitely.

131783 With robust mutexes the problem can be solved: whenever a fatal signal terminates a process,  
131784 current or future waiters of the mutex are notified about this fact. The locking function provides  
131785 notification of this condition through the error condition [EOWNERDEAD]. A thread then has  
131786 the chance to clean up the state protected by the mutex and mark the state as consistent again by  
131787 a call to `pthread_mutex_consistent()`.

131788 Pre-existing implementations have used the semantics of robust mutexes for a variety of  
131789 situations, some of them not defined in the standard. Where a normally terminated process (i.e.,  
131790 when one thread calls `exit()`) causes notification of other waiters of robust mutexes if the mutex  
131791 is locked by any thread in the process. This behavior is defined in the standard and makes sense  
131792 because no thread other than the thread calling `exit()` has the chance to clean up its data.

131793 If a thread is terminated by cancellation or if it calls `pthread_exit()`, the situation is different. In  
131794 both these situations the thread has the chance to clean up after itself by registering appropriate  
131795 cleanup handlers. There is no real reason to demand that other waiters for a robust mutex the  
131796 terminating thread owns are notified. The committee felt that this is actively encouraging bad  
131797 practice because programmers are tempted to rely on the robust mutex semantics instead of  
131798 correctly cleaning up after themselves.

131799 Therefore, the standard does not require notification of other waiters at the time a thread is  
131800 terminated while the process continues to run. The mutex is still recognized as being locked by  
131801 the process (with the thread gone it makes no sense to refer to the thread owning the mutex).  
131802 Therefore, a terminating process will cause notifications about the dead owner to be sent to all  
131803 waiters. This delay in the notification is not required, but programmers cannot rely on prompt  
131804 notification after a thread is terminated.

131805 For the same reason is it not required that an implementation supports robust mutexes that are  
131806 not shared between processes. If a robust mutex is used only within one process, all the cleanup  
131807 can be performed by the threads themselves by registering appropriate cleanup handlers. Fatal  
131808 signals are of no importance in this case because after the signal is delivered there is no thread  
131809 remaining to use the mutex.

131810 Some implementations might choose to support intra-process robust mutexes and they might  
131811 also send notification of a dead owner right after the previous owner died. But applications  
131812 must not rely on this. Applications should only use robust mutexes for the purpose of handling  
131813 fatal signals in situations where inter-process mutexes are in use.

#### 131814 **Supported Threads Functions**

131815 On POSIX-conforming systems, the following symbolic constants are always conforming:

131816        \_POSIX\_READER\_WRITER\_LOCKS  
131817        \_POSIX\_THREADS

131818 Therefore, the following threads functions are always supported:

131819	<i>pthread_atfork()</i>	<i>pthread_kill()</i>
131820	<i>pthread_attr_destroy()</i>	<i>pthread_mutex_destroy()</i>
131821	<i>pthread_attr_getdetachstate()</i>	<i>pthread_mutex_init()</i>
131822	<i>pthread_attr_getguardsize()</i>	<i>pthread_mutex_lock()</i>
131823	<i>pthread_attr_getschedparam()</i>	<i>pthread_mutex_trylock()</i>
131824	<i>pthread_attr_init()</i>	<i>pthread_mutex_unlock()</i>
131825	<i>pthread_attr_setdetachstate()</i>	<i>pthread_mutexattr_destroy()</i>
131826	<i>pthread_attr_setguardsize()</i>	<i>pthread_mutexattr_getpshared()</i>
131827	<i>pthread_attr_setschedparam()</i>	<i>pthread_mutexattr_gettype()</i>
131828	<i>pthread_cancel()</i>	<i>pthread_mutexattr_init()</i>
131829	<i>pthread_cleanup_pop()</i>	<i>pthread_mutexattr_setpshared()</i>
131830	<i>pthread_cleanup_push()</i>	<i>pthread_mutexattr_settype()</i>
131831	<i>pthread_cond_broadcast()</i>	<i>pthread_once()</i>
131832	<i>pthread_cond_clockwait()</i>	<i>pthread_rwlock_destroy()</i>
131833	<i>pthread_cond_destroy()</i>	<i>pthread_rwlock_init()</i>
131834	<i>pthread_cond_init()</i>	<i>pthread_rwlock_rdlock()</i>
131835	<i>pthread_cond_signal()</i>	<i>pthread_rwlock_tryrdlock()</i>
131836	<i>pthread_cond_timedwait()</i>	<i>pthread_rwlock_trywrlock()</i>
131837	<i>pthread_cond_wait()</i>	<i>pthread_rwlock_unlock()</i>
131838	<i>pthread_condattr_destroy()</i>	<i>pthread_rwlock_wrlock()</i>
131839	<i>pthread_condattr_getpshared()</i>	<i>pthread_rwlockattr_destroy()</i>
131840	<i>pthread_condattr_init()</i>	<i>pthread_rwlockattr_getpshared()</i>
131841	<i>pthread_condattr_setpshared()</i>	<i>pthread_rwlockattr_init()</i>
131842	<i>pthread_create()</i>	<i>pthread_rwlockattr_setpshared()</i>
131843	<i>pthread_detach()</i>	<i>pthread_self()</i>
131844	<i>pthread_equal()</i>	<i>pthread_setcancelstate()</i>
131845	<i>pthread_exit()</i>	<i>pthread_setcanceltype()</i>
131846	<i>pthread_getspecific()</i>	<i>pthread_setspecific()</i>
131847	<i>pthread_join()</i>	<i>pthread_sigmask()</i>
131848	<i>pthread_key_create()</i>	<i>pthread_testcancel()</i>
131849	<i>pthread_key_delete()</i>	<i>sigwait()</i>

131850 On POSIX-conforming systems, the symbolic constant `_POSIX_THREAD_SAFE_FUNCTIONS` is  
 131851 always defined. Therefore, the following functions are always supported:

131852	<code>flockfile()</code>	<code>getpwuid_r()</code>
131853	<code>ftrylockfile()</code>	<code>gmtime_r()</code>
131854	<code>funlockfile()</code>	<code>localtime_r()</code>
131855	<code>getc_unlocked()</code>	<code>putc_unlocked()</code>
131856	<code>getchar_unlocked()</code>	<code>putchar_unlocked()</code>
131857	<code>getgrgid_r()</code>	<code>readdir_r()</code>
131858	<code>getgrnam_r()</code>	<code>strerror_r()</code>
131859	<code>getpwnam_r()</code>	<code>strtok_r()</code>

## 131860 Threads Extensions

131861 The following extensions to the IEEE P1003.1c draft standard are now supported in  
 131862 POSIX.1-2024 as part of the alignment with the Single UNIX Specification:

- 131863 • Extended mutex attribute types
- 131864 • Read-write locks and attributes (also introduced by the IEEE Std 1003.1j-2000 amendment)
- 131865 • Thread concurrency level
- 131866 • Thread stack guard size
- 131867 • Parallel I/O
- 131868 • Robust mutexes

131869 These extensions carefully follow the threads programming model specified in POSIX.1c. As  
 131870 with POSIX.1c, all the new functions return zero if successful; otherwise, an error number is  
 131871 returned to indicate the error.

131872 The concept of attribute objects was introduced in POSIX.1c to allow implementations to extend  
 131873 POSIX.1-2024 without changing the existing interfaces. Attribute objects were defined for  
 131874 threads, mutexes, and condition variables. Attributes objects are defined as implementation-  
 131875 defined opaque types to aid extensibility, and functions are defined to allow attributes to be set  
 131876 or retrieved. This model has been followed when adding the new type attribute of  
 131877 **`pthread_mutexattr_t`** or the new read-write lock attributes object **`pthread_rwlockattr_t`**.

- 131878 • Extended Mutex Attributes

131879 POSIX.1c defines a mutex attributes object as an implementation-defined opaque object of  
 131880 type **`pthread_mutexattr_t`**, and specifies a number of attributes which this object must  
 131881 have and a number of functions which manipulate these attributes. These attributes  
 131882 include *detachstate*, *inheritsched*, *schedparam*, *schedpolicy*, *contentionscope*, *stackaddr*, and  
 131883 *stacksize*.

131884 The System Interfaces volume of POSIX.1-2024 specifies another mutex attribute called  
 131885 *type*. The *type* attribute allows applications to specify the behavior of mutex locking  
 131886 operations in situations where POSIX.1c behavior is undefined. The OSF DCE threads  
 131887 implementation, based on Draft 4 of POSIX.1c, specified a similar attribute. Note that the  
 131888 names of the attributes have changed somewhat from the OSF DCE threads  
 131889 implementation.

131890 The System Interfaces volume of POSIX.1-2024 also extends the specification of the  
 131891 following POSIX.1c functions which manipulate mutexes:

131892 `pthread_mutex_lock()`  
 131893 `pthread_mutex_trylock()`  
 131894 `pthread_mutex_unlock()`

131895 to take account of the new mutex attribute type and to specify behavior which was  
 131896 declared as undefined in POSIX.1c. How a calling thread acquires or releases a mutex now  
 131897 depends upon the mutex *type* attribute.

131898 The *type* attribute can have the following values:

131899 PTHREAD\_MUTEX\_NORMAL

131900 Basic mutex with no specific error checking built in. Does not report a deadlock error.

131901 PTHREAD\_MUTEX\_RECURSIVE

131902 Allows any thread to recursively lock a mutex. The mutex must be unlocked an equal  
 131903 number of times to release the mutex.

131904 PTHREAD\_MUTEX\_ERRORCHECK

131905 Detects and reports simple usage errors; that is, an attempt to unlock a mutex that is  
 131906 not locked by the calling thread or that is not locked at all, or an attempt to relock a  
 131907 mutex the thread already owns.

131908 PTHREAD\_MUTEX\_DEFAULT

131909 The default mutex type. May be mapped to any of the above mutex types or may be  
 131910 an implementation-defined type.

131911 *Normal* mutexes do not detect deadlock conditions; for example, a thread will hang if it  
 131912 tries to relock a normal mutex that it already owns. Attempting to unlock a mutex locked  
 131913 by another thread, or unlocking an unlocked mutex, results in undefined behavior. Normal  
 131914 mutexes will usually be the fastest type of mutex available on a platform but provide the  
 131915 least error checking.

131916 *Recursive* mutexes are useful for converting old code where it is difficult to establish clear  
 131917 boundaries of synchronization. A thread can relock a recursive mutex without first  
 131918 unlocking it. The relocking deadlock which can occur with normal mutexes cannot occur  
 131919 with this type of mutex. However, multiple locks of a recursive mutex require the same  
 131920 number of unlocks to release the mutex before another thread can acquire the mutex.  
 131921 Furthermore, this type of mutex maintains the concept of an owner. Thus, a thread  
 131922 attempting to unlock a recursive mutex which another thread has locked returns with an  
 131923 error. A thread attempting to unlock a recursive mutex that is not locked returns with an  
 131924 error. Never use a recursive mutex with condition variables because the implicit unlock  
 131925 performed by `pthread_cond_clockwait()`, `pthread_cond_timedwait()`, or `pthread_cond_wait()`  
 131926 will not actually release the mutex if it had been locked multiple times.

131927 *Errorcheck* mutexes provide error checking and are useful primarily as a debugging aid. A  
 131928 thread attempting to relock an errorcheck mutex without first unlocking it returns with an  
 131929 error. Again, this type of mutex maintains the concept of an owner. Thus, a thread  
 131930 attempting to unlock an errorcheck mutex which another thread has locked returns with  
 131931 an error. A thread attempting to unlock an errorcheck mutex that is not locked also returns  
 131932 with an error. It should be noted that errorcheck mutexes will almost always be much  
 131933 slower than normal mutexes due to the extra state checks performed.

131934 The default mutex type provides implementation-defined error checking. The default  
 131935 mutex may be mapped to one of the other defined types or may be something entirely  
 131936 different. This enables each vendor to provide the mutex semantics which the vendor feels  
 131937 will be most useful to their target users. Most vendors will probably choose to make  
 131938 normal mutexes the default so as to give applications the benefit of the fastest type of



- 131939 mutexes available on their platform. Check your implementation's documentation.
- 131940 An application developer can use any of the mutex types almost interchangeably as long  
131941 as the application does not depend upon the implementation detecting (or failing to  
131942 detect) any particular errors. Note that a recursive mutex can be used with condition  
131943 variable waits as long as the application never recursively locks the mutex.
- 131944 Two functions are provided for manipulating the *type* attribute of a mutex attributes object.  
131945 This attribute is set or returned in the *type* parameter of these functions. The  
131946 *pthread\_mutexattr\_settype()* function is used to set a specific type value while  
131947 *pthread\_mutexattr\_gettype()* is used to return the type of the mutex. Setting the *type*  
131948 attribute of a mutex attributes object affects only mutexes initialized using that mutex  
131949 attributes object. Changing the *type* attribute does not affect mutexes previously initialized  
131950 using that mutex attributes object.
- 131951 • Read-Write Locks and Attributes
- 131952 The read-write locks introduced have been harmonized with those in IEEE Std  
131953 1003.1j-2000; see also [Section B.2.9.6](#) (on page 3833).
- 131954 Read-write locks (also known as reader-writer locks) allow a thread to exclusively lock  
131955 some shared data while updating that data, or allow any number of threads to have  
131956 simultaneous read-only access to the data.
- 131957 Unlike a mutex, a read-write lock distinguishes between reading data and writing data. A  
131958 mutex excludes all other threads. A read-write lock allows other threads access to the data,  
131959 providing no thread is modifying the data. Thus, a read-write lock is less primitive than  
131960 either a mutex-condition variable pair or a semaphore.
- 131961 Application developers should consider using a read-write lock rather than a mutex to  
131962 protect data that is frequently referenced but seldom modified. Most threads (readers) will  
131963 be able to read the data without waiting and will only have to block when some other  
131964 thread (a writer) is in the process of modifying the data. Conversely a thread that wants to  
131965 change the data is forced to wait until there are no readers. This type of lock is often used  
131966 to facilitate parallel access to data on multi-processor platforms or to avoid context  
131967 switches on single processor platforms where multiple threads access the same data.
- 131968 If a read-write lock becomes unlocked and there are multiple threads waiting to acquire  
131969 the write lock, the implementation's scheduling policy determines which thread acquires  
131970 the read-write lock for writing. If there are multiple threads blocked on a read-write lock  
131971 for both read locks and write locks, it is unspecified whether the readers or a writer  
131972 acquire the lock first. However, for performance reasons, implementations often favor  
131973 writers over readers to avoid potential writer starvation.
- 131974 A read-write lock object is an implementation-defined opaque object of type  
131975 **pthread\_rwlock\_t** as defined in `<pthread.h>`. There are two different sorts of locks  
131976 associated with a read-write lock: a read lock and a write lock.
- 131977 The *pthread\_rwlockattr\_init()* function initializes a read-write lock attributes object with the  
131978 default value for all the attributes defined in the implementation. After a read-write lock  
131979 attributes object has been used to initialize one or more read-write locks, changes to the  
131980 read-write lock attributes object, including destruction, do not affect previously initialized  
131981 read-write locks.
- 131982 Implementations must provide at least the read-write lock attribute *process-shared*. This  
131983 attribute can have the following values:

- 131984 PTHREAD\_PROCESS\_SHARED  
 131985 Any thread of any process that has access to the memory where the read-write lock  
 131986 resides can manipulate the read-write lock.
- 131987 PTHREAD\_PROCESS\_PRIVATE  
 131988 Only threads created within the same process as the thread that initialized the read-  
 131989 write lock can manipulate the read-write lock. This is the default value.
- 131990 The *pthread\_rwlockattr\_setpshared()* function is used to set the *process-shared* attribute of an  
 131991 initialized read-write lock attributes object while the function  
 131992 *pthread\_rwlockattr\_getpshared()* obtains the current value of the *process-shared* attribute.
- 131993 A read-write lock attributes object is destroyed using the *pthread\_rwlockattr\_destroy()*  
 131994 function. The effect of subsequent use of the read-write lock attributes object is undefined.
- 131995 A thread creates a read-write lock using the *pthread\_rwlock\_init()* function. The attributes  
 131996 of the read-write lock can be specified by the application developer; otherwise, the default  
 131997 implementation-defined read-write lock attributes are used if the pointer to the read-write  
 131998 lock attributes object is NULL. In cases where the default attributes are appropriate, the  
 131999 PTHREAD\_RWLOCK\_INITIALIZER macro can be used to initialize read-write locks.
- 132000 A thread which wants to apply a read lock to the read-write lock can use either  
 132001 *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_tryrdlock()*. If *pthread\_rwlock\_rdlock()* is used, the  
 132002 thread acquires a read lock if a writer does not hold the write lock and there are no writers  
 132003 blocked on the write lock. If a read lock is not acquired, the calling thread blocks until it  
 132004 can acquire a lock. However, if *pthread\_rwlock\_tryrdlock()* is used, the function returns  
 132005 immediately with the error [EBUSY] if any thread holds a write lock or there are blocked  
 132006 writers waiting for the write lock.
- 132007 A thread which wants to apply a write lock to the read-write lock can use either of two  
 132008 functions: *pthread\_rwlock\_wrlock()* or *pthread\_rwlock\_trywrlock()*. If *pthread\_rwlock\_wrlock()*  
 132009 is used, the thread acquires the write lock if no other reader or writer threads hold the  
 132010 read-write lock. If the write lock is not acquired, the thread blocks until it can acquire the  
 132011 write lock. However, if *pthread\_rwlock\_trywrlock()* is used, the function returns  
 132012 immediately with the error [EBUSY] if any thread is holding either a read or a write lock.
- 132013 The *pthread\_rwlock\_unlock()* function is used to unlock a read-write lock object held by the  
 132014 calling thread. Results are undefined if the read-write lock is not held by the calling thread.  
 132015 If there are other read locks currently held on the read-write lock object, the read-write  
 132016 lock object remains in the read locked state but without the current thread as one of its  
 132017 owners. If this function releases the last read lock for this read-write lock object, the read-  
 132018 write lock object is put in the unlocked read state. If this function is called to release a write  
 132019 lock for this read-write lock object, the read-write lock object is put in the unlocked state.
- 132020 • Thread Concurrency Level
- 132021 On threads implementations that multiplex user threads onto a smaller set of kernel  
 132022 execution entities, the system attempts to create a reasonable number of kernel execution  
 132023 entities for the application upon application startup.
- 132024 On some implementations, these kernel entities are retained by user threads that block in  
 132025 the kernel. Other implementations do not *timeslice* user threads so that multiple compute-  
 132026 bound user threads can share a kernel thread. On such implementations, some  
 132027 applications may use up all the available kernel execution entities before their user-space  
 132028 threads are used up. The process may be left with user threads capable of doing work for  
 132029 the application but with no way to schedule them.

## 132030 • Thread Stack Guard Size

132031 DCE threads introduced the concept of a “thread stack guard size”. Most thread  
132032 implementations add a region of protected memory to a thread’s stack, commonly known  
132033 as a “guard region”, as a safety measure to prevent stack pointer overflow in one thread  
132034 from corrupting the contents of another thread’s stack. The default size of the guard  
132035 regions attribute is {PAGESIZE} bytes and is implementation-defined.

132036 Some application developers may wish to change the stack guard size. When an  
132037 application creates a large number of threads, the extra page allocated for each stack may  
132038 strain system resources. In addition to the extra page of memory, the kernel’s memory  
132039 manager has to keep track of the different protections on adjoining pages. When this is a  
132040 problem, the application developer may request a guard size of 0 bytes to conserve system  
132041 resources by eliminating stack overflow protection.

132042 Conversely an application that allocates large data structures such as arrays on the stack  
132043 may wish to increase the default guard size in order to detect stack overflow. If a thread  
132044 allocates two pages for a data array, a single guard page provides little protection against  
132045 thread stack overflows since the thread can corrupt adjoining memory beyond the guard  
132046 page.

132047 The System Interfaces volume of POSIX.1-2024 defines a new attribute of a thread  
132048 attributes object; that is, the *guardsize* attribute which allows applications to specify the size  
132049 of the guard region of a thread’s stack.

132050 Two functions are provided for manipulating a thread’s stack guard size. The  
132051 *pthread\_attr\_setguardsize()* function sets the thread *guardsize* attribute, and the  
132052 *pthread\_attr\_getguardsize()* function retrieves the current value.

132053 An implementation may round up the requested guard size to a multiple of the  
132054 configurable system variable {PAGESIZE}. In this case, *pthread\_attr\_getguardsize()* returns  
132055 the guard size specified by the previous *pthread\_attr\_setguardsize()* function call and not  
132056 the rounded up value.

132057 If an application is managing its own thread stacks using the *stackaddr* attribute, the  
132058 *guardsize* attribute is ignored and no stack overflow protection is provided. In this case, it is  
132059 the responsibility of the application to manage stack overflow along with stack allocation.

## 132060 • Parallel I/O

132061 Suppose two or more threads independently issue read requests on the same file. To read  
132062 specific data from a file, a thread must first call *lseek()* to seek to the proper offset in the  
132063 file, and then call *read()* to retrieve the required data. If more than one thread does this at  
132064 the same time, the first thread may complete its seek call, but before it gets a chance to  
132065 issue its read call a second thread may complete its seek call, resulting in the first thread  
132066 accessing incorrect data when it issues its read call. One workaround is to lock the file  
132067 descriptor while seeking and reading or writing, but this reduces parallelism and adds  
132068 overhead.

132069 Instead, the System Interfaces volume of POSIX.1-2024 provides two functions to make  
132070 seek/read and seek/write operations atomic. The file descriptor’s current offset is  
132071 unchanged, thus allowing multiple read and write operations to proceed in parallel. This  
132072 improves the I/O performance of threaded applications. The *pread()* function is used to do  
132073 an atomic read of data from a file into a buffer. Conversely, the *pwrite()* function does an  
132074 atomic write of data from a buffer to a file.

132075 B.2.9.1 *Thread-Safety*

132076 All functions required by POSIX.1-2024 need to be thread-safe. Implementations have to  
 132077 provide internal synchronization when necessary in order to achieve this goal. In certain cases—  
 132078 for example, most floating-point implementations—context switch code may have to manage  
 132079 the writable shared state.

132080 While a read from a pipe of  $\{\text{PIPE\_BUF}\} \times 2$  bytes may not generate a single atomic and thread-  
 132081 safe stream of bytes, it should generate “several” (individually atomic) thread-safe streams of  
 132082 bytes. Similarly, while reading from a terminal device may not generate a single atomic and  
 132083 thread-safe stream of bytes, it should generate some finite number of (individually atomic) and  
 132084 thread-safe streams of bytes. That is, concurrent calls to read for a pipe, FIFO, or terminal device  
 132085 are not allowed to result in corrupting the stream of bytes or other internal data. However,  
 132086 *read()*, in these cases, is not required to return a single contiguous and atomic stream of bytes.

132087 It is not required that all functions provided by POSIX.1-2024 be either async-cancel-safe or  
 132088 async-signal-safe.

132089 As it turns out, some functions are inherently not thread-safe; that is, their interface  
 132090 specifications preclude thread-safety. For example, some functions (such as *asctime()*) return a  
 132091 pointer to a result stored in memory space allocated by the function on a per-process basis. Such  
 132092 a function is not thread-safe, because its result can be overwritten by successive invocations.  
 132093 Other functions, while not inherently non-thread-safe, may be implemented in ways that lead to  
 132094 them not being thread-safe. For example, some functions (such as *rand()*) store state information  
 132095 (such as a seed value, which survives multiple function invocations) in memory space allocated  
 132096 by the function on a per-process basis. The implementation of such a function is not thread-safe  
 132097 if the implementation fails to synchronize invocations of the function and thus fails to protect  
 132098 the state information. The problem is that when the state information is not protected,  
 132099 concurrent invocations can interfere with one another (for example, applications using *rand()*  
 132100 may see the same seed value).

132101 *Thread-Safety and Locking of Existing Functions*

132102 Originally, POSIX.1 was not designed to work in a multi-threaded environment, and some  
 132103 implementations of some existing functions will not work properly when executed concurrently.  
 132104 To provide routines that will work correctly in an environment with threads (“thread-safe”), two  
 132105 problems need to be solved:

- 132106 1. Routines that maintain or return pointers to static areas internal to the routine (which  
 132107 may now be shared) need to be modified. The routines *ttynname()* and *localtime()* are  
 132108 examples.
- 132109 2. Routines that access data space shared by more than one thread need to be modified. The  
 132110 *malloc()* function and the *stdio* family routines are examples.

132111 There are a variety of constraints on these changes. The first is compatibility with the existing  
 132112 versions of these functions—non-thread-safe functions will continue to be in use for some time,  
 132113 as the original interfaces are used by existing code. Another is that the new thread-safe versions  
 132114 of these functions represent as small a change as possible over the familiar interfaces provided  
 132115 by the existing non-thread-safe versions. The new interfaces should be independent of any  
 132116 particular threads implementation. In particular, they should be thread-safe without depending  
 132117 on explicit thread-specific memory. Finally, there should be minimal performance penalty due to  
 132118 the changes made to the functions.

132119 It is intended that the list of functions from POSIX.1 that cannot be made thread-safe and for  
 132120 which corrected versions are provided be complete.

132121 *Thread-Safety and Locking Solutions*

132122 Many of the POSIX.1 functions were thread-safe and did not change at all. However, some  
132123 functions (for example, the math functions typically found in **libm**) are not thread-safe because  
132124 of writable shared global state. For instance, in IEEE Std 754-1985 floating-point  
132125 implementations, the computation modes and flags are global and shared.

132126 Some functions are not thread-safe because a particular implementation is not reentrant,  
132127 typically because of a non-essential use of static storage. These require only a new  
132128 implementation.

132129 Thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming  
132130 environments, not just within pthreads. In order to be used outside the context of pthreads,  
132131 however, such libraries still have to use some synchronization method. These could either be  
132132 independent of the pthread synchronization operations, or they could be a subset of the pthread  
132133 interfaces. Either method results in thread-safe library implementations that can be used without  
132134 the rest of pthreads.

132135 Some functions, such as the *stdio* family interface and dynamic memory allocation functions  
132136 such as *malloc()*, are inter-dependent routines that share resources (for example, buffers) across  
132137 related calls. These require synchronization to work correctly, but they do not require any  
132138 change to their external (user-visible) interfaces.

132139 In some cases, such as *getc()* and *putc()*, adding synchronization is likely to create an  
132140 unacceptable performance impact. In this case, slower thread-safe synchronized functions are to  
132141 be provided, but the original, faster (but unsafe) functions (which may be implemented as  
132142 macros) are retained under new names. Some additional special-purpose synchronization  
132143 facilities are necessary for these macros to be usable in multi-threaded programs. This also  
132144 requires changes in **<stdio.h>**.

132145 The other common reason that functions are unsafe is that they return a pointer to static storage,  
132146 making the functions non-thread-safe. This has to be changed, and there are three natural  
132147 choices:

132148 1. Return a pointer to thread-specific storage

132149 This could incur a severe performance penalty on those architectures with a costly  
132150 implementation of the thread-specific data interface.

132151 A variation on this technique is to use *malloc()* to allocate storage for the function output  
132152 and return a pointer to this storage. This technique may also have an undesirable  
132153 performance impact, however, and a simplistic implementation requires that the user  
132154 program explicitly free the storage object when it is no longer needed. This technique is  
132155 used by some existing POSIX.1 functions. With careful implementation for infrequently  
132156 used functions, there may be little or no performance or storage penalty, and the  
132157 maintenance of already-standardized interfaces is a significant benefit.

132158 2. Return the actual value computed by the function

132159 This technique can only be used with functions that return pointers to structures—  
132160 routines that return character strings would have to wrap their output in an enclosing  
132161 structure in order to return the output on the stack. There is also a negative performance  
132162 impact inherent in this solution in that the output value has to be copied twice before it  
132163 can be used by the calling function: once from the called routine's local buffers to the top  
132164 of the stack, then from the top of the stack to the assignment target. Finally, many older  
132165 compilers cannot support this technique due to a historical tendency to use internal static  
132166 buffers to deliver the results of structure-valued functions.

- 132167 3. Have the caller pass the address of a buffer to contain the computed value
- 132168 The only disadvantage of this approach is that extra arguments have to be provided by
- 132169 the calling program. It represents the most efficient solution to the problem, however,
- 132170 and, unlike the `malloc()` technique, it is semantically clear.
- 132171 There are some routines (often groups of related routines) whose interfaces are inherently non-
- 132172 thread-safe because they communicate across multiple function invocations by means of static
- 132173 memory locations. The solution is to redesign the calls so that they are thread-safe, typically by
- 132174 passing the needed data as extra parameters. Unfortunately, this may require major changes to
- 132175 the interface as well.
- 132176 A floating-point implementation using IEEE Std 754-1985 is a case in point. A less problematic
- 132177 example is the `rand48` family of pseudo-random number generators. The functions `getgrgid()`,
- 132178 `getgrnam()`, `getpwnam()`, and `getpwuid()` are another such case.
- 132179 The problems with `errno` are discussed in [Alternative Solutions for Per-Thread `errno`](#) (on page
- 132180 3745).
- 132181 Some functions can be thread-safe or not, depending on their arguments. These include the
- 132182 `tmpnam()` and `ctermid()` functions. These functions have pointers to character strings as
- 132183 arguments. If the pointers are not NULL, the functions store their results in the character string;
- 132184 however, if the pointers are NULL, the functions store their results in an area that may be static
- 132185 and thus subject to overwriting by successive calls. These should only be called by multi-thread
- 132186 applications when their arguments are non-NULL.
- 132187 *Asynchronous Safety and Thread-Safety*
- 132188 A floating-point implementation has many modes that effect rounding and other aspects of
- 132189 computation. Functions in some math library implementations may change the computation
- 132190 modes for the duration of a function call. If such a function call is interrupted by a signal or
- 132191 cancellation, the floating-point state is not required to be protected.
- 132192 There is a significant cost to make floating-point operations async-cancel-safe or async-signal-
- 132193 safe; accordingly, neither form of async safety is required.
- 132194 *Functions Returning Pointers to Static Storage*
- 132195 For those functions that are not thread-safe because they return values in fixed size statically
- 132196 allocated structures, alternate ```_r``` forms are provided that pass a pointer to an explicit result
- 132197 structure. Those that return pointers into library-allocated buffers have forms provided with
- 132198 explicit buffer and length parameters.
- 132199 For functions that return pointers to library-allocated buffers, it makes sense to provide ```_r```
- 132200 versions that allow the application control over allocation of the storage in which results are
- 132201 returned. This allows the state used by these functions to be managed on an application-specific
- 132202 basis, supporting per-thread, per-process, or other application-specific sharing relationships.
- 132203 Early proposals had provided ```_r``` versions for functions that returned pointers to variable-size
- 132204 buffers without providing a means for determining the required buffer size. This would have
- 132205 made using such functions exceedingly clumsy, potentially requiring iteratively calling them
- 132206 with increasingly larger guesses for the amount of storage required. Hence, `sysconf()` variables
- 132207 have been provided for such functions that return the maximum required buffer size.
- 132208 Thus, the rule that has been followed by POSIX.1-2024 when adapting single-threaded non-
- 132209 thread-safe functions is as follows: all functions returning pointers to library-allocated storage
- 132210 should have ```_r``` versions provided, allowing the application control over the storage
- 132211 allocation. Those with variable-sized return values accept both a buffer address and a length
- 132212 parameter. The `sysconf()` variables are provided to supply the appropriate buffer sizes when

- 132213 required. Implementors are encouraged to apply the same rule when adapting their own  
132214 existing functions to a pthreads environment.
- 132215 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0020 [631], XSH/TC2-2008/0021 [826],  
132216 and XSH/TC2-2008/0022 [631] are applied.
- 132217 Austin Group Defect 188 is applied, removing *getenv()* from the list of functions that need not be  
132218 thread-safe.
- 132219 Austin Group Defect 696 is applied, requiring *readdir()* to be thread-safe except when concurrent  
132220 calls are made for the same directory stream.
- 132221 Austin Group Defect 922 is applied, adding the *secure\_getenv()* function.
- 132222 Austin Group Defect 1064 is applied, removing *basename()* and *dirname()* from the list of  
132223 functions that need not be thread-safe.
- 132224 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

132225 *B.2.9.2 Thread IDs*

132226 Separate applications should communicate through well-defined interfaces and should not  
132227 depend on each other's implementation. For example, if a programmer decides to rewrite the  
132228 *sort* utility using multiple threads, it should be easy to do this so that the interface to the *sort*  
132229 utility does not change. Consider that if the user causes SIGINT to be generated while the *sort*  
132230 utility is running, keeping the same interface means that the entire *sort* utility is killed, not just  
132231 one of its threads. As another example, consider a realtime application that manages a reactor.  
132232 Such an application may wish to allow other applications to control the priority at which it  
132233 watches the control rods. One technique to accomplish this is to write the ID of the thread  
132234 watching the control rods into a file and allow other programs to change the priority of that  
132235 thread as they see fit. A simpler technique is to have the reactor process accept IPCs  
132236 (Interprocess Communication messages) from other processes, telling it at a semantic level what  
132237 priority the program should assign to watching the control rods. This allows the programmer  
132238 greater flexibility in the implementation. For example, the programmer can change the  
132239 implementation from having one thread per rod to having one thread watching all of the rods  
132240 without changing the interface. Having threads live inside the process means that the  
132241 implementation of a process is invisible to outside processes (excepting debuggers and system  
132242 management tools).

132243 Threads do not provide a protection boundary. Every thread model allows threads to share  
132244 memory with other threads and encourages this sharing to be widespread. This means that one  
132245 thread can wipe out memory that is needed for the correct functioning of other threads that are  
132246 sharing its memory. Consequently, providing each thread with its own user and/or group IDs  
132247 would not provide a protection boundary between threads sharing memory.

132248 Some applications make the assumption that the implementation can always detect invalid uses  
132249 of thread IDs of type **pthread\_t**. This is an invalid assumption. Specifically, if **pthread\_t**  
132250 is defined as a pointer type, no access check needs to be performed before using the ID.

132251 As with other interfaces that take pointer parameters, the outcome of passing an invalid  
132252 parameter can result in an invalid memory reference or an attempt to access an undefined  
132253 portion of a memory object, cause signals to be sent (SIGSEGV or SIGBUS) and possible  
132254 termination of the process. This is a similar case to passing an invalid buffer pointer to *read()*.  
132255 Some implementations might implement *read()* as a system call and set an [EFAULT] error  
132256 condition. Other implementations might contain parts of *read()* at user level and the first  
132257 attempt to access data at an invalid reference will cause a signal to be sent instead.

132258 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended

132259 that the function should fail and report an [ESRCH] error. This does not imply that  
 132260 implementations are required to return in this case. It is legitimate behavior to send an “invalid  
 132261 memory reference” signal (SIGSEGV or SIGBUS). It is the application’s responsibility to use only  
 132262 valid thread IDs and to keep track of the lifetime of the underlying threads.

132263 Austin Group Defect 792 is applied, clarifying thread lifetime.

132264 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

#### 132265 B.2.9.3 Thread Mutexes

132266 Austin Group Defect 1216 is applied, adding `pthread_cond_clockwait()` and  
 132267 `pthread_mutex_clocklock()`.

132268 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

#### 132269 B.2.9.4 Thread Scheduling

##### 132270 • Scheduling Implementation Models

132271 The following scheduling implementation models are presented in terms of threads and  
 132272 “kernel entities”. This is to simplify exposition of the models, and it does not imply that  
 132273 an implementation actually has an identifiable “kernel entity”.

132274 A kernel entity is not defined beyond the fact that it has scheduling attributes that are used  
 132275 to resolve contention with other kernel entities for execution resources. A kernel entity  
 132276 may be thought of as an envelope that holds a thread or a separate kernel thread. It is not a  
 132277 conventional process, although it shares with the process the attribute that it has a single  
 132278 thread of control; it does not necessarily imply an address space, open files, and so on. It is  
 132279 better thought of as a primitive facility upon which conventional processes and threads  
 132280 may be constructed.

##### 132281 — System Thread Scheduling Model

132282 This model consists of one thread per kernel entity. The kernel entity is solely  
 132283 responsible for scheduling thread execution on one or more processors. This model  
 132284 schedules all threads against all other threads in the system using the scheduling  
 132285 attributes of the thread.

##### 132286 — Process Scheduling Model

132287 A generalized process scheduling model consists of two levels of scheduling. A  
 132288 threads library creates a pool of kernel entities, as required, and schedules threads to  
 132289 run on them using the scheduling attributes of the threads. Typically, the size of the  
 132290 pool is a function of the simultaneously runnable threads, not the total number of  
 132291 threads. The kernel then schedules the kernel entities onto processors according to  
 132292 their scheduling attributes, which are managed by the threads library. This set model  
 132293 potentially allows a wide range of mappings between threads and kernel entities.

##### 132294 • System and Process Scheduling Model Performance

132295 There are a number of important implications on the performance of applications using  
 132296 these scheduling models. The process scheduling model potentially provides lower  
 132297 overhead for making scheduling decisions, since there is no need to access kernel-level  
 132298 information or functions and the set of schedulable entities is smaller (only the threads  
 132299 within the process).

132300 On the other hand, since the kernel is also making scheduling decisions regarding the



132301 system resources under its control (for example, CPU(s), I/O devices, memory), decisions  
132302 that do not take thread scheduling parameters into account can result in unspecified  
132303 delays for realtime application threads, causing them to miss maximum response time  
132304 limits.

132305 • Rate Monotonic Scheduling

132306 Rate monotonic scheduling was considered, but rejected for standardization in the context  
132307 of pthreads. A sporadic server policy is included.

132308 • Scheduling Options

132309 In POSIX.1-2024, the basic thread scheduling functions are defined under the threads  
132310 functionality, so that they are required of all threads implementations. However, there are  
132311 no specific scheduling policies required by this functionality to allow for conforming  
132312 thread implementations that are not targeted to realtime applications.

132313 Specific standard scheduling policies are defined to be under the Thread Execution  
132314 Scheduling option, and they are specifically designed to support realtime applications by  
132315 providing predictable resource-sharing sequences. The name of this option was chosen to  
132316 emphasize that this functionality is defined as appropriate for realtime applications that  
132317 require simple priority-based scheduling.

132318 It is recognized that these policies are not necessarily satisfactory for some multi-processor  
132319 implementations, and work is ongoing to address a wider range of scheduling behaviors.  
132320 The interfaces have been chosen to create abundant opportunity for future scheduling  
132321 policies to be implemented and standardized based on this interface. In order to  
132322 standardize a new scheduling policy, all that is required (from the standpoint of thread  
132323 scheduling attributes) is to define a new policy name, new members of the thread  
132324 attributes object, and functions to set these members when the scheduling policy is equal  
132325 to the new value.

132326 **Scheduling Contention Scope**

132327 In order to accommodate the requirement for realtime response, each thread has a scheduling  
132328 contention scope attribute. Threads with a system scheduling contention scope have to be  
132329 scheduled with respect to all other threads in the system. These threads are usually bound to a  
132330 single kernel entity that reflects their scheduling attributes and are directly scheduled by the  
132331 kernel.

132332 Threads with a process scheduling contention scope need be scheduled only with respect to the  
132333 other threads in the process. These threads may be scheduled within the process onto a pool of  
132334 kernel entities. The implementation is also free to bind these threads directly to kernel entities  
132335 and let them be scheduled by the kernel. Process scheduling contention scope allows the  
132336 implementation the most flexibility and is the default if both contention scopes are supported  
132337 and none is specified.

132338 Thus, the choice by implementors to provide one or the other (or both) of these scheduling  
132339 models is driven by the need of their supported application domains for worst-case (that is,  
132340 realtime) response, or average-case (non-realtime) response.

**132341 Scheduling Allocation Domain**

132342 The SCHED\_FIFO and SCHED\_RR scheduling policies take on different characteristics on a  
132343 multi-processor. Other scheduling policies are also subject to changed behavior when executed  
132344 on a multi-processor. The concept of scheduling allocation domain determines the set of  
132345 processors on which the threads of an application may run. By considering the application's  
132346 processor scheduling allocation domain for its threads, scheduling policies can be defined in  
132347 terms of their behavior for varying processor scheduling allocation domain values. It is  
132348 conceivable that not all scheduling allocation domain sizes make sense for all scheduling  
132349 policies on all implementations. The concept of scheduling allocation domain, however, is a  
132350 useful tool for the description of multi-processor scheduling policies.

132351 The "process control" approach to scheduling obtains significant performance advantages from  
132352 dynamic scheduling allocation domain sizes when it is applicable.

132353 Non-Uniform Memory Access (NUMA) multi-processors may use a system scheduling structure  
132354 that involves reassignment of threads among scheduling allocation domains. In NUMA  
132355 machines, a natural model of scheduling is to match scheduling allocation domains to clusters of  
132356 processors. Load balancing in such an environment requires changing the scheduling allocation  
132357 domain to which a thread is assigned.

**132358 Scheduling Documentation**

132359 Implementation-provided scheduling policies need to be completely documented in order to be  
132360 useful. This documentation includes a description of the attributes required for the policy, the  
132361 scheduling interaction of threads running under this policy and all other supported policies, and  
132362 the effects of all possible values for processor scheduling allocation domain. Note that for the  
132363 implementor wishing to be minimally-compliant, it is (minimally) acceptable to define the  
132364 behavior as undefined.

**132365 Scheduling Contention Scope Attribute**

132366 The scheduling contention scope defines how threads compete for resources. Within  
132367 POSIX.1-2024, scheduling contention scope is used to describe only how threads are scheduled  
132368 in relation to one another in the system. That is, either they are scheduled against all other  
132369 threads in the system ("system scope") or only against those threads in the process ("process  
132370 scope"). In fact, scheduling contention scope may apply to additional resources, including  
132371 virtual timers and profiling, which are not currently considered by POSIX.1-2024.

**132372 Mixed Scopes**

132373 If only one scheduling contention scope is supported, the scheduling decision is straightforward.  
132374 To perform the processor scheduling decision in a mixed scope environment, it is necessary to  
132375 map the scheduling attributes of the thread with process-wide contention scope to the same  
132376 attribute space as the thread with system-wide contention scope.

132377 Since a conforming implementation has to support one and may support both scopes, it is useful  
132378 to discuss the effects of such choices with respect to example applications. If an implementation  
132379 supports both scopes, mixing scopes provides a means of better managing system-level (that is,  
132380 kernel-level) and library-level resources. In general, threads with system scope will require the  
132381 resources of a separate kernel entity in order to guarantee the scheduling semantics. On the  
132382 other hand, threads with process scope can share the resources of a kernel entity while  
132383 maintaining the scheduling semantics.

132384 The application is free to create threads with dedicated kernel resources, and other threads that  
132385 multiplex kernel resources. Consider the example of a window server. The server allocates two  
132386 threads per widget: one thread manages the widget user interface (including drawing), while

132387 the other thread takes any required application action. This allows the widget to be “active”  
132388 while the application is computing. A screen image may be built from thousands of widgets. If  
132389 each of these threads had been created with system scope, then most of the kernel-level  
132390 resources might be wasted, since only a few widgets are active at any one time. In addition,  
132391 mixed scope is particularly useful in a window server where one thread with high priority and  
132392 system scope handles the mouse so that it tracks well. As another example, consider a database  
132393 server. For each of the hundreds or thousands of clients supported by a large server, an  
132394 equivalent number of threads will have to be created. If each of these threads were system scope,  
132395 the consequences would be the same as for the window server example above. However, the  
132396 server could be constructed so that actual retrieval of data is done by several dedicated threads.  
132397 Dedicated threads that do work for all clients frequently justify the added expense of system  
132398 scope. If it were not permissible to mix system and process threads in the same process, this type  
132399 of solution would not be possible.

#### 132400 **Dynamic Thread Scheduling Parameters Access**

132401 In many time-constrained applications, there is no need to change the scheduling attributes  
132402 dynamically during thread or process execution, since the general use of these attributes is to  
132403 reflect directly the time constraints of the application. Since these time constraints are generally  
132404 imposed to meet higher-level system requirements, such as accuracy or availability, they  
132405 frequently should remain unchanged during application execution.

132406 However, there are important situations in which the scheduling attributes should be changed.  
132407 Generally, this will occur when external environmental conditions exist in which the time  
132408 constraints change. Consider, for example, a space vehicle major mode change, such as the  
132409 change from ascent to descent mode, or the change from the space environment to the  
132410 atmospheric environment. In such cases, the frequency with which many of the sensors or  
132411 actuators need to be read or written will change, which will necessitate a priority change. In  
132412 other cases, even the existence of a time constraint might be temporary, necessitating not just a  
132413 priority change, but also a policy change for ongoing threads or processes. For this reason, it is  
132414 critical that the interface should provide functions to change the scheduling parameters  
132415 dynamically, but, as with many of the other realtime functions, it is important that applications  
132416 use them properly to avoid the possibility of unnecessarily degrading performance.

132417 In providing functions for dynamically changing the scheduling behavior of threads, there were  
132418 two options: provide functions to get and set the individual scheduling parameters of threads,  
132419 or provide a single interface to get and set all the scheduling parameters for a given thread  
132420 simultaneously. Both approaches have merit. Access functions for individual parameters allow  
132421 simpler control of thread scheduling for simple thread scheduling parameters. However, a single  
132422 function for setting all the parameters for a given scheduling policy is required when first setting  
132423 that scheduling policy. Since the single all-encompassing functions are required, it was decided  
132424 to leave the interface as minimal as possible. Note that simpler functions (such as  
132425 *pthread\_setprio()* for threads running under the priority-based schedulers) can be easily defined  
132426 in terms of the all-encompassing functions.

132427 If the *pthread\_setschedparam()* function executes successfully, it will have set all of the scheduling  
132428 parameter values indicated in *param*; otherwise, none of the scheduling parameters will have  
132429 been modified. This is necessary to ensure that the scheduling of this and all other threads  
132430 continues to be consistent in the presence of an erroneous scheduling parameter.

132431 The [EPERM] error value is included in the list of possible *pthread\_setschedparam()* error returns  
132432 as a reflection of the fact that the ability to change scheduling parameters increases risks to the  
132433 implementation and application performance if the scheduling parameters are changed  
132434 improperly. For this reason, and based on some existing practice, it was felt that some  
132435 implementations would probably choose to define specific permissions for changing either a

132436 thread's own or another thread's scheduling parameters. POSIX.1-2024 does not include  
132437 portable methods for setting or retrieving permissions, so any such use of permissions is  
132438 completely unspecified.

#### 132439 **Mutex Initialization Scheduling Attributes**

132440 In a priority-driven environment, a direct use of traditional primitives like mutexes and  
132441 condition variables can lead to unbounded priority inversion, where a higher priority thread can  
132442 be blocked by a lower priority thread, or set of threads, for an unbounded duration of time. As a  
132443 result, it becomes impossible to guarantee thread deadlines. Priority inversion can be bounded  
132444 and minimized by the use of priority inheritance protocols. This allows thread deadlines to be  
132445 guaranteed even in the presence of synchronization requirements.

132446 Two useful but simple members of the family of priority inheritance protocols are the basic  
132447 priority inheritance protocol and the priority ceiling protocol emulation. Under the Basic  
132448 Priority Inheritance protocol (governed by the Non-Robust Mutex Priority Inheritance option), a  
132449 thread that is blocking higher priority threads executes at the priority of the highest priority  
132450 thread that it blocks. This simple mechanism allows priority inversion to be bounded by the  
132451 duration of critical sections and makes timing analysis possible.

132452 Under the Priority Ceiling Protocol Emulation protocol (governed by the Thread Priority  
132453 Protection option), each mutex has a priority ceiling, usually defined as the priority of the  
132454 highest priority thread that can lock the mutex. When a thread is executing inside critical  
132455 sections, its priority is unconditionally increased to the highest of the priority ceilings of all the  
132456 mutexes owned by the thread. This protocol has two very desirable properties in uni-processor  
132457 systems. First, a thread can be blocked by a lower priority thread for at most the duration of one  
132458 single critical section. Furthermore, when the protocol is correctly used in a single processor, and  
132459 if threads do not become blocked while owning mutexes, mutual deadlocks are prevented.

132460 The priority ceiling emulation can be extended to multiple processor environments, in which  
132461 case the values of the priority ceilings will be assigned depending on the kind of mutex that is  
132462 being used: local to only one processor, or global, shared by several processors. Local priority  
132463 ceilings will be assigned the usual way, equal to the priority of the highest priority thread that  
132464 may lock that mutex. Global priority ceilings will usually be assigned a priority level higher  
132465 than all the priorities assigned to any of the threads that reside in the involved processors to  
132466 avoid the effect called remote blocking.

#### 132467 **Change the Priority Ceiling of a Mutex**

132468 In order for the priority protect protocol to exhibit its desired properties of bounding priority  
132469 inversion and avoidance of deadlock, it is critical that the ceiling priority of a mutex be the same  
132470 as the priority of the highest thread that can ever hold it, or higher. Thus, if the priorities of the  
132471 threads using such mutexes never change dynamically, there is no need ever to change the  
132472 priority ceiling of a mutex.

132473 However, if a major system mode change results in an altered response time requirement for one  
132474 or more application threads, their priority has to change to reflect it. It will occasionally be the  
132475 case that the priority ceilings of mutexes held also need to change. While changing priority  
132476 ceilings should generally be avoided, it is important that POSIX.1-2024 provide these interfaces  
132477 for those cases in which it is necessary.

132478 B.2.9.5 *Thread Cancellation*

132479 Many existing threads packages have facilities for canceling an operation or canceling a thread.  
132480 These facilities are used for implementing user requests (such as the CANCEL button in a  
132481 window-based application), for implementing OR parallelism (for example, telling the other  
132482 threads to stop working once one thread has found a forced mate in a parallel chess program), or  
132483 for implementing the ABORT mechanism in Ada.

132484 POSIX programs traditionally have used the signal mechanism combined with either *longjmp()*  
132485 or polling to cancel operations. Many POSIX programmers have trouble using these facilities to  
132486 solve their problems efficiently in a single-threaded process. With the introduction of threads,  
132487 these solutions become even more difficult to use.

132488 The main issues with implementing a cancellation facility are specifying the operation to be  
132489 canceled, cleanly releasing any resources allocated to that operation, controlling when the target  
132490 notices that it has been canceled, and defining the interaction between asynchronous signals and  
132491 cancellation.

132492 **Specifying the Operation to Cancel**

132493 Consider a thread that calls through five distinct levels of program abstraction and then, inside  
132494 the lowest-level abstraction, calls a function that suspends the thread. (An abstraction boundary  
132495 is a layer at which the client of the abstraction sees only the service being provided and can  
132496 remain ignorant of the implementation. Abstractions are often layered, each level of abstraction  
132497 being a client of the lower-level abstraction and implementing a higher-level abstraction.)  
132498 Depending on the semantics of each abstraction, one could imagine wanting to cancel only the  
132499 call that causes suspension, only the bottom two levels, or the operation being done by the entire  
132500 thread. Canceling operations at a finer grain than the entire thread is difficult because threads  
132501 are active and they may be run in parallel on a multi-processor. By the time one thread can make  
132502 a request to cancel an operation, the thread performing the operation may have completed that  
132503 operation and gone on to start another operation whose cancellation is not desired. Thread IDs  
132504 are not reused until the thread has exited, and either it was created with the *Attr detachstate*  
132505 attribute set to *PTHREAD\_CREATE\_DETACHED* or the *pthread\_join()* or *pthread\_detach()*  
132506 function has been called for that thread. Consequently, a thread cancellation will never be  
132507 misdirected when the thread terminates. For these reasons, the canceling of operations is done at  
132508 the granularity of the thread. Threads are designed to be inexpensive enough so that a separate  
132509 thread may be created to perform each separately cancelable operation; for example, each  
132510 possibly long running user request.

132511 For cancellation to be used in existing code, cancellation scopes and handlers will have to be  
132512 established for code that needs to release resources upon cancellation, so that it follows the  
132513 programming discipline described in the text.

132514 **A Special Signal Versus a Special Interface**

132515 Two different mechanisms were considered for providing the cancellation interfaces. The first  
132516 was to provide an interface to direct signals at a thread and then to define a special signal that  
132517 had the required semantics. The other alternative was to use a special interface that delivered the  
132518 correct semantics to the target thread.

132519 The solution using signals produced a number of problems. It required the implementation to  
132520 provide cancellation in terms of signals whereas a perfectly valid (and possibly more efficient)  
132521 implementation could have both layered on a low-level set of primitives. There were so many  
132522 exceptions to the special signal (it cannot be used with *kill()*, no POSIX.1 interfaces can be used  
132523 with it) that it was clearly not a valid signal. Its semantics on delivery were also completely  
132524 different from any existing POSIX.1 signal. As such, a special interface that did not mandate the  
132525 implementation and did not confuse the semantics of signals and cancellation was felt to be the

132526 better solution.

### 132527 Races Between Cancellation and Resuming Execution

132528 Due to the nature of cancellation, there is generally no synchronization between the thread  
132529 requesting the cancellation of a blocked thread and events that may cause that thread to resume  
132530 execution. For this reason, and because excess serialization hurts performance, when both an  
132531 event that a thread is waiting for has occurred and a cancellation request has been made and  
132532 cancellation is enabled, POSIX.1-2024 explicitly allows the implementation to choose between  
132533 returning from the blocking call or acting on the cancellation request.

### 132534 Interaction of Cancellation with Asynchronous Signals

132535 A typical use of cancellation is to acquire a lock on some resource and to establish a cancellation  
132536 cleanup handler for releasing the resource when and if the thread is canceled.

132537 A correct and complete implementation of cancellation in the presence of asynchronous signals  
132538 requires considerable care. An implementation has to push a cancellation cleanup handler on the  
132539 cancellation cleanup stack while maintaining the integrity of the stack data structure. If an  
132540 asynchronously-generated signal is posted to the thread during a stack operation, the signal  
132541 handler cannot manipulate the cancellation cleanup stack. As a consequence, asynchronous  
132542 signal handlers may not cancel threads or otherwise manipulate the cancellation state of a  
132543 thread. Threads may, of course, be canceled by another thread that used a *sigwait()* function to  
132544 wait synchronously for an asynchronous signal.

132545 In order for cancellation to function correctly, it is required that asynchronous signal handlers  
132546 not change the cancellation state. This requires that some elements of existing practice, such as  
132547 using *longjmp()* to exit from an asynchronous signal handler implicitly, be prohibited in cases  
132548 where the integrity of the cancellation state of the interrupt thread cannot be ensured.

### 132549 Thread Cancellation Overview

#### 132550 • Cancelability States

132551 The three possible cancelability states (disabled, deferred, and asynchronous) are encoded  
132552 into two separate bits ((disable, enable) and (deferred, asynchronous)) to allow them to be  
132553 changed and restored independently. For instance, short code sequences that will not block  
132554 sometimes disable cancelability on entry and restore the previous state upon exit.  
132555 Likewise, long or unbounded code sequences containing no convenient explicit  
132556 cancellation points will sometimes set the cancelability type to asynchronous on entry and  
132557 restore the previous value upon exit.

#### 132558 • Cancellation Points

132559 Cancellation points are points inside of certain functions where a thread has to act on any  
132560 pending cancellation request when cancelability is enabled. For functions in the “shall  
132561 occur” list, a cancellation check must be performed on every call regardless of whether,  
132562 absent the cancellation, the call would have blocked. For functions in the “may occur” list,  
132563 a cancellation check may be performed on some calls but not others; i.e., whether or not a  
132564 cancellation point occurs when one of these functions is being executed can depend on  
132565 current conditions.

132566 The idea was considered of allowing implementations to define whether blocking calls  
132567 such as *read()* should be cancellation points. It was decided that it would adversely affect  
132568 the design of conforming applications if blocking calls were not cancellation points  
132569 because threads could be left blocked in an uncancelable state.

132570 There are several important blocking routines that are specifically not made cancellation

- 132571 points:
- 132572 — `pthread_mutex_lock()`
- 132573 If `pthread_mutex_lock()` were a cancellation point, every routine that called it would  
 132574 also become a cancellation point (that is, any routine that touched shared state would  
 132575 automatically become a cancellation point). For example, `malloc()`, `free()`, and `rand()`  
 132576 would become cancellation points under this scheme. Having too many cancellation  
 132577 points makes programming very difficult, leading to either much disabling and  
 132578 restoring of cancelability or much difficulty in trying to arrange for reliable cleanup  
 132579 at every possible place.
- 132580 Since `pthread_mutex_lock()` is not a cancellation point, threads could result in being  
 132581 blocked uninterruptibly for long periods of time if mutexes were used as a general  
 132582 synchronization mechanism. As this is normally not acceptable, mutexes should only  
 132583 be used to protect resources that are held for small fixed lengths of time where not  
 132584 being able to be canceled will not be a problem. Resources that need to be held  
 132585 exclusively for long periods of time should be protected with condition variables.
- 132586 — `pthread_barrier_wait()`
- 132587 Canceling a barrier wait will render a barrier unusable. Similar to a barrier timeout  
 132588 (which the standard developers rejected), there is no way to guarantee the  
 132589 consistency of a barrier's internal data structures if a barrier wait is canceled.
- 132590 — `pthread_spin_lock()`
- 132591 As with mutexes, spin locks should only be used to protect resources that are held for  
 132592 small fixed lengths of time where not being cancelable will not be a problem.
- 132593 Every library routine should specify whether or not it includes any cancellation points.  
 132594 Typically, only those routines that may block or compute indefinitely need to include  
 132595 cancellation points.
- 132596 Correctly coded routines only reach cancellation points after having set up a cancellation  
 132597 cleanup handler to restore invariants if the thread is canceled at that point. Being  
 132598 cancelable only at specified cancellation points allows programmers to keep track of  
 132599 actions needed in a cancellation cleanup handler more easily. A thread should only be  
 132600 made asynchronously cancelable when it is not in the process of acquiring or releasing  
 132601 resources or otherwise in a state from which it would be difficult or impossible to recover.
- 132602 • Thread Cancellation Cleanup Handlers
- 132603 The cancellation cleanup handlers provide a portable mechanism, easy to implement, for  
 132604 releasing resources and restoring invariants. They are easier to use than signal handlers  
 132605 because they provide a stack of cancellation cleanup handlers rather than a single handler,  
 132606 and because they have an argument that can be used to pass context information to the  
 132607 handler.
- 132608 The alternative to providing these simple cancellation cleanup handlers (whose only use is  
 132609 for cleaning up when a thread is canceled) is to define a general exception package that  
 132610 could be used for handling and cleaning up after hardware traps and software-detected  
 132611 errors. This was too far removed from the charter of providing threads to handle  
 132612 asynchrony. However, it is an explicit goal of POSIX.1-2024 to be compatible with existing  
 132613 exception facilities and languages having exceptions.
- 132614 The interaction of this facility and other procedure-based or language-level exception  
 132615 facilities is unspecified in this version of POSIX.1-2024. However, it is intended that it be  
 132616 possible for an implementation to define the relationship between these cancellation

- 132617 cleanup handlers and Ada, C++, or other language-level exception handling facilities.
- 132618 It was suggested that the cancellation cleanup handlers should also be called when the  
132619 process exits or calls the *exec* function. This was rejected partly due to the performance  
132620 problem caused by having to call the cancellation cleanup handlers of every thread before  
132621 the operation could continue. The other reason was that the only state expected to be  
132622 cleaned up by the cancellation cleanup handlers would be the intraprocess state. Any  
132623 handlers that are to clean up the interprocess state would be registered with *atexit()*. There  
132624 is the orthogonal problem that the *exec* functions do not honor the *atexit()* handlers, but  
132625 resolving this is beyond the scope of POSIX.1-2024.
- 132626 • Async-Cancel Safety
- 132627 A function is said to be async-cancel-safe if it is written in such a way that entering the  
132628 function with asynchronous cancelability enabled will not cause any invariants to be  
132629 violated, even if a cancellation request is delivered at any arbitrary instruction. Functions  
132630 that are async-cancel-safe are often written in such a way that they need to acquire no  
132631 resources for their operation and the visible variables that they may write are strictly  
132632 limited.
- 132633 Any routine that gets a resource as a side-effect cannot be made async-cancel-safe (for  
132634 example, *malloc()*). If such a routine were called with asynchronous cancelability enabled,  
132635 it might acquire the resource successfully, but as it was returning to the client, it could act  
132636 on a cancellation request. In such a case, the application would have no way of knowing  
132637 whether the resource was acquired or not.
- 132638 Indeed, because many interesting routines cannot be made async-cancel-safe, most library  
132639 routines in general are not async-cancel-safe. Every library routine should specify whether  
132640 or not it is async-cancel safe so that programmers know which routines can be called from  
132641 code that is asynchronously cancelable.
- 132642 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/8 is applied, adding the *pselect()* function  
132643 to the list of functions with cancellation points.
- 132644 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/5 is applied, adding the *fdatasync()*  
132645 function into the table of functions that shall have cancellation points.
- 132646 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/6 is applied, adding the numerous  
132647 functions into the table of functions that may have cancellation points.
- 132648 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/7 is applied, clarifying the requirements  
132649 in Thread Cancellation Cleanup Handlers.
- 132650 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0023 [627], XSH/TC2-2008/0024  
132651 [627,632], XSH/TC2-2008/0025 [627], XSH/TC2-2008/0026 [632], and XSH/TC2-2008/0027  
132652 [622] are applied.
- 132653 Austin Group Defect 411 is applied, adding *accept4()* to the table of functions that shall have  
132654 cancellation points.
- 132655 Austin Group Defect 508 is applied, adding *ptsname()* and *ptsname\_r()* to the table of functions  
132656 that may have cancellation points.
- 132657 Austin Group Defect 614 is applied, adding *posix\_close()* to the table of functions that shall have  
132658 cancellation points.
- 132659 Austin Group Defect 697 is applied, adding *posix\_getdents()* to the table of functions that may  
132660 have cancellation points.
- 132661 Austin Group Defect 729 is applied, adding *posix\_devctl()* to the table of functions that may have



- 132662 cancellation points.
- 132663 Austin Group Defect 841 is applied, allowing `pthread_setcancelstate()` to be used to disable  
132664 cancellation in a signal catching function in order to avoid undefined behavior when the signal  
132665 is delivered during execution of a function that is not `async-cancel-safe`.
- 132666 Austin Group Defect 1076 is applied, moving `sem_wait()` and `sem_timedwait()` from the table of  
132667 functions that are required to have cancellation points to the table of functions that may have  
132668 cancellation points.
- 132669 Austin Group Defect 1122 is applied, adding `bindtextdomain()` and the `gettext` family of functions  
132670 to the table of functions that may have cancellation points.
- 132671 Austin Group Defect 1143 is applied, clarifying the conditions under which it is unspecified  
132672 whether the cancellation request is acted upon or whether the cancellation request remains  
132673 pending.
- 132674 Austin Group Defect 1216 is applied, adding `pthread_cond_clockwait()` to the table of functions  
132675 that are required to have cancellation points, and adding `pthread_rwlock_clockwrlock()`,  
132676 `pthread_rwlock_clockrdlock()`, and `sem_clockwait()` to the table of functions that may have  
132677 cancellation points.
- 132678 Austin Group Defect 1263 is applied, adding `ppoll()` to the table of functions that are required to  
132679 have cancellation points.
- 132680 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.
- 132681 Austin Group Defect 1410 is applied, removing the `asctime_r()` and `ctime_r()` functions.

132682 *B.2.9.6 Thread Read-Write Locks*

132683 **Background**

132684 Read-write locks are often used to allow parallel access to data on multi-processors, to avoid  
132685 context switches on uni-processors when multiple threads access the same data, and to protect  
132686 data structures that are frequently accessed (that is, read) but rarely updated (that is, written).  
132687 The in-core representation of a file system directory is a good example of such a data structure.  
132688 One would like to achieve as much concurrency as possible when searching directories, but limit  
132689 concurrent access when adding or deleting files.

132690 Although read-write locks can be implemented with mutexes and condition variables, such  
132691 implementations are significantly less efficient than is possible. Therefore, this synchronization  
132692 primitive is included in POSIX.1-2024 for the purpose of allowing more efficient  
132693 implementations in multi-processor systems.

132694 **Queuing of Waiting Threads**

132695 The `pthread_rwlock_unlock()` function description states that one writer or one or more readers  
132696 must acquire the lock if it is no longer held by any thread as a result of the call. However, the  
132697 function does not specify which thread(s) acquire the lock, unless the Thread Execution  
132698 Scheduling option is supported.

132699 The standard developers considered the issue of scheduling with respect to the queuing of  
132700 threads blocked on a read-write lock. The question turned out to be whether POSIX.1-2024  
132701 should require priority scheduling of read-write locks for threads whose execution scheduling  
132702 policy is priority-based (for example, `SCHED_FIFO` or `SCHED_RR`). There are tradeoffs  
132703 between priority scheduling, the amount of concurrency achievable among readers, and the  
132704 prevention of writer and/or reader starvation.

132705 For example, suppose one or more readers hold a read-write lock and the following threads  
132706 request the lock in the listed order:

132707 pthread\_rwlock\_wrlock() - Low priority thread writer\_a  
132708 pthread\_rwlock\_rdlock() - High priority thread reader\_a  
132709 pthread\_rwlock\_rdlock() - High priority thread reader\_b  
132710 pthread\_rwlock\_rdlock() - High priority thread reader\_c

132711 When the lock becomes available, should *writer\_a* block the high priority readers? Or, suppose a  
132712 read-write lock becomes available and the following are queued:

132713 pthread\_rwlock\_rdlock() - Low priority thread reader\_a  
132714 pthread\_rwlock\_rdlock() - Low priority thread reader\_b  
132715 pthread\_rwlock\_rdlock() - Low priority thread reader\_c  
132716 pthread\_rwlock\_wrlock() - Medium priority thread writer\_a  
132717 pthread\_rwlock\_rdlock() - High priority thread reader\_d

132718 If priority scheduling is applied then *reader\_d* would acquire the lock and *writer\_a* would block  
132719 the remaining readers. But should the remaining readers also acquire the lock to increase  
132720 concurrency? The solution adopted takes into account that when the Thread Execution  
132721 Scheduling option is supported, high priority threads may in fact starve low priority threads  
132722 (the application developer is responsible in this case for designing the system in such a way that  
132723 this starvation is avoided). Therefore, POSIX.1-2024 specifies that high priority readers take  
132724 precedence over lower priority writers. However, to prevent writer starvation from threads of  
132725 the same or lower priority, writers take precedence over readers of the same or lower priority.

132726 Priority inheritance mechanisms are non-trivial in the context of read-write locks. When a high  
132727 priority writer is forced to wait for multiple readers, for example, it is not clear which subset of  
132728 the readers should inherit the writer's priority. Furthermore, the internal data structures that  
132729 record the inheritance must be accessible to all readers, and this implies some sort of  
132730 serialization that could negate any gain in parallelism achieved through the use of multiple  
132731 readers in the first place. Finally, existing practice does not support the use of priority  
132732 inheritance for read-write locks. Therefore, no specification of priority inheritance or priority  
132733 ceiling is attempted. If reliable priority-scheduled synchronization is absolutely required, it can  
132734 always be obtained through the use of mutexes.

### 132735 **Comparison to fcntl() Locks**

132736 The read-write locks and the *fcntl()* locks in POSIX.1-2024 share a common goal: increasing  
132737 concurrency among readers, thus increasing throughput and decreasing delay.

132738 However, the read-write locks have two features not present in the *fcntl()* locks. First, under  
132739 priority scheduling, read-write locks are granted in priority order. Second, also under priority  
132740 scheduling, writer starvation is prevented by giving writers preference over readers of equal or  
132741 lower priority.

132742 Also, read-write locks can be used in systems lacking a file system, such as those conforming to  
132743 the minimal realtime system profile of IEEE Std 1003.13-1998.

132744 **History of Resolution Issues**

132745 Based upon some balloting objections, early drafts specified the behavior of threads waiting on a  
 132746 read-write lock during the execution of a signal handler, as if the thread had not called the lock  
 132747 operation. However, this specified behavior would require implementations to establish  
 132748 internal signal handlers even though this situation would be rare, or never happen for many  
 132749 programs. This would introduce an unacceptable performance hit in comparison to the little  
 132750 additional functionality gained. Therefore, the behavior of read-write locks and signals was  
 132751 reverted back to its previous mutex-like specification.

132752 *B.2.9.7 Thread Interactions with File Operations*

132753 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0028 [498] is applied.

132754 Austin Group Defect 411 is applied, adding *dup3()*.

132755 Austin Group Defect 695 is applied, extending the requirements in this section to non-regular  
 132756 files.

132757 *B.2.9.8 Use of Application-Managed Thread Stacks*

132758 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/8 is applied, adding this new section. It  
 132759 was added to make it clear that the current standard does not allow an application to determine  
 132760 when a stack can be reclaimed. This may be addressed in a future version.

132761 *B.2.9.9 Synchronization Object Copies and Alternative Mappings*

132762 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0029 [972] is applied.

132763 Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.

132764 **B.2.10 Sockets**

132765 The base document for the sockets interfaces in POSIX.1-2024 is the XNS, Issue 5.2 specification.  
 132766 This was primarily chosen as it aligns with IPv6. Additional material has been added from  
 132767 IEEE Std 1003.1g-2000, notably socket concepts, raw sockets, the *pselect()* function, the  
 132768 *socketmark()* function, and the `<sys/select.h>` header.

132769 *B.2.10.1 Address Families*

132770 There is no additional rationale provided for this section.

132771 *B.2.10.2 Addressing*

132772 There is no additional rationale provided for this section.

132773 *B.2.10.3 Protocols*

132774 There is no additional rationale provided for this section.

## 132775 B.2.10.4 Routing

132776 There is no additional rationale provided for this section.

## 132777 B.2.10.5 Interfaces

132778 There is no additional rationale provided for this section.

## 132779 B.2.10.6 Socket Types

132780 The type **socklen\_t** was invented to cover the range of implementations seen in the field. The  
132781 intent of **socklen\_t** is to be the type for all lengths that are naturally bounded in size; that is, that  
132782 they are the length of a buffer which cannot sensibly become of massive size: network addresses,  
132783 host names, string representations of these, ancillary data, control messages, and socket options  
132784 are examples. Truly boundless sizes are represented by **size\_t** as in *read()*, *write()*, and so on.

132785 All **socklen\_t** types were originally (in BSD UNIX) of type **int**. During the development of  
132786 POSIX.1-2024, it was decided to change all buffer lengths to **size\_t**, which appears at face value  
132787 to make sense. When dual mode 32/64-bit systems came along, this choice unnecessarily  
132788 complicated system interfaces because **size\_t** (with **long**) was a different size under ILP32 and  
132789 LP64 models. Reverting to **int** would have happened except that some implementations had  
132790 already shipped 64-bit-only interfaces. The compromise was a type which could be defined to be  
132791 any size by the implementation: **socklen\_t**.

## 132792 B.2.10.7 Socket I/O Mode

132793 There is no additional rationale provided for this section.

## 132794 B.2.10.8 Socket Owner

132795 There is no additional rationale provided for this section.

## 132796 B.2.10.9 Socket Queue Limits

132797 There is no additional rationale provided for this section.

## 132798 B.2.10.10 Pending Error

132799 There is no additional rationale provided for this section.

## 132800 B.2.10.11 Socket Receive Queue

132801 There is no additional rationale provided for this section.

## 132802 B.2.10.12 Socket Out-of-Band Data State

132803 There is no additional rationale provided for this section.

132804 B.2.10.13 *Connection Indication Queue*

132805 There is no additional rationale provided for this section.

132806 B.2.10.14 *Signals*

132807 There is no additional rationale provided for this section.

132808 B.2.10.15 *Asynchronous Errors*

132809 Austin Group Defect 1010 is applied, removing [EHOSTDOWN] from the list of asynchronous  
132810 errors.

132811 B.2.10.16 *Use of Options*

132812 Austin Group Defect 840 is applied, adding SO\_DOMAIN and SO\_PROTOCOL.

132813 Austin Group Defect 1337 is applied, clarifying socket option default values.

132814 B.2.10.17 *Use of Sockets for Local UNIX Connections*

132815 There is no additional rationale provided for this section.

132816 B.2.10.18 *Use of Sockets over Internet Protocols*

132817 A raw socket allows privileged users direct access to a protocol; for example, raw access to the  
132818 IP and ICMP protocols is possible through raw sockets. Raw sockets are intended for  
132819 knowledgeable applications that wish to take advantage of some protocol feature not directly  
132820 accessible through the other sockets interfaces.

132821 B.2.10.19 *Use of Sockets over Internet Protocols Based on IPv4*

132822 There is no additional rationale provided for this section.

132823 B.2.10.20 *Use of Sockets over Internet Protocols Based on IPv6*

132824 The Open Group Base Resolution bwg2001-012 is applied, clarifying that IPv6 implementations  
132825 are required to support use of AF\_INET6 sockets over IPv4.

132826 Austin Group Defect 411 is applied, adding *accept4()*.

132827 **B.2.11 Data Types**132828 B.2.11.1 *Defined Types*

132829 The requirement that additional types defined in this section end in ``\_t'' was prompted by the  
132830 problem of name space pollution. It is difficult to define a type (where that type is not one  
132831 defined by POSIX.1-2024) in one header file and use it in another without adding symbols to the  
132832 name space of the program. To allow implementors to provide their own types, all conforming  
132833 applications are required to avoid symbols ending in ``\_t'', which permits the implementor to

132834 provide additional types. Because a major use of types is in the definition of structure members,  
 132835 which can (and in many cases must) be added to the structures defined in POSIX.1-2024, the  
 132836 need for additional types is compelling.

132837 The types, such as **ushort** and **ulong**, which are in common usage, are not defined in  
 132838 POSIX.1-2024 (although **ushort\_t** would be permitted as an extension). They can be added to  
 132839 **<sys/types.h>** using a feature test macro (see [Section B.2.2.1](#), on page 3737). A suggested symbol  
 132840 for these is **\_SYSIII**. Similarly, the types like **u\_short** would probably be best controlled by **\_BSD**.

132841 Some of these symbols may appear in other headers; see [Section B.2.2.2](#) (on page 3738).

132842 **dev\_t** This type may be made large enough to accommodate host-locality considerations  
 132843 of networked systems.

132844 This type must be arithmetic. Earlier proposals allowed this to be non-arithmetic  
 132845 (such as a structure) and provided a *samefile()* function for comparison.

132846 **gid\_t** Some implementations had separated **gid\_t** from **uid\_t** before POSIX.1 was  
 132847 completed. It would be difficult for them to coalesce them when it was  
 132848 unnecessary. Additionally, it is quite possible that user IDs might be different from  
 132849 group IDs because the user ID might wish to span a heterogeneous network,  
 132850 where the group ID might not.

132851 For current implementations, the cost of having a separate **gid\_t** will be only  
 132852 lexical.

132853 **mode\_t** This type was chosen so that implementations could choose the appropriate  
 132854 integer type, and for compatibility with the ISO C standard. 4.3 BSD uses  
 132855 **unsigned short** and the SVID uses **ushort**, which is the same. Historically, only the  
 132856 low-order sixteen bits are significant.

132857 **nlink\_t** This type was introduced in place of **short** for *st\_nlink* (see the **<sys/stat.h>** header)  
 132858 in response to an objection that **short** was too small.

132859 **off\_t** This type is used to represent a file offset or file size. On systems supporting large  
 132860 files, **off\_t** is larger than 32 bits in at least one programming environment. Other  
 132861 programming environments may use different sizes for **off\_t**, for compatibility or  
 132862 other reasons.

132863 **pid\_t** The inclusion of this symbol was controversial because it is tied to the issue of the  
 132864 representation of a process ID as a number. From the point of view of a  
 132865 conforming application, process IDs should be “magic cookies”<sup>9</sup> that are produced  
 132866 by calls such as *fork()*, used by calls such as *waitpid()* or *kill()*, and not otherwise  
 132867 analyzed (except that the sign is used as a flag for certain operations).

132868 The concept of a {PID\_MAX} value interacted with this in early proposals. Treating  
 132869 process IDs as an opaque type both removes the requirement for {PID\_MAX} and  
 132870 allows systems to be more flexible in providing process IDs that span a large range  
 132871 of values, or a small one.

132872 Since the values in **uid\_t**, **gid\_t**, and **pid\_t** will be numbers generally, and  
 132873 potentially both large in magnitude and sparse, applications that are based on  
 132874 arrays of objects of this type are unlikely to be fully portable in any case. Solutions  
 132875 that treat them as magic cookies will be portable.

---

132876 9. An historical term meaning: “An opaque object, or token, of determinate size, whose significance is known only to the entity which  
 132877 created it. An entity receiving such a token from the generating entity may only make such use of the ‘cookie’ as is defined and permitted  
 132878 by the supplying entity.”

132879		{CHILD_MAX} precludes the possibility of a “toy implementation”, where there would only be one process.
132880		
132881	<b>ssize_t</b>	This is intended to be a signed analog of <b>size_t</b> . The wording is such that an implementation may either choose to use a longer type or simply to use the signed version of the type that underlies <b>size_t</b> . All functions that return <b>ssize_t</b> ( <i>read()</i> and <i>write()</i> ) describe as “implementation-defined” the result of an input exceeding {SSIZE_MAX}. It is recognized that some implementations might have <b>ints</b> that are smaller than <b>size_t</b> . A conforming application would be constrained not to perform I/O in pieces larger than {SSIZE_MAX}, but a conforming application using extensions would be able to use the full range if the implementation provided an extended range, while still having a single type-compatible interface.
132882		
132883		
132884		
132885		
132886		
132887		
132888		
132889		
132890		The symbols <b>size_t</b> and <b>ssize_t</b> are also required in <b>&lt;unistd.h&gt;</b> to minimize the changes needed for calls to <i>read()</i> and <i>write()</i> . Implementors are reminded that it must be possible to include both <b>&lt;sys/types.h&gt;</b> and <b>&lt;unistd.h&gt;</b> in the same program (in either order) without error.
132891		
132892		
132893		
132894	<b>uid_t</b>	Before the addition of this type, the data types used to represent these values varied throughout early proposals. The <b>&lt;sys/stat.h&gt;</b> header defined these values as type <b>short</b> , the <b>&lt;passwd.h&gt;</b> file (now <b>&lt;pwd.h&gt;</b> and <b>&lt;grp.h&gt;</b> ) used an <b>int</b> , and <i>getuid()</i> returned an <b>int</b> . In response to a strong objection to the inconsistent definitions, all the types were switched to <b>uid_t</b> .
132895		
132896		
132897		
132898		
132899		In practice, those historical implementations that use varying types of this sort can typedef <b>uid_t</b> to <b>short</b> with no serious consequences.
132900		
132901		The problem associated with this change concerns object compatibility after structure size changes. Since most implementations will define <b>uid_t</b> as a <b>short</b> , the only substantive change will be a reduction in the size of the <b>passwd</b> structure. Consequently, implementations with an overriding concern for object compatibility can pad the structure back to its current size. For that reason, this problem was not considered critical enough to warrant the addition of a separate type to POSIX.1.
132902		
132903		
132904		
132905		
132906		
132907		
132908		The types <b>uid_t</b> and <b>gid_t</b> are magic cookies. There is no {UID_MAX} defined by POSIX.1, and no structure imposed on <b>uid_t</b> and <b>gid_t</b> other than that they be positive arithmetic types. (In fact, they could be <b>unsigned char</b> .) There is no maximum or minimum specified for the number of distinct user or group IDs.
132909		
132910		
132911		
132912		POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0030 [733] is applied.
132913		Austin Group Defect 697 is applied, adding <b>reclen_t</b> .
132914		Austin Group Defect 1302 is applied, aligning this section with the ISO/IEC 9899:2018 standard.
132915	B.2.11.2	<i>The char Type</i>
132916		POSIX.1-2024 explicitly requires that a <b>char</b> type is exactly one byte (8 bits).

**132917 B.2.12 Status Information**

132918 POSIX.1-2024 does not require all matching WNOWAIT threads (threads in a matching call to  
132919 *waitid()* with the WNOWAIT flag set) to obtain a child's status information because the status  
132920 information might be discarded (consumed or replaced) before one of the matching WNOWAIT  
132921 threads is scheduled. If the status information is not discarded, it will remain available, so all of  
132922 the matching WNOWAIT threads will (eventually) obtain the status information.

132923 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0031 [690] is applied.

**132924 B.3 System Interfaces**

132925 See the RATIONALE sections on the individual reference pages.

**132926 B.3.1 System Interfaces Removed in this Version**

132927 This section contains a list of options and interfaces removed in POSIX.1-2024, together with  
132928 advice for application developers on the alternative interfaces that should be used.

**132929 B.3.1.1 STREAMS Option**

132930 Applications are recommended to use UNIX domain sockets as an alternative for much of the  
132931 functionality provided by this option. For example, file descriptor passing can be performed  
132932 using *sendmsg()* and *recvmsg()* with SCM\_RIGHTS on a UNIX domain socket instead of using  
132933 *ioctl()* with I\_SENDFD and I\_RECVFD on a STREAM.

**132934 B.3.1.2 Tracing Option**

132935 Applications are recommended to use implementation-provided extension interfaces instead of  
132936 the functionality provided by this option. (Such interfaces were in widespread use before the  
132937 Tracing option was added to POSIX.1 and continued to be used in preference to the Tracing  
132938 option interfaces.)

**132939 B.3.1.3 *\_longjmp()* and *\_setjmp()***

132940 Applications are recommended to use *siglongjmp()* and *sigsetjmp()* instead of these functions.

**132941 B.3.1.4 *\_tolower()* and *\_toupper()***

132942 Applications are recommended to use *tolower()* and *toupper()* instead of these functions.

**132943 B.3.1.5 *ftw()***

132944 Applications are recommended to use *nftw()* instead of this function.



- 132945 B.3.1.6 *getitimer()* and *setitimer()*
- 132946 Applications are recommended to use *timer\_gettime()* and *timer\_settime()* instead of these  
132947 functions.
- 132948 B.3.1.7 *gets()*
- 132949 Applications are recommended to use *fgets()* instead of this function.
- 132950 B.3.1.8 *gettimeofday()*
- 132951 Applications are recommended to use *clock\_gettime()* instead of this function.
- 132952 B.3.1.9 *isascii()* and *toascii()*
- 132953 Applications are recommended to use macros equivalent to the following instead of these  
132954 functions:
- ```
132955 #define isascii(c) (((c) & ~0177) == 0)
132956 #define toascii(c) ((c) & 0177)
```
- 132957 An alternative replacement for *isascii()*, depending on the intended outcome if the code is  
132958 ported to implementations with different character encodings, might be:
- ```
132959 #define isascii(c) (isprint((c)) || iscntrl((c)))
```
- 132960 (In the C or POSIX locale, this determines whether *c* is a character in the portable character set.)
- 132961 B.3.1.10 *pthread\_getconcurrency()* and *pthread\_setconcurrency()*
- 132962 Applications are recommended to use thread scheduling (on implementations that support the  
132963 Thread Execution Scheduling option) instead of these functions; see XSH [Section 2.9.4](#) (on page  
132964 540).
- 132965 B.3.1.11 *rand\_r()*
- 132966 Applications are recommended to use *nrand48()* or *random()* instead of this function.
- 132967 B.3.1.12 *setpggrp()*
- 132968 Applications are recommended to use *setpgid()* or *setsid()* instead of this function.
- 132969 B.3.1.13 *sighold()*, *sigpause()*, and *sigrelse()*
- 132970 Applications are recommended to use *pthread\_sigmask()* or *sigprocmask()* instead of these  
132971 functions.
- 132972 B.3.1.14 *sigignore()*, *siginterrupt()*, and *sigset()*
- 132973 Applications are recommended to use *sigtaction()* instead of these functions.

132974 B.3.1.15 *tempnam()*

132975 Applications are recommended to use *mkdtemp()*, *mkstemp()*, or *tmpfile()* instead of this  
132976 function.

132977 B.3.1.16 *ulimit()*

132978 Applications are recommended to use *getrlimit()* or *setrlimit()* instead of this function.

132979 B.3.1.17 *utime()*

132980 Applications are recommended to use *futimens()* if a file descriptor for the file is open, otherwise  
132981 *utimensat()*, instead of this function.

### 132982 B.3.2 System Interfaces Removed in the Previous Version

132983 The functions and symbols removed in Issue 7 (from the Issue 6 base document) were as  
132984 follows:

132985

Removed Functions and Symbols in Issue 7		
<i>bcmp()</i>	<i>gethostbyaddr()</i>	<i>rindex()</i>
<i>bcopy()</i>	<i>gethostbyname()</i>	<i>scalb()</i>
<i>bsd_signal()</i>	<i>getwd()</i>	<i>setcontext()</i>
<i>bzero()</i>	<i>h_errno</i>	<i>swapcontext()</i>
<i>ecvt()</i>	<i>index()</i>	<i>ualarm()</i>
<i>fcvt()</i>	<i>makecontext()</i>	<i>usleep()</i>
<i>ftime()</i>	<i>mktemp()</i>	<i>vfork()</i>
<i>gcvt()</i>	<i>pthread_attr_getstackaddr()</i>	<i>wcswcs()</i>
<i>getcontext()</i>	<i>pthread_attr_setstackaddr()</i>	

### 132995 B.3.3 Examples for Spawn

132996 The following long examples are provided in the Rationale (Informative) volume of  
132997 POSIX.1-2024 as a supplement to the reference page for *posix\_spawn()*.

#### 132998 Example Library Implementation of Spawn

132999 The *posix\_spawn()* or *posix\_spawnnp()* functions provide the following:

- 133000 • Simply start a process executing a process image. This is the simplest application for  
133001 process creation, and it may cover most executions of *fork()*.
- 133002 • Support I/O redirection, including pipes.
- 133003 • Run the child under a user and group ID in the domain of the parent.
- 133004 • Run the child at any priority in the domain of the parent.

133005 The *posix\_spawn()* or *posix\_spawnnp()* functions do not cover every possible use of the *fork()*  
133006 function, but they do span the common applications: typical use by a shell and a login utility.

133007 The price for an application is that before it calls *posix\_spawn()* or *posix\_spawnnp()*, the parent  
133008 must adjust to a state that *posix\_spawn()* or *posix\_spawnnp()* can map to the desired state for the  
133009 child. Environment changes require the parent to save some of its state and restore it afterwards.  
133010 The example below demonstrates an initial approach to implementing *posix\_spawn()* using other

```

133011     POSIX operations, although an actual implementation will need to be more robust at handling
133012     all possible filenames.

133013     #include <sys/types.h>
133014     #include <stdlib.h>
133015     #include <stdio.h>
133016     #include <unistd.h>
133017     #include <sched.h>
133018     #include <fcntl.h>
133019     #include <signal.h>
133020     #include <errno.h>
133021     #include <string.h>
133022     #include <signal.h>

133023     /* #include <spawn.h> */
133024     /*****
133025     /* Things that could be defined in spawn.h */
133026     /*****
133027     typedef struct
133028     {
133029         short posix_attr_flags;
133030         #define POSIX_SPAWN_SETPGROUP          0x1
133031         #define POSIX_SPAWN_SETSIGMASK       0x2
133032         #define POSIX_SPAWN_SETSIGDEF       0x4
133033         #define POSIX_SPAWN_SETSCHEDULER    0x8
133034         #define POSIX_SPAWN_SETSCHEDPARAM   0x10
133035         #define POSIX_SPAWN_RESETEIDS      0x20
133036         #define POSIX_SPAWN_SETSID        0x40
133037         pid_t posix_attr_pgroup;
133038         sigset_t posix_attr_sigmask;
133039         sigset_t posix_attr_sigdefault;
133040         int posix_attr_schedpolicy;
133041         struct sched_param posix_attr_schedparam;
133042     } posix_spawnattr_t;

133043     typedef char *posix_spawn_file_actions_t;

133044     int posix_spawn_file_actions_init(
133045         posix_spawn_file_actions_t *file_actions);
133046     int posix_spawn_file_actions_destroy(
133047         posix_spawn_file_actions_t *file_actions);
133048     int posix_spawn_file_actions_addchdir(
133049         posix_spawn_file_actions_t *restrict file_actions,
133050         const char *restrict path);
133051     int posix_spawn_file_actions_addclose(
133052         posix_spawn_file_actions_t *file_actions, int fildes);
133053     int posix_spawn_file_actions_adddup2(
133054         posix_spawn_file_actions_t *file_actions, int fildes,
133055         int newfildes);
133056     int posix_spawn_file_actions_addfchdir(
133057         posix_spawn_file_actions_t *file_actions, int fildes);
133058     int posix_spawn_file_actions_addopen(
133059         posix_spawn_file_actions_t *file_actions, int fildes,
133060         const char *path, int oflag, mode_t mode);
133061     int posix_spawnattr_init(posix_spawnattr_t *attr);

```

```

133062     int posix_spawnattr_destroy(posix_spawnattr_t *attr);
133063     int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
133064         short *lags);
133065     int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
133066     int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
133067         pid_t *pgroup);
133068     int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
133069     int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
133070         int *schedpolicy);
133071     int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
133072         int schedpolicy);
133073     int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
133074         struct sched_param *schedparam);
133075     int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
133076         const struct sched_param *schedparam);
133077     int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
133078         sigset_t *sigmask);
133079     int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
133080         const sigset_t *sigmask);
133081     int posix_spawnattr_getdefault(const posix_spawnattr_t *attr,
133082         sigset_t *sigdefault);
133083     int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
133084         const sigset_t *sigdefault);
133085     int posix_spawn(pid_t *pid, const char *path,
133086         const posix_spawn_file_actions_t *file_actions,
133087         const posix_spawnattr_t *attrp, char *const argv[],
133088         char *const envp[]);
133089     int posix_spawnnp(pid_t *pid, const char *file,
133090         const posix_spawn_file_actions_t *file_actions,
133091         const posix_spawnattr_t *attrp, char *const argv[],
133092         char *const envp[]);

133093     /*****
133094     /* Example posix_spawn() library routine */
133095     /*****
133096     int posix_spawn(pid_t *pid,
133097         const char *path,
133098         const posix_spawn_file_actions_t *file_actions,
133099         const posix_spawnattr_t *attrp,
133100         char *const argv[],
133101         char *const envp[])
133102     {
133103         /* Create process */
133104         if ((*pid = fork()) == (pid_t) 0)
133105         {
133106             /* This is the child process */
133107             /* Handle creating a new session */
133108             if (attrp->posix_attr_flags & POSIX_SPAWN_SETSID)
133109             {
133110                 /* Create a new session */
133111                 if (setsid() == -1)
133112                 {
133113                     /* Failed */

```

```

133114         _exit(127);
133115     }
133116 }
133117 /* Handle process group */
133118 if (attrp->posix_attr_flags & POSIX_SPAWN_SETPGROUP)
133119 {
133120     /* Override inherited process group */
133121     if (setpgid(0, attrp->posix_attr_pgroup) != 0)
133122     {
133123         /* Failed */
133124         _exit(127);
133125     }
133126 }
133127 /* Handle thread signal mask */
133128 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGMASK)
133129 {
133130     /* Set the signal mask (cannot fail) */
133131     sigprocmask(SIG_SETMASK, &attrp->posix_attr_sigmask, NULL);
133132 }
133133 /* Handle resetting effective user and group IDs */
133134 if (attrp->posix_attr_flags & POSIX_SPAWN_RESETEIDS)
133135 {
133136     /* None of these can fail for this case. */
133137     setuid(getuid());
133138     setgid(getgid());
133139 }
133140 /* Handle defaulted signals */
133141 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGDEF)
133142 {
133143     struct sigaction deflt;
133144     sigset_t all_signals;
133145
133146     int s;
133147
133148     /* Construct default signal action */
133149     deflt.sa_handler = SIG_DFL;
133150     deflt.sa_flags = 0;
133151
133152     /* Construct the set of all signals */
133153     sigfillset(&all_signals);
133154
133155     /* Loop for all signals */
133156     for (s = 0; sigismember(&all_signals, s); s++)
133157     {
133158         /* Signal to be defaulted? */
133159         if (sigismember(&attrp->posix_attr_sigdefault, s))
133160         {
133161             /* Yes; default this signal */
133162             if (sigaction(s, &deflt, NULL) == -1)
133163             {
133164                 /* Failed */
133165                 _exit(127);
133166             }
133167         }
133168     }

```

```

133163     }
133164     }
133165 }
133166 /* Handle the fds if they are to be mapped */
133167 if (file_actions != NULL)
133168 {
133169     /* Loop for all actions in object file_actions */
133170     /* (implementation dives beneath abstraction) */
133171     char *p = *file_actions;
133172     while (*p != '\0')
133173     {
133174         if (strncmp(p, "close(", 6) == 0)
133175         {
133176             int fd;
133177             if (sscanf(p + 6, "%d", &fd) != 1)
133178             {
133179                 _exit(127);
133180             }
133181             if (close(fd) == -1 && errno != EBADF)
133182                 _exit(127);
133183         }
133184         else if (strncmp(p, "dup2(", 5) == 0)
133185         {
133186             int fd, newfd;
133187             if (sscanf(p + 5, "%d,%d", &fd, &newfd) != 2)
133188             {
133189                 _exit(127);
133190             }
133191             if (fd == newfd)
133192             {
133193                 int flags = fcntl(fd, F_GETFD);
133194                 if (flags == -1)
133195                     _exit(127);
133196                 flags &= ~FD_CLOEXEC;
133197                 if (fcntl(fd, F_SETFD, flags) == -1)
133198                     _exit(127);
133199             }
133200             else if (dup2(fd, newfd) == -1)
133201                 _exit(127);
133202         }
133203         else if (strncmp(p, "open(", 5) == 0)
133204         {
133205             int fd, oflag;
133206             mode_t mode;
133207             int tempfd;
133208             char path[1000];    /* Should be dynamic */
133209             char *q;
133210             if (sscanf(p + 5, "%d,", &fd) != 1)
133211             {
133212                 _exit(127);

```

```

133213     }
133214     p = strchr(p, ',') + 1;
133215     q = strchr(p, '*');
133216     if (q == NULL)
133217         _exit(127);
133218     strncpy(path, p, q - p);
133219     path[q - p] = '\0';
133220     if (sscanf(q + 1, "%o,%o", &oflag, &mode) != 2)
133221     {
133222         _exit(127);
133223     }
133224     if (close(fd) == -1)
133225     {
133226         if (errno != EBADF)
133227             _exit(127);
133228     }
133229     tempfd = open(path, oflag, mode);
133230     if (tempfd == -1)
133231         _exit(127);
133232     if (tempfd != fd)
133233     {
133234         if (dup2(tempfd, fd) == -1)
133235         {
133236             _exit(127);
133237         }
133238         if (close(tempfd) == -1)
133239         {
133240             _exit(127);
133241         }
133242     }
133243 }
133244 else if (strncmp(p, "chdir(", 6) == 0)
133245 {
133246     char path[1000]; /* Should be dynamic */
133247     char *q;

133248     p += 6
133249     q = strchr(p, '*');
133250     if (q == NULL)
133251         _exit(127);
133252     strncpy(path, p, q - p);
133253     path[q - p] = '\0';
133254     if (chdir(path) == -1)
133255         _exit(127);
133256 }
133257 else if (strncmp(p, "fchdir(", 7) == 0)
133258 {
133259     int fd;

133260     if (sscanf(p + 7, "%d", &fd) != 1)
133261         _exit(127);
133262     if (fchdir(fd) == -1)
133263         _exit(127);

```

```

133264         }
133265         else
133266         {
133267             _exit(127);
133268         }
133269         p = strchr(p, ' ') + 1;
133270     }
133271 }

133272 /* Handle setting new scheduling policy and parameters */
133273 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDULER)
133274 {
133275     if (sched_setscheduler(0, attrp->posix_attr_schedpolicy,
133276         &attrp->posix_attr_schedparam) == -1)
133277     {
133278         _exit(127);
133279     }
133280 }

133281 /* Handle setting only new scheduling parameters */
133282 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDPARAM)
133283 {
133284     if (sched_setparam(0, &attrp->posix_attr_schedparam) == -1)
133285     {
133286         _exit(127);
133287     }
133288 }

133289 /* Now execute the program at path */
133290 /* Any fd that still has FD_CLOEXEC set will be closed */
133291 execve(path, argv, envp);
133292 _exit(127);          /* exec failed */
133293 }
133294 else
133295 {
133296     /* This is the parent (calling) process */
133297     if (*pid == (pid_t) - 1)
133298         return errno;
133299     return 0;
133300 }
133301 }

133302 /*****
133303 /* Here is a crude but effective implementation of the */
133304 /* file action object operators which store actions as */
133305 /* concatenated token-separated strings.          */
133306 /*****
133307 /* Create object with no actions. */
133308 int posix_spawn_file_actions_init(
133309     posix_spawn_file_actions_t *file_actions)
133310 {
133311     *file_actions = malloc(sizeof(char));
133312     if (*file_actions == NULL)
133313         return ENOMEM;
133314     strcpy(*file_actions, "");

```



```
133315         return 0;
133316     }
133317     /* Free object storage and make invalid. */
133318     int posix_spawn_file_actions_destroy(
133319         posix_spawn_file_actions_t *file_actions)
133320     {
133321         free(*file_actions);
133322         *file_actions = NULL;
133323         return 0;
133324     }
133325     /* Add a new action string to object. */
133326     static int add_to_file_actions(
133327         posix_spawn_file_actions_t *file_actions, char *new_action)
133328     {
133329         *file_actions = realloc
133330             (*file_actions, strlen(*file_actions) + strlen(new_action) + 1);
133331         if (*file_actions == NULL)
133332             return ENOMEM;
133333         strcat(*file_actions, new_action);
133334         return 0;
133335     }
133336     /* Add a chdir action to object. */
133337     int posix_spawn_file_actions_addchdir(
133338         posix_spawn_file_actions_t *restrict file_actions,
133339         const char *restrict path)
133340     {
133341         char temp[100];
133342         sprintf(temp, "chdir(%s)", path);
133343         return add_to_file_actions(file_actions, temp);
133344     }
133345     /* Add a close action to object. */
133346     int posix_spawn_file_actions_addclose(
133347         posix_spawn_file_actions_t *file_actions, int fildes)
133348     {
133349         char temp[100];
133350         sprintf(temp, "close(%d)", fildes);
133351         return add_to_file_actions(file_actions, temp);
133352     }
133353     /* Add a dup2 action to object. */
133354     int posix_spawn_file_actions_adddup2(
133355         posix_spawn_file_actions_t *file_actions, int fildes,
133356         int newfildes)
133357     {
133358         char temp[100];
133359         sprintf(temp, "dup2(%d,%d)", fildes, newfildes);
133360         return add_to_file_actions(file_actions, temp);
133361     }
133362     /* Add a fchdir action to object. */
```

```

133363     int posix_spawn_file_actions_addfchdir(
133364         posix_spawn_file_actions_t *file_actions, int fildes)
133365     {
133366         char temp[100];
133367         sprintf(temp, "fchdir(%d)", fildes);
133368         return add_to_file_actions(file_actions, temp);
133369     }
133370     /* Add an open action to object. */
133371     int posix_spawn_file_actions_addopen(
133372         posix_spawn_file_actions_t *file_actions, int fildes,
133373         const char *path, int oflag, mode_t mode)
133374     {
133375         char temp[100];
133376         sprintf(temp, "open(%d,%s*%o,%o)", fildes, path, oflag, mode);
133377         return add_to_file_actions(file_actions, temp);
133378     }
133379     /******
133380     /* Here is a crude but effective implementation of the */
133381     /* spawn attributes object functions which manipulate */
133382     /* the individual attributes. */
133383     /******
133384     /* Initialize object with default values. */
133385     int posix_spawnattr_init(posix_spawnattr_t *attr)
133386     {
133387         attr->posix_attr_flags = 0;
133388         attr->posix_attr_pgroup = 0;
133389         /* Default value of signal mask is the parent's signal mask; */
133390         /* other values are also allowed */
133391         sigprocmask(0, NULL, &attr->posix_attr_sigmask);
133392         sigemptyset(&attr->posix_attr_sigdefault);
133393         /* Default values of scheduling attr inherited from the parent; */
133394         /* other values are also allowed */
133395         attr->posix_attr_schedpolicy = sched_getscheduler(0);
133396         sched_getparam(0, &attr->posix_attr_schedparam);
133397         return 0;
133398     }
133399     int posix_spawnattr_destroy(posix_spawnattr_t *attr)
133400     {
133401         /* No action needed */
133402         return 0;
133403     }
133404     int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
133405         short *flags)
133406     {
133407         *flags = attr->posix_attr_flags;
133408         return 0;
133409     }
133410     int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags)
133411     {

```

```
133412     attr->posix_attr_flags = flags;
133413     return 0;
133414 }

133415 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
133416 pid_t *pgroup)
133417 {
133418     *pgroup = attr->posix_attr_pgroup;
133419     return 0;
133420 }

133421 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup)
133422 {
133423     attr->posix_attr_pgroup = pgroup;
133424     return 0;
133425 }

133426 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
133427 int *schedpolicy)
133428 {
133429     *schedpolicy = attr->posix_attr_schedpolicy;
133430     return 0;
133431 }

133432 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
133433 int schedpolicy)
133434 {
133435     attr->posix_attr_schedpolicy = schedpolicy;
133436     return 0;
133437 }

133438 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
133439 struct sched_param *schedparam)
133440 {
133441     *schedparam = attr->posix_attr_schedparam;
133442     return 0;
133443 }

133444 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
133445 const struct sched_param *schedparam)
133446 {
133447     attr->posix_attr_schedparam = *schedparam;
133448     return 0;
133449 }

133450 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
133451 sigset_t *sigmask)
133452 {
133453     *sigmask = attr->posix_attr_sigmask;
133454     return 0;
133455 }

133456 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
133457 const sigset_t *sigmask)
133458 {
133459     attr->posix_attr_sigmask = *sigmask;
133460     return 0;
```

```

133461     }
133462     int posix_spawnattr_getsigdefault(const posix_spawnattr_t *attr,
133463         sigset_t *sigdefault)
133464     {
133465         *sigdefault = attr->posix_attr_sigdefault;
133466         return 0;
133467     }
133468     int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
133469         const sigset_t *sigdefault)
133470     {
133471         attr->posix_attr_sigdefault = *sigdefault;
133472         return 0;
133473     }

```

### 133474 **I/O Redirection with Spawn**

133475 I/O redirection with *posix\_spawn()* or *posix\_spawnnp()* is accomplished by crafting a *file\_actions*  
133476 argument to effect the desired redirection. Such a redirection follows the general outline of the  
133477 following example:

```

133478     /* To redirect new standard output (fd 1) to a file, */
133479     /* and redirect new standard input (fd 0) from my fd socket_pair[1], */
133480     /* and close my fd socket_pair[0] in the new process. */
133481     posix_spawn_file_actions_t file_actions;
133482     posix_spawn_file_actions_init(&file_actions);
133483     posix_spawn_file_actions_addopen(&file_actions, 1, "newout", ...);
133484     posix_spawn_file_actions_dup2(&file_actions, socket_pair[1], 0);
133485     posix_spawn_file_actions_close(&file_actions, socket_pair[0]);
133486     posix_spawn_file_actions_close(&file_actions, socket_pair[1]);
133487     posix_spawn(..., &file_actions, ...);
133488     posix_spawn_file_actions_destroy(&file_actions);

```

### 133489 **Spawning a Process Under a New User ID**


133490 Spawning a process under a new user ID follows the outline shown in the following example:

```

133491     Save = getuid();
133492     setuid(newid);
133493     posix_spawn(...);
133494     setuid(Save);

```

133495

 *Rationale (Informative)*

133496

**Part C:**

133497

**Shell and Utilities**

133498

*The Open Group*

133499

*The Institute of Electrical and Electronics Engineers, Inc.*



133500

133501

# Rationale for Shell and Utilities

## C.1 Introduction

### C.1.1 Change History

The change history is provided as an informative section, to track changes from earlier versions of this standard.

The following sections describe changes made to the Shell and Utilities volume of POSIX.1-2024 since Issue 7 of the base document. The CHANGE HISTORY section for each utility describes technical changes made to that utility in Issue 5 and later. Changes made before Issue 5 are not included.

#### Changes from Issue 7 to Issue 8 (POSIX.1-2024)

The following list summarizes the major changes that were made in the Shell and Utilities volume of POSIX.1-2024 from Issue 7 to Issue 8:

- The Open Group Standard, 2022, Additional APIs for the Base Specifications Issue 8, Part 2 is incorporated.
- Austin Group defect reports and IEEE Interpretations against IEEE Std 1003.1 are applied.
- The Open Group corrigenda and resolutions are applied.
- Features, marked obsolescent in the base document, have been considered for removal in this version.

#### New Features in Issue 8

The utilities first introduced in Issue 8 (over the Issue 7 base document) are as follows:

133521

133522

133523

133524

New Utilities in Issue 8		
<i>gettext</i>	<i>readlink</i>	<i>xgettext</i>
<i>msgfmt</i>	<i>realpath</i>	
<i>ngettext</i>	<i>timeout</i>	

#### Removed Utilities in Issue 8

The utilities removed in Issue 8 (from the Issue 7 base document) are as follows:

133527

133528

133529

133530

133531

Removed Utilities in Issue 8		
<i>fort77</i>	<i>qmove</i>	<i>qselect</i>
<i>qalter</i>	<i>qmsg</i>	<i>qsig</i>
<i>qdel</i>	<i>qrerun</i>	<i>qstat</i>
<i>qhold</i>	<i>qrls</i>	<i>qsub</i>

133532 **C.1.2 Relationship to Other Documents**133533 *C.1.2.1 System Interfaces*

133534 It has been pointed out that the Shell and Utilities volume of POSIX.1-2024 assumes that a great  
 133535 deal of functionality from the System Interfaces volume of POSIX.1-2024 is present, but never  
 133536 states exactly how much (and strictly does not need to since both are mandated on a conforming  
 133537 system). This section is an attempt to clarify the assumptions.

133538 **File Read, Write, and Creation**

133539 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/2 is applied, updating Table 1-1.

133540 **File Removal**

133541 This is intended to be a summary of the *unlink()* and *rmdir()* requirements. Note that it is  
 133542 possible using the *unlink()* function for item 4. to occur.

133543 *C.1.2.2 Concepts Derived from the ISO C Standard*

133544 This section was introduced to address the issue that there was insufficient detail presented by  
 133545 such utilities as *awk* or *sh* about their procedural control statements and their methods of  
 133546 performing arithmetic functions.

133547 The ISO C standard was selected as a model because most historical implementations of the  
 133548 standard utilities were written in C. Thus, it was more likely that they would act in the desired  
 133549 manner without modification.

133550 Using the ISO C standard is primarily a notational convenience so that the many procedural  
 133551 languages in the Shell and Utilities volume of POSIX.1-2024 would not have to be rigorously  
 133552 described in every aspect. Its selection does not require that the standard utilities be written in  
 133553 Standard C; they could be written in Common Usage C, Ada, Pascal, assembler language, or  
 133554 anything else.

133555 The sizes of the various numeric values refer to C-language data types that are allowed to be  
 133556 different sizes by the ISO C standard. Thus, like a C-language application, a shell application  
 133557 cannot rely on their exact size. However, it can rely on their minimum sizes expressed in the  
 133558 ISO C standard, such as {LONG\_MAX} for a **long** type.

133559 The behavior on overflow is undefined for ISO C standard arithmetic. Therefore, the standard  
 133560 utilities can use “bignum” representation for integers so that there is no fixed maximum unless  
 133561 otherwise stated in the utility description. Similarly, standard utilities can use infinite-precision  
 133562 representations for floating-point arithmetic, as long as these representations exceed the ISO C  
 133563 standard requirements.

133564 This section addresses only the issue of semantics; it is not intended to specify syntax. For  
 133565 example, the ISO C standard requires that 0L be recognized as an integer constant equal to zero,  
 133566 but utilities such as *awk* and *sh* are not required to recognize 0L (though they are allowed to, as  
 133567 an extension).

133568 The ISO C standard requires that a C compiler must issue a diagnostic for constants that are too  
 133569 large to represent. Most standard utilities are not required to issue these diagnostics; for  
 133570 example, the command:

133571 `diff -C 2147483648 file1 file2`



133572 has undefined behavior, and the *diff* utility is not required to issue a diagnostic even if the  
 133573 number 2 147 483 648 cannot be represented.  
 133574 Austin Group Defect 1128 is applied, adding a note about the comma operator.

### 133575 C.1.3 Utility Limits

133576 This section grew out of an idea that originated with the original POSIX.1, in the tables of system  
 133577 limits for the *sysconf*() and *pathconf*() functions. The idea being that a conforming application  
 133578 can be written to use the most restrictive values that a minimal system can provide, but it should  
 133579 not have to. The values provided represent compromises so that some vendors can use  
 133580 historically limited versions of UNIX system utilities. They are the highest values that a strictly  
 133581 conforming application can assume, given no other information.

133582 However, by using the *getconf* utility or the *sysconf*() function, the elegant application can be  
 133583 tailored to more liberal values on some of the specific instances of specific implementations.

133584 There is no explicitly stated requirement that an implementation provide finite limits for any of  
 133585 these numeric values; the implementation is free to provide essentially unbounded capabilities  
 133586 (where it makes sense), stopping only at reasonable points such as {ULONG\_MAX} (from the  
 133587 ISO C standard). Therefore, applications desiring to tailor themselves to the values on a  
 133588 particular implementation need to be ready for possibly huge values; it may not be a good idea  
 133589 to allocate blindly a buffer for an input line based on the value of {LINE\_MAX}, for instance.  
 133590 However, unlike the System Interfaces volume of POSIX.1-2024, there is no set of limits that  
 133591 return a special indication meaning “unbounded”. The implementation should always return an  
 133592 actual number, even if the number is very large.

133593 The statement:

133594 “It is not guaranteed that the application ...”

133595 is an indication that many of these limits are designed to ensure that implementors design their  
 133596 utilities without arbitrary constraints related to unimaginative programming. There are certainly  
 133597 conditions under which combinations of options can cause failures that would not render an  
 133598 implementation non-conforming. For example, {EXPR\_NEST\_MAX} and {ARG\_MAX} could  
 133599 collide when expressions are large; combinations of {BC\_SCALE\_MAX} and {BC\_DIM\_MAX}  
 133600 could exceed virtual memory.

133601 In the Shell and Utilities volume of POSIX.1-2024, the notion of a limit being guaranteed for the  
 133602 process lifetime, as it is in the System Interfaces volume of POSIX.1-2024, is not as useful to a  
 133603 shell script. The *getconf* utility is probably a process itself, so the guarantee would be without  
 133604 value. Therefore, the Shell and Utilities volume of POSIX.1-2024 requires the guarantee to be for  
 133605 the session lifetime. This will mean that many vendors will either return very conservative  
 133606 values or possibly implement *getconf* as a built-in.

133607 It may seem confusing to have limits that apply only to a single utility grouped into one global  
 133608 section. However, the alternative, which would be to disperse them out into their utility  
 133609 description sections, would cause great difficulty when *sysconf*() and *getconf* were described.  
 133610 Therefore, the standard developers chose the global approach.

133611 Each language binding could provide symbol names that are slightly different from those shown  
 133612 here. For example, the C-Language Binding option adds a leading <underscore> to the symbols  
 133613 as a prefix.

133614 The following comments describe selection criteria for the symbols and their values:

- 133615 {ARG\_MAX}  
 133616 This is defined by the System Interfaces volume of POSIX.1-2024. Unfortunately, it is very  
 133617 difficult for a conforming application to deal with this value, as it does not know how much  
 133618 of its argument space is being consumed by the environment variables of the user.
- 133619 {BC\_BASE\_MAX}  
 133620 {BC\_DIM\_MAX}  
 133621 {BC\_SCALE\_MAX}  
 133622 These were originally one value, {BC\_SCALE\_MAX}, but it was unreasonable to link all  
 133623 three concepts into one limit.
- 133624 {CHILD\_MAX}  
 133625 This is defined by the System Interfaces volume of POSIX.1-2024.
- 133626 {COLL\_WEIGHTS\_MAX}  
 133627 The weights assigned to **order** can be considered as “passes” through the collation  
 133628 algorithm.
- 133629 {EXPR\_NEST\_MAX}  
 133630 The value for expression nesting was borrowed from the ISO C standard.
- 133631 {LINE\_MAX}  
 133632 This is a global limit that affects all utilities, unless otherwise noted. The {MAX\_CANON}  
 133633 value from the System Interfaces volume of POSIX.1-2024 may further limit input lines from  
 133634 terminals. The {LINE\_MAX} value was the subject of much debate and is a compromise  
 133635 between those who wished to have unlimited lines and those who understood that many  
 133636 historical utilities were written with fixed buffers. Frequently, utility writers selected the  
 133637 UNIX system constant BUFSIZ to allocate these buffers; therefore, some utilities were  
 133638 limited to 512 bytes for I/O lines, while others achieved 4 096 bytes or greater.
- 133639 It should be noted that {LINE\_MAX} applies only to input line length; there is no  
 133640 requirement in POSIX.1-2024 that limits the length of output lines. Utilities such as *awk*, *sed*,  
 133641 and *paste* could theoretically construct lines longer than any of the input lines they received,  
 133642 depending on the options used or the instructions from the application. They are not  
 133643 required to truncate their output to {LINE\_MAX}. It is the responsibility of the application  
 133644 to deal with this. If the output of one of those utilities is to be piped into another of the  
 133645 standard utilities, line length restrictions will have to be considered; the *fold* utility, among  
 133646 others, could be used to ensure that only reasonable line lengths reach utilities or  
 133647 applications.
- 133648 {LINK\_MAX}  
 133649 This is defined by the System Interfaces volume of POSIX.1-2024.
- 133650 {MAX\_CANON}  
 133651 {MAX\_INPUT}  
 133652 {NAME\_MAX}  
 133653 {NGROUPS\_MAX}  
 133654 {OPEN\_MAX}  
 133655 {PATH\_MAX}  
 133656 {PIPE\_BUF}  
 133657 These limits are defined by the System Interfaces volume of POSIX.1-2024. Note that the  
 133658 byte lengths described by some of these values continue to represent bytes, even if the  
 133659 applicable character set uses a multi-byte encoding.
- 133660 {RE\_DUP\_MAX}  
 133661 The value selected is consistent with historical practice. Although the name implies that it  
 133662 applies to all REs, only BREs use the interval notation  $\{m,n\}$  addressed by this limit.

133663 {POSIX2\_SYMLINKS}

133664 The {POSIX2\_SYMLINKS} variable indicates that the underlying operating system supports  
 133665 the creation of symbolic links in specific directories. Many of the utilities defined in  
 133666 POSIX.1-2024 that deal with symbolic links do not depend on this value. For example, a  
 133667 utility that follows symbolic links (or does not, as the case may be) will only be affected by a  
 133668 symbolic link if it encounters one. Presumably, a file system that does not support symbolic  
 133669 links will not contain any. This variable does affect such utilities as *ln -s* and *pax* that  
 133670 attempt to create symbolic links.

133671 There are different limits associated with command lines and input to utilities, depending on the  
 133672 method of invocation. In the case of a C program *exec*-ing a utility, {ARG\_MAX} is the  
 133673 underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE\_MAX}  
 133674 limits the length of lines the shell is required to process, and {ARG\_MAX} will still be a limit. If a  
 133675 user is entering a command on a terminal to the shell, requesting that it invoke the utility,  
 133676 {MAX\_INPUT} may restrict the length of the line that can be given to the shell to a value below  
 133677 {LINE\_MAX}.

133678 When an option is supported, *getconf* returns a value of 1. For example, when C development is  
 133679 supported:

```
133680 if [ "$(getconf POSIX2_C_DEV)" -eq 1 ]; then
133681     echo C supported
133682 fi
```

133683 The *sysconf()* function in the C-Language Binding option would return 1.

133684 The following comments describe selection criteria for the symbols and their values:

```
133685 POSIX2_C_BIND
133686 POSIX2_C_DEV
133687 POSIX2_FORT_RUN
133688 POSIX2_SW_DEV
133689 POSIX2_UPE
```

133690 It is possible for some (usually privileged) operations to remove utilities that support these  
 133691 options or otherwise to render these options unsupported. The header files, the *sysconf()*  
 133692 function, or the *getconf* utility will not necessarily detect such actions, in which case they  
 133693 should not be considered as rendering the implementation non-conforming. A test suite  
 133694 should not attempt tests such as:

```
133695 rm /usr/bin/c17
133696 getconf POSIX2_C_DEV
```

133697 POSIX2\_LOCALEDEF

133698 This symbol was introduced to allow implementations to restrict supported locales to only  
 133699 those supplied by the implementation.

133700 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/2 is applied, deleting the entry for  
 133701 {POSIX2\_VERSION} since it is not a utility limit minimum value.

133702 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/3 is applied, changing the text in Utility  
 133703 Limits from: “utility (see *getconf*) through the *sysconf()* function defined in the System Interfaces  
 133704 volume of POSIX.1-2024. The literal names shown in Table 1-3 apply only to the *getconf* utility;  
 133705 the high-level language binding describes the exact form of each name to be used by the  
 133706 interfaces in that binding.” to: “utility (see *getconf*).”.

133707 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0001 [666] is applied.

133708 **C.1.4 Grammar Conventions**

133709 There is no additional rationale provided for this section.

133710 **C.1.5 Utility Description Defaults**133711 This section is arranged with headings in the same order as all the utility descriptions. It is a  
133712 collection of related and unrelated information concerning:

- 133713 1. The default actions of utilities
- 133714 2. The meanings of notations used in POSIX.1-2024 that are specific to individual utility  
133715 sections

133716 Although this material may seem out of place here, it is important that this information appear  
133717 before any of the utilities to be described later.133718 **NAME**

133719 There is no additional rationale provided for this section.

133720 **SYNOPSIS**

133721 There is no additional rationale provided for this section.

133722 **DESCRIPTION**

133723 Austin Group Defect 351 is applied, adding a requirement relating to declaration utilities.

133724 **OPTIONS**133725 Although it has not always been possible, the standard developers tried to avoid repeating  
133726 information to reduce the risk that duplicate explanations could each be modified differently.133727 The need to recognize `--` is required because conforming applications need to shield their  
133728 operands from any arbitrary options that the implementation may provide as an extension. For  
133729 example, if the standard utility *foo* is listed as taking no options, and the application needed to  
133730 give it a pathname with a leading `<hyphen-minus>`, it could safely do it as:133731 `foo -- -myfile`133732 and avoid any problems with `-m` used as an extension.

133733 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0002 [584] is applied.

133734 **OPERANDS**133735 The usage of `-` is never shown in the SYNOPSIS. Similarly, the usage of `--` is never shown.133736 The requirement for processing operands in command-line order is to avoid a “WeirdNIX”  
133737 utility that might choose to sort the input files alphabetically, by size, or by directory order.  
133738 Although this might be acceptable for some utilities, in general the programmer has a right to  
133739 know exactly what order will be chosen.133740 Some of the standard utilities take multiple *file* operands and act as if they were processing the  
133741 concatenation of those files. For example:133742 `asa file1 file2`

133743 and:

133744 `cat file1 file2 | asa`

133745 have similar results when questions of file access, errors, and performance are ignored. Other  
 133746 utilities such as *grep* or *wc* have completely different results in these two cases. This latter type of  
 133747 utility is always identified in its DESCRIPTION or OPERANDS sections, whereas the former is  
 133748 not. Although it might be possible to create a general assertion about the former case, the  
 133749 following points must be addressed:

- 133750 • Access times for the files might be different in the operand case *versus* the *cat* case.
- 133751 • The utility may have error messages that are cognizant of the input filename, and this  
 133752 added value should not be suppressed. (As an example, *awk* sets a variable with the  
 133753 filename at each file boundary.)

## 133754 **STDIN**

133755 There is no additional rationale provided for this section.

## 133756 **INPUT FILES**

133757 A conforming application cannot assume the following three commands are equivalent:

```
133758 tail -n +2 file
133759 (sed -n 1q; cat) < file
133760 cat file | (sed -n 1q; cat)
```

133761 The second command is equivalent to the first only when the file is seekable. In the third  
 133762 command, if the file offset in the open file description were not unspecified, *sed* would have to  
 133763 be implemented so that it read from the pipe 1 byte at a time or it would have to employ some  
 133764 method to seek backwards on the pipe. Such functionality is not defined currently in POSIX.1  
 133765 and does not exist on all historical systems. Other utilities, such as *head*, *read*, and *sh*, have similar  
 133766 properties, so the restriction is described globally in this section.

133767 The definition of “text file” is strictly enforced for input to the standard utilities; very few of  
 133768 them list exceptions to the undefined results called for here. (Of course, “undefined” here does  
 133769 not mean that historical implementations necessarily have to change to start indicating error  
 133770 conditions. Conforming applications cannot rely on implementations succeeding or failing when  
 133771 non-text files are used.)

133772 The utilities that allow line continuation are generally those that accept input languages, rather  
 133773 than pure data. It would be unusual for an input line of this type to exceed {LINE\_MAX} bytes  
 133774 and unreasonable to require that the implementation allow unlimited accumulation of multiple  
 133775 lines, each of which could reach {LINE\_MAX}. Thus, for a conforming application the total of all  
 133776 the continued lines in a set cannot exceed {LINE\_MAX}.

133777 The format description is intended to be sufficiently rigorous to allow other applications to  
 133778 generate these input files. However, since <blank> characters can legitimately be included in  
 133779 some of the fields described by the standard utilities, particularly in locales other than the POSIX  
 133780 locale, this intent is not always realized.

**133781 ENVIRONMENT VARIABLES**

133782 There is no additional rationale provided for this section.

**133783 ASYNCHRONOUS EVENTS**

133784 Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some  
133785 additional processing (such as deleting temporary files), restore the default signal action, and  
133786 resignal itself.

133787 Austin Group Defects 1648 and 1772 are applied, clarifying the default behavior for signal  
133788 handling.

**133789 STDOUT**

133790 The format description is intended to be sufficiently rigorous to allow post-processing of output  
133791 by other programs, particularly by an *awk* or *lex* parser.

**133792 STDERR**

133793 This section does not describe error messages that refer to incorrect operation of the utility.  
133794 Consider a utility that processes program source code as its input. This section is used to  
133795 describe messages produced by a correctly operating utility that encounters an error in the  
133796 program source code on which it is processing. However, a message indicating that the utility  
133797 had insufficient memory in which to operate would not be described.

133798 Some utilities have traditionally produced warning messages without returning a non-zero exit  
133799 status; these are specifically noted in their sections. Other utilities shall not write to standard  
133800 error if they complete successfully, unless the implementation provides some sort of extension to  
133801 increase the verbosity or debugging level.

133802 The format descriptions are intended to be sufficiently rigorous to allow post-processing of  
133803 output by other programs.

**133804 OUTPUT FILES**

133805 The format description is intended to be sufficiently rigorous to allow post-processing of output  
133806 by other programs, particularly by an *awk* or *lex* parser.

133807 Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging  
133808 mode) that would bypass any attempted recovery actions.

**133809 EXTENDED DESCRIPTION**

133810 There is no additional rationale provided for this section.

**133811 EXIT STATUS**

133812 Note the additional discussion of exit values in *Exit Status for Commands* in the *sh* utility. It  
133813 describes requirements for returning exit values greater than 125.

133814 A utility may list zero as a successful return, 1 as a failure for a specific reason, and greater than  
133815 1 as "an error occurred". In this case, unspecified conditions may cause a 2 or 3, or other value,  
133816 to be returned. A strictly conforming application should be written so that it tests for successful  
133817 exit status values (zero in this case), rather than relying upon the single specific error value listed  
133818 in POSIX.1-2024. In that way, it will have maximum portability, even on implementations with  
133819 extensions.

133820 The standard developers are aware that the general non-enumeration of errors makes it difficult

133821 to write test suites that test the *incorrect* operation of utilities. There are some historical  
133822 implementations that have expended effort to provide detailed status messages and a helpful  
133823 environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant  
133824 syntax errors; other implementations have not. Since there is no realistic way to mandate system  
133825 behavior in cases of undefined application actions or system problems—in a manner acceptable  
133826 to all cultures and environments—attention has been limited to the correct operation of utilities  
133827 by the conforming application. Furthermore, the conforming application does not need detailed  
133828 information concerning errors that it caused through incorrect usage or that it cannot correct.

133829 Austin Group Defect 1492 is applied, adding the **Default Behavior** paragraph.

### 133830 CONSEQUENCES OF ERRORS

133831 Several actions are possible when a utility encounters an error condition, depending on the  
133832 severity of the error and the state of the utility. Included in the possible actions of various  
133833 utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; and  
133834 validity checking of the file system or directory.

133835 The text about recursive traversing is meant to ensure that utilities such as *find* process as many  
133836 files in the hierarchy as they can. They should not abandon all of the hierarchy at the first error  
133837 and resume with the next command-line operand, but should attempt to keep going.

133838 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0001 [150] is applied.

133839 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0003 [913] is applied.

133840 Austin Group Defect 251 is applied, adding a note about the treatment of pathnames containing  
133841 any bytes that have the encoded value of a <newline> character.

133842 Austin Group Defect 1499 is applied, requiring utilities to exit with an exit status that indicates  
133843 an error occurred, instead of any non-zero exit status.

### 133844 APPLICATION USAGE

133845 This section provides additional caveats, issues, and recommendations to the developer.

### 133846 EXAMPLES

133847 This section provides sample usage.

### 133848 RATIONALE

133849 There is no additional rationale provided for this section.

### 133850 FUTURE DIRECTIONS

133851 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
133852 the future, and often cautions the developer to architect the code to account for a change in this  
133853 area. Note that a future directions statement should not be taken as a commitment to adopt a  
133854 feature or interface in the future.

133855 **SEE ALSO**  
 133856 There is no additional rationale provided for this section.

133857 **CHANGE HISTORY**  
 133858 There is no additional rationale provided for this section.

### 133859 **C.1.6 Considerations for Utilities in Support of Files of Arbitrary Size**

133860 This section is intended to clarify the requirements for utilities in support of large files.

133861 The utilities listed in this section are utilities which are used to perform administrative tasks  
 133862 such as to create, move, copy, remove, change the permissions, or measure the resources of a file.  
 133863 They are useful both as end-user tools and as utilities invoked by applications during software  
 133864 installation and operation.

133865 The *chgrp*, *chmod*, *chown*, *ln*, and *rm* utilities probably require use of large file-capable versions of  
 133866 *stat()*, *lstat()*, *nftw()*, and the **stat** structure.

133867 The *cat*, *cksum*, *cmp*, *cp*, *dd*, *mv*, and *touch* utilities probably require use of large file-capable  
 133868 versions of *creat()*, *open()*, and *fopen()*.

133869 The *cat*, *cksum*, *cmp*, *dd*, *df*, *du*, and *ls* utilities may require writing large integer values. For  
 133870 example:

- 133871 • The *cat* utility might have a **-n** option which counts <newline> characters.
- 133872 • The *cksum* and *ls* utilities report file sizes.
- 133873 • The *cmp* utility reports the line number at which the first difference occurs, and also has a  
 133874 **-l** option which reports file offsets.
- 133875 • The *dd*, *df*, *du*, and *ls* utilities report block counts.

133876 The *dd*, *find*, and *test* utilities may need to interpret command arguments that contain 64-bit  
 133877 values. For *dd*, the arguments include *skip=n*, *seek=n*, and *count=n*. For *find*, the arguments  
 133878 include **-sizen**. For *test*, the arguments are those associated with algebraic comparisons.

133879 The *df* utility might need to access large file systems with *statvfs()*.

133880 The *ulimit* utility will need to use large file-capable versions of *getrlimit()* and *setrlimit()* and be  
 133881 able to read and write large integer values.

133882 Austin Group Defect 1568 is applied, removing references to the *sum* utility.

### 133883 **C.1.7 Built-In Utilities**

133884 Other than the special built-in utilities, there is no requirement to build utilities into the shell  
 133885 itself. However, many shells implement certain utilities as regular built-ins for the following  
 133886 reasons:

- 133887 • To improve performance, especially for frequently used lightweight utilities (such as *test*,  
 133888 *true*, and *false*).
- 133889 • To eliminate the need for some sort of interprocess communication between the shell and  
 133890 those utilities that read or modify the shell's execution environment (such as *cd*).



133891 • To make it easier to satisfy the command search and execution requirements in XCU  
 133892 [Section 2.9.1.4](#) (on page 2502) for intrinsic utilities. Intrinsic utilities must be found prior to  
 133893 the *PATH* search. The shell could satisfy this requirement by keeping a list of the intrinsic  
 133894 utility pathnames and directly accessing the file-system versions regardless of *PATH*, but  
 133895 these utilities usually need to read or modify the shell's execution environment anyway.

133896 With the exception of the intrinsic utilities, all regular built-in utilities are subject to the *PATH*  
 133897 search and can be overridden by a specially crafted *PATH* environment variable.

133898 Earlier versions of this standard required that all of the regular built-in utilities, including  
 133899 intrinsic utilities, could be *exec*-ed. This was always a contentious requirement, and with the  
 133900 introduction of intrinsic utilities the standard developers decided to exempt the utilities that this  
 133901 standard requires to be intrinsic, with the exception of *kill*. The *kill* utility is still genuinely  
 133902 useful when *exec*-ed, only lacking support for the % job ID notation, whereas examples given of  
 133903 uses for the other utilities that are now exempted were considered contrived (such as using *cd* to  
 133904 test accessibility of a directory, which can be done using *test -x*). If an application needs *exec*-  
 133905 able versions of some of the exempted intrinsic utilities, it can easily provide them itself, on  
 133906 systems that support the (non-standard but ubiquitous) "#!" mechanism to make scripts  
 133907 executable by the *exec* family of functions, as links to a two-line shell script:

```
133908 #! /path/to/sh
133909 ${0##*/} "$@"
```

133910 Austin Group Defect 854 is applied, replacing the table of Regular Built-In Utilities with a  
 133911 reference to the new *Intrinsic Utilities* section.

133912 Austin Group Defect 1600 is applied, exempting the intrinsic utilities other than *kill* from the  
 133913 requirement that they can be *exec*-ed.

## 133914 C.1.8 Intrinsic Utilities

133915 There were varying reasons for including utilities in the table of intrinsic utilities:

133916 *alias, fc, unalias*

133917 The functionality of these utilities is performed more simply within the shell itself and that  
 133918 is the model most historical implementations have used.

133919 *bg, fg, jobs*

133920 All of the job control-related utilities are eligible for built-in status because that is the model  
 133921 most historical implementations have used.

133922 *cd, getopts, hash, read, type, ulimit, umask, wait*

133923 The functionality of these utilities is performed more simply within the context of the  
 133924 current process. An example can be taken from the usage of the *cd* utility. The purpose of  
 133925 the *cd* utility is to change the working directory for subsequent operations. The actions of *cd*  
 133926 affect the process in which *cd* is executed and all subsequent child processes of that process.  
 133927 Based on the POSIX standard process model, changes in the process environment of a child  
 133928 process have no effect on the parent process. If the *cd* utility were executed from a child  
 133929 process, the working directory change would be effective only in the child process. Child  
 133930 processes initiated subsequent to the child process that executed the *cd* utility would not  
 133931 have a changed working directory relative to the parent process.

133932 *command*

133933 This utility was placed in the table primarily to protect scripts that are concerned about  
 133934 their *PATH* being manipulated. The "secure" shell script example in the *command* utility in  
 133935 the Shell and Utilities volume of POSIX.1-2024 would not be possible if a *PATH* change  
 133936 retrieved an alien version of *command*. (An alternative would have been to implement

133937 *getconf* as a built-in, but the standard developers considered that it carried too many  
133938 changing configuration strings to require in the shell.)

133939 *kill* Since *kill* provides optional job control functionality using shell notation (%1, %2, and so on),  
133940 some implementations would find it extremely difficult to provide this outside the shell.

133941 The following utilities are frequently implemented as intrinsic (and built-in) utilities. Future  
133942 versions of this standard might not allow these utilities, or any other standard utility not in  
133943 Table 1-5 (on page 2470), to be intrinsic; implementations are encouraged to implement these as  
133944 non-intrinsic utilities instead (but still built-in if they were previously built-in).

133945 *l, echo, false, newgrp, printf, pwd, test, true*

133946 All utilities, including those in the table, are accessible via the *system()* and *popen()* functions in  
133947 the System Interfaces volume of POSIX.1-2024. There are situations where the return  
133948 functionality of *system()* and *popen()* is not desirable. Applications that require the exit status of  
133949 the invoked utility will not be able to use *system()* or *popen()*, since the exit status returned is  
133950 that of the command language interpreter rather than that of the invoked utility. The alternative  
133951 for such applications is the use of the *exec* family.

133952 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0004 [705] is applied.

133953 Austin Group Defect 854 is applied, adding intrinsic utilities.

## 133954 C.2 Shell Command Language

### 133955 C.2.1 Shell Introduction

133956 The System V shell was selected as the starting point for the Shell and Utilities volume of  
133957 POSIX.1-2024. The BSD C shell was excluded from consideration for the following reasons:

- 133958 • Most historically portable shell scripts assume the Version 7 Bourne shell, from which the  
133959 System V shell is derived.
- 133960 • The majority of tutorial materials on shell programming assume the System V shell.

133961 The construct "#!" is reserved for implementations wishing to provide that extension. If it were  
133962 not reserved, the Shell and Utilities volume of POSIX.1-2024 would disallow it by forcing it to be  
133963 a comment. As it stands, a strictly conforming application must not use "#!" as the first two  
133964 characters of the file.

133965 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

133966 Austin Group Defect 1514 is applied, correcting a misuse of the term "positional parameter".

### 133967 C.2.2 Quoting

133968 Although this section contains a note indicating that a future version of this standard may  
133969 extend the conditions under which some characters are special, there are no plans to do so. The  
133970 note is there to encourage application writers to future-proof their shell code. In some cases  
133971 existing widespread use of the characters unquoted would preclude them being given a special  
133972 meaning in those use cases. For example, commas are in widespread use in filenames (notably  
133973 by RCS and CVS) and it is common to pass the token "{}" as an argument to *find* and *xargs*  
133974 unquoted.

133975 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

133976 Austin Group Defects 1191 and 1193 are applied, adding:

133977 ] ^ - ! { , }

133978 to the list of characters that might need to be quoted under certain circumstances.

#### 133979 C.2.2.1 *Escape Character (Backslash)*

133980 Austin Group Defect 500 is applied, changing “follows” to “immediately follows”.

#### 133981 C.2.2.2 *Single-Quotes*

133982 A <backslash> cannot be used to escape a single-quote in a single-quoted string. An embedded  
 133983 quote can be created by writing, for example: "'a'\''b'", which yields "a'b". (See XCU  
 133984 [Section 2.6.5](#) (on page 2491) for a better understanding of how portions of words are either split  
 133985 into fields or remain concatenated.) A single token can be made up of concatenated partial  
 133986 strings containing all three kinds of quoting or escaping, thus permitting any combination of  
 133987 characters.

#### 133988 C.2.2.3 *Double-Quotes*

133989 The escaped <newline> used for line continuation is removed entirely from the input and is not  
 133990 replaced by any white space. Therefore, it cannot serve as a token separator.

133991 In double-quoting, if a <backslash> is immediately followed by a character that would be  
 133992 interpreted as having a special meaning, the <backslash> is deleted and the subsequent  
 133993 character is taken literally. If a <backslash> does not precede a character that would have a  
 133994 special meaning, it is left in place unmodified and the character immediately following it is also  
 133995 left unmodified. Thus, for example:

133996 "\\$" -> \$

133997 "\a" -> \a

133998 It would be desirable to include the statement “The characters from an enclosed "\${" to the  
 133999 matching '}' shall not be affected by the double-quotes”, similar to the one for "\$()".  
 134000 However, historical practice in the System V shell prevents this.

134001 Shell implementations differ widely in their handling of unescaped double-quote characters  
 134002 inside "\${...}" (except for the four substring-processing variants). Hence this standard leaves  
 134003 the behavior unspecified. Single-quotes are ordinary characters in this context, and so cannot be  
 134004 used to quote a '}' within "\${...}". However, <backslash> can be used to escape a '}'. For  
 134005 example, the value of *foo* assigned by the following commands is '}' :

134006 unset bar

134007 foo="\$\${bar-\\} "

134008 When <backslash> is used in this way it is a special character and is therefore removed during  
 134009 quote removal, even though it would not be removed in:

134010 foo="\} "

134011 Differences in processing the "\${...}" form led to inconsistencies between the historical  
 134012 System V shell, BSD, and KornShells, and the text in the Shell and Utilities volume of  
 134013 POSIX.1-2024 is an attempt to converge them without breaking too many applications. The only  
 134014 alternative to this compromise between shells would be to make the behavior unspecified not

134015 just for unescaped double-quote but also for unescaped single-quote, '{', or '}'. The chosen  
 134016 requirements provide the maximum consistency between normal double-quote behavior and  
 134017 parameter expansion within double-quotes; the only real difference being the ability to escape a  
 134018 '}' with <backslash>.

134019 Some implementations have allowed the end of the word to terminate the backquoted command  
 134020 substitution, such as in:

```
134021 "`echo hello"
```

134022 This usage is undefined; the matching backquote is required by the Shell and Utilities volume of  
 134023 POSIX.1-2024. The other undefined usage can be illustrated by the example:

```
134024 sh -c '` echo "foo`'
```

134025 The description of the recursive actions involving command substitution can be illustrated with  
 134026 an example. Upon recognizing the introduction of command substitution, the shell parses input  
 134027 (in a new context), gathering the source for the command substitution until an unbalanced ')' or  
 134028 or '' is located. For example, in the following:

```
134029 echo "$(date; echo "  

  134030     one" )"
```

134031 the double-quote following the *echo* does not terminate the first double-quote; it is part of the  
 134032 command substitution script. Similarly, in:

```
134033 echo "$(echo *)" "
```

134034 the <asterisk> is not quoted since it is inside command substitution; however:

```
134035 echo "$(echo "*" )"
```

134036 is quoted (and represents the <asterisk> character itself).

134037 The '\$'...' construct does not retain its special meaning inside double quotes. This was  
 134038 discussed by the standard developers and rejected. Note that '\$'...' is a quoting mechanism  
 134039 and not an expansion. Losing the special meaning inside double-quotes is consistent with other  
 134040 quoting mechanisms losing their special meaning when quoted.

134041 Austin Group Defect 221 is applied, clarifying the behavior of double-quotes within the string of  
 134042 characters from "\${" to the matching '}' in parameter expansions using that form.

134043 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

134044 Austin Group Defect 500 is applied, clarifying the behavior of <backslash> within double-  
 134045 quotes.

134046 Austin Group Defect 1268 is applied, clarifying the effect of double-quotes on the results of  
 134047 parameter expansion, command substitution, or arithmetic expansion.

134048 Austin Group Defect 1342 is applied, clarifying the requirements for alias substitutions inside  
 134049 command substitutions.

#### 134050 C.2.2.4 Dollar-Single-Quotes

134051 The '\$'...' quoting construct has been implemented in several recent shells. It is similar to  
 134052 character string literals ("...") in the ISO C standard with the following exceptions:

- 134053 • The \x escape sequence in C can be followed by an arbitrary number of hexadecimal  
 134054 digits. The *ksh93* implementation of '\$'...' also consumes an arbitrary number of  
 134055 hexadecimal digits; *bash* consumes at most two hexadecimal digits in this case. This  
 134056 standard leaves the result unspecified if more than two hexadecimal digits follow \x.

134057 (Note that a hexadecimal escape followed by a literal hexadecimal character can always be  
134058 represented as `$'\xXX'X`.)

134059 • The `\c` escape sequence is not included in the ISO C standard. There was also some  
134060 disagreement in shells that historically supported `\c` escape sequences in `$'...'`. These  
134061 include:

134062 — Whether `\cA` through `\cZ` produced the byte values 1 through 26, respectively or  
134063 supported the codeset independent control character as specified by the *stty* utility.  
134064 This standard requires codeset independence.

134065 — Whether `\c[`, `\c\`, `\c]`, `\c^`, `\c_`, and `\c?` could be used to yield the <ESC>,  
134066 <FS>, <GS>, <RS>, <US>, and <DEL> control characters, respectively. This standard  
134067 requires support for all of the control characters except NULL (matching what is  
134068 done in the *stty* utility).

134069 — Whether `\c\` or `\c\` was used to represent <FS>. This standard requires `\c\` to  
134070 make <backslash>-escape processing consistent.

134071 The implementors of the most common shells that implement `$'\cX'` agreed to convert to  
134072 the behavior specified in this standard.

134073 Some shells also allow `\c<arbitrary_control_character>` to act as an inverse function to  
134074 `\cX` (that is, `\cm` and `\cM` yield <CR> and `\c<CR>` yields `m` or `M`). This standard leaves this  
134075 behavior implementation-defined.

134076 • The `\e` escape sequence is not included in the ISO C standard, but was provided by all  
134077 historical shells that supported `$'...'`. Some also supported `\E` as a synonym. One  
134078 member of the group objected to adding `\e` because the <ESC> control character is not  
134079 required to be in the portable character set. The `\e` sequence is included because many  
134080 historical users of `$'...'` expect it to be there. The `\E` sequence is not included in this  
134081 standard because <backslash>-escape sequences that start with <backslash> followed by  
134082 an uppercase letter (except `\U`) are reserved by the ISO C standard for implementation use.

134083 • The `\ddd` octal escape sequence and the `\xXX` hexadecimal escape sequence can be used to  
134084 insert a null byte into a C character string literal and into a `$'...'` quoted word in this  
134085 standard. In C, any characters specified after that null byte (including escape sequences)  
134086 continue to be processed and added to the character string literal. In `$'...'` in the shell  
134087 this standard allows the equivalent behavior but also allows the null byte and all  
134088 remaining characters up to the terminating unescaped single-quote to be evaluated and  
134089 discarded. The latter (which was historic practice in *bash*, but not in *ksh93*) allows an  
134090 escape sequence producing a null byte to terminate the dollar-single-quoted expansion,  
134091 but not terminate the token in which it appears if there are characters remaining in the  
134092 token. For example:

```
134093 printf a$b\0c\'d
```

134094 is required by this standard to produce:

```
134095 abd
```

134096 while historic versions of *ksh93* produced:

```
134097 ab
```

134098 • The ISO C standard specifies `\uXXXX` and `\UXXXXXXXX` escape sequences. These need not  
134099 be supported by `$'...'` in the shell. They were omitted because current shell  
134100 implementations that support them differ in behavior. In particular, some shells always  
134101 convert them to the UTF-8 encoding for the named character, even if the current locale's  
134102 character set does not have UTF-8 encoding.

- 134103 • The double-quote character can be used literally, while the single-quote character must be  
134104 represented as an escape sequence. In C, single-quote can be used literally, while double-  
134105 quote requires an escape sequence.
- 134106 • A <backslash> immediately followed by a <newline> has unspecified behavior. In C, this  
134107 sequence is used for line continuations, where both the <backslash> and <newline> are  
134108 deleted and a diagnostic is required if a closing quote is not encountered before a  
134109 <newline> that is not preceded by <backslash>. In current shell implementations, three  
134110 different behaviors have been observed.
- 134111 • The use of <backslash>-escape sequences not described in this standard results in  
134112 unspecified behavior. In C, the result is not a token and a diagnostic is required. This  
134113 allows shells to recognize other <backslash>-escape sequences in other ways as extensions  
134114 to this standard. Furthermore, existing implementations already had different behaviors  
134115 for some <backslash>-escape sequences when \$'...' processing was added to this  
134116 standard.

134117 This standard makes the results implementation-defined if `\e` or `\cX` specifies a character that is  
134118 not present in the current locale. Application authors should note that implementations are  
134119 permitted to have a wide range of behaviors when encountering an unsupported character. For  
134120 example:

- 134121 • The shell might produce an error, possibly causing the shell to terminate.
- 134122 • The unsupported character might be silently discarded.
- 134123 • The unsupported character might be replaced with another character of a different  
134124 character class.
- 134125 • The unsupported character might be replaced with a shell-special character (e.g., ' ? ').
- 134126 • The unsupported character might be replaced with multiple characters, shell-special or  
134127 regular (e.g. if <ESC> is not supported, \$'\e' may be replaced by "???", "XXX", or  
134128 "<ESC>").

134129 However, implementations must document their behavior, and they are prohibited from  
134130 replacing an unsupported character with bytes that do not form valid characters in the current  
134131 locale's character set (e.g., encoding in UTF-8 when the locale has a 7-bit character set). This  
134132 standard does not specify a way for script authors to determine beforehand whether a particular  
134133 `\cX` sequence specifies a character that exists in the current locale. At the time this feature was  
134134 standardized, no known implementations provided such a capability.

134135 Note that the escape sequences recognized by \$'...', file format notation (see [Table 5-1](#), on  
134136 page 113), XSI-conforming implementations of the *echo* utility (see the utility's OPERANDS  
134137 section in *echo*), and the *printf* utility's *format* operand (see the utility's EXTENDED  
134138 DESCRIPTION in *printf*) are not the same. Some escape sequences are not recognized by all of  
134139 the above, the `\c` escape sequence in *echo* is not at all like the `\c` escape sequence in \$'...',  
134140 octal escape sequences in some of the above accept one to four octal digits and require a leading  
134141 zero while others accept one to three octal digits and do not require a leading zero.

134142 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

134143 **C.2.3 Token Recognition**

134144 The "(" and ")" symbols are control operators in the KornShell, used for an alternative  
 134145 syntax of an arithmetic expression command. A conforming application cannot use "(" as a  
 134146 single token (with the exception of the "\$ (" form for shell arithmetic).

134147 On some implementations, the symbol "(" is a control operator; its use produces unspecified  
 134148 results. Applications that wish to have nested subshells, such as:

```
134149 ((echo Hello); echo World))
```

134150 must separate the "(" characters into two tokens by including white space between them.  
 134151 Some systems may treat these as invalid arithmetic expressions instead of subshells.

134152 Certain combinations of characters are invalid in portable scripts, as shown in the grammar.  
 134153 Implementations may use these combinations (such as "|&") as valid control operators. Portable  
 134154 scripts cannot rely on receiving errors in all cases where this volume of POSIX.1-2024 indicates  
 134155 that a syntax is invalid.

134156 The (3) rule about combining characters to form operators is not meant to preclude systems from  
 134157 extending the shell language when characters are combined in otherwise invalid ways.  
 134158 Conforming applications cannot use invalid combinations, and test suites should not penalize  
 134159 systems that take advantage of this fact. For example, the unquoted combination "|&" is not  
 134160 valid in a POSIX script, but has a specific KornShell meaning.

134161 The (10) rule about '#' as the current character is the first in the sequence in which a new token  
 134162 is being assembled. The '#' starts a comment only when it is at the beginning of a token. This  
 134163 rule is also written to indicate that the search for the end-of-comment does not consider escaped  
 134164 <newline> specially, so that a comment cannot be continued to the next line.

134165 Because a *complete\_command* encountered during a *program* is executed before the next  
 134166 *complete\_command* is tokenized and parsed, syntax errors are not discovered by the shell until  
 134167 just before the code would be executed. While in some cases it might be desirable to detect and  
 134168 react to syntax errors before anything is executed (possible with *sh -n*), deferring the discovery  
 134169 of syntax errors has several benefits:

- 134170 • It makes it possible for script authors to test for the availability of a nonstandard extension  
 134171 and react appropriately before the use of the extension would trigger a syntax error.
- 134172 • It makes it possible to create self-extracting tarballs (a shell script concatenated with a  
 134173 payload archive that extracts the archive when executed).
- 134174 • The shell does not have to read and parse the complete script before execution, which  
 134175 reduces memory usage when executing extremely long scripts.

134176 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0005 [718], XCU/TC2-2008/0006  
 134177 [647], XCU/TC2-2008/0007 [568], and XCU/TC2-2008/0008 [648] are applied.

134178 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

134179 Austin Group Defect 1036 is applied, clarifying how here-documents are parsed.

134180 Austin Group Defect 1055 is applied, clarifying how much of a *program* is parsed before the  
 134181 parsed commands are executed.

134182 Austin Group Defect 1083 is applied, changing "the next character" to "each character in turn".

134183 Austin Group Defect 1085 is applied, clarifying requirements for the start and end of tokens.

134184 C.2.3.1 *Alias Substitution*

134185 The alias capability was added because it is widely used in historical implementations by  
134186 interactive users.

134187 The definition of `alias name` precludes an alias name containing a `<slash>` character. Since the  
134188 text applies to the command words of simple commands, reserved words (in their proper  
134189 places) cannot be confused with aliases.

134190 The placement of alias substitution in token recognition makes it clear that it precedes all of the  
134191 word expansion steps.

134192 An example concerning trailing `<blank>` characters and reserved words follows. If the user  
134193 types:

```
134194 $ alias foo="/bin/ls "  
134195 $ alias while="/"
```

134196 The effect of executing:

```
134197 $ while true  
134198 > do  
134199 > echo "Hello, World"  
134200 > done
```

134201 is a never-ending sequence of `"Hello, World"` strings to the screen. However, if the user  
134202 types:

```
134203 $ foo while
```

134204 the result is an `ls` listing of `/`. Since the alias substitution for `foo` ends in a `<space>`, the next word  
134205 is checked for alias substitution. The next word, `while`, has also been aliased, so it is substituted  
134206 as well. Since it is not in the proper position as a command word, it is not recognized as a  
134207 reserved word.

134208 If the user types:

```
134209 $ foo; while
```

134210 `while` retains its normal reserved-word properties.

134211 Some implementations add a `<space>` after the alias value when performing alias substitution in  
134212 order to prevent the last character of the alias value and the first character after the alias name in  
134213 the input from combining to form an operator. However, the extra `<space>` can have side-effects  
134214 in other situations, such as if the alias value ends with an unquoted `<backslash>`.  
134215 Implementations which do this are encouraged to change to an alternative method of delimiting  
134216 a partial operator token at the end of an alias value.

134217 Some, but not all, shell implementations do not process changes to alias definitions until the  
134218 current *compound\_list* (see XCU Section 2.10, on page 2512) has completed. In these shells, alias  
134219 changes do not take effect until the end of the dot script, *eval* command, function invocation, **if**  
134220 statement, **case** statement, **for** statement, **while** statement, or **until** statement containing the alias  
134221 change.

134222 Many shell implementations execute the contents of a file, typically `~/.profile`, when invoked as  
134223 a login shell. The standard developers are unaware of any such implementations that process  
134224 the contents of `~/.profile` (and similar startup files) as a single *compound\_list*, so alias changes in  
134225 `~/.profile` typically do take effect before the end of `~/.profile`.

134226 Austin Group Defects 953 and 1630 are applied, providing additional detail on how alias  
134227 substitution is performed.



134228 **C.2.4 Reserved Words**

134229 All reserved words are recognized syntactically as such in the contexts described. However, note  
 134230 that **in** is the only meaningful reserved word after a **case** or **for**; similarly, **in** is not meaningful as  
 134231 the first word of a simple command.

134232 Reserved words are recognized only when they are delimited (that is, meet the definition of XBD  
 134233 Section 3.420, on page 93), whereas operators are themselves delimiters. For instance, ' ( ' and  
 134234 ' ) ' are control operators, so that no <space> is needed in (*list*). However, '{ ' and ' } ' are  
 134235 reserved words in { *list*; }, so that in this case the leading <space> and <semicolon> are required.

134236 The list of unspecified reserved words is from the KornShell, so conforming applications cannot  
 134237 use them in places a reserved word would be recognized. Earlier versions of this standard  
 134238 omitted **time** from this list, so that the *time* utility could be included without requiring  
 134239 applications to quote all or part of its name (or use other measures) in order to avoid it being  
 134240 treated as a reserved word. However, although the intent was to allow the reserved word  
 134241 implementation (as evidenced by use of *time* in pipelines being unspecified, and explicit  
 134242 mention in the rationale of the *time* utility), the conditions under which the behavior was  
 134243 unspecified were insufficient to allow this. In particular, redirection in KornShell does not work  
 134244 in the normal way when **time** is a reserved word:

```
134245 time utility 2> time.out
```

134246 only writes the standard error from *utility* to **time.out**; the timing information is written to the  
 134247 shell's standard error, but these versions of the standard required the timing information to be  
 134248 written to **time.out**. Another issue was that if **time** is a reserved word, an application cannot  
 134249 define a function with that name, but these versions of the standard required that applications  
 134250 could do so. Hence **time** has now been added to the list of unspecified reserved words, but with  
 134251 its use as a reserved word limited in order to be compatible with its use as a utility in the cases  
 134252 where the two have traditionally had the same effect (other than possible output format  
 134253 differences).

134254 There was a strong argument for promoting braces to operators (instead of reserved words), so  
 134255 they would be syntactically equivalent to subshell operators. Concerns about compatibility  
 134256 outweighed the advantages of this approach. Nevertheless, conforming applications should  
 134257 consider quoting '{ ' and ' } ' when they represent themselves.

134258 When used in circumstances where reserved words are recognized, all words whose final  
 134259 character is a <colon> ( ' : ' ) are reserved. The case of a name suffixed with a colon is reserved to  
 134260 allow implementations to support named labels for flow control; see the RATIONALE for the  
 134261 *break* special built-in utility. Other words ending in <colon> are reserved to provide  
 134262 implementations with a way to add new reserved words while still conforming to this standard.

134263 It is possible that a future version of the Shell and Utilities volume of POSIX.1-2024 may require  
 134264 that '{ ' and ' } ' be treated individually as control operators, although the token "{ } " will  
 134265 probably be a special-case exemption from this because of the often-used *find*{ } construct.

134266 Austin Group Defect 267 is applied, adding **time** to the list of words that may be recognized as  
 134267 reserved words while specifying its behavior if it is recognized as a reserved word, and  
 134268 extending the reservation of words whose final character is <colon> from those that are a name  
 134269 followed by a <colon> to all such words.

134270 Austin Group Defect 465 is applied, adding **namespace** to the list of words that may be  
 134271 recognized as reserved words.

134272 **C.2.5 Parameters and Variables**

134273 Austin Group Defect 1561 is applied, clarifying that parameters can contain byte sequences that  
 134274 do not form valid characters and that the shell processes their values as characters only when  
 134275 performing operations that are described in this standard in terms of characters.

134276 C.2.5.1 *Positional Parameters*

134277 Austin Group Defect 1491 is applied, clarifying the handling of leading zeros in positional  
 134278 parameter identifiers.

134279 C.2.5.2 *Special Parameters*

134280 Most historical implementations implement subshells by forking; thus, the special parameter  
 134281 '\$' does not necessarily represent the process ID of the shell process executing the commands  
 134282 since the subshell execution environment preserves the value of '\$'.

134283 If a subshell were to execute a background command, the value of "\$!" for the parent would  
 134284 not change. For example:

```
134285 (
134286 date &
134287 echo $!
134288 )
134289 echo $!
```

134290 would echo two different values for "\$!".

134291 The "\$-" special parameter can be used to save and restore *set* options:

```
134292 Save=$(echo $- | sed 's/[ics]//g')
134293 ...
134294 set +aCefnuvx
134295 if [ -n "$Save" ]; then
134296     set -$Save
134297 fi
```

134298 The three options are removed using *sed* in the example because they may appear in the value of  
 134299 "\$-" (from the *sh* command line), but are not valid options to *set*.

134300 The descriptions of parameters '\*' and '@' assume the reader is familiar with the field  
 134301 splitting discussion in XCU [Section 2.6.5](#) (on page 2491) and understands that portions of the  
 134302 word remain concatenated unless there is some reason to split them into separate fields.

134303 The following examples illustrate some of the ways in which '\*' and '@' can be expanded:

```
134304 set "abc" "def ghi" "jkl"
134305 unset novar
134306 IFS=' ' # a space
134307 printf '%s\n' $*
134308 abc
134309 def
134310 ghi
134311 jkl
134312 printf '%s\n' "$*"
134313 abc def ghi jkl
134314 printf '%s\n' xx$*yy
134315 xxabc
```

```

134316     def
134317     ghi
134318     jklyy
134319     printf '%s\n' "xx$*yy"
134320     xxabc def ghi jklyy
134321     printf '%s\n' $@
134322     abc
134323     def
134324     ghi
134325     jkl
134326     printf '%s\n' "$@"
134327     abc
134328     def ghi
134329     jkl
134330     printf '%s\n' ${1+"$@"}
134331     abc
134332     def ghi
134333     jkl
134334     printf '%s\n' ${novar-"$@"}
134335     abc
134336     def ghi
134337     jkl
134338     printf '%s\n' xx$@yy
134339     xxabc
134340     def
134341     ghi
134342     jklyy
134343     printf '%s\n' "xx$@yy"
134344     xxabc
134345     def ghi
134346     jklyy
134347     printf '%s\n' $@$@
134348     abc
134349     def
134350     ghi
134351     jklabc
134352     def
134353     ghi
134354     jkl
134355     printf '%s\n' "$@$@"
134356     abc
134357     def ghi
134358     jklabc
134359     def ghi
134360     jkl
134361     IFS=:
134362     printf '%s\n' "$*"
134363     abc: def ghi: jkl
134364     var=$*; printf '%s\n' "$var"
134365     abc: def ghi: jkl
134366     var="$*"; printf '%s\n' "$var"
134367     abc: def ghi: jkl
134368     unset var

```

```
134369 printf '%s\n' ${var-$*}
134370 abc
134371 def ghi
134372 jkl
134373 printf '%s\n' "${var-$*}"
134374 abc: def ghi: jkl
134375 printf '%s\n' ${var-"$*"}
134376 abc: def ghi: jkl
134377 printf '%s\n' ${var=$*}
134378 abc
134379 def ghi
134380 jkl
134381 printf 'var=%s\n' "$var"
134382 var=abc: def ghi: jkl
134383 unset var
134384 printf '%s\n' "${var=$*}"
134385 abc: def ghi: jkl
134386 printf 'var=%s\n' "$var"
134387 var=abc: def ghi: jkl

134388 IFS=' ' # null
134389 printf '%s\n' "$*"
134390 abcdef ghi jkl
134391 var=$*; printf '%s\n' "$var"
134392 abcdef ghi jkl
134393 var="$*"; printf '%s\n' "$var"
134394 abcdef ghi jkl
134395 unset var
134396 printf '%s\n' ${var-$*}
134397 abc
134398 def ghi
134399 jkl
134400 printf '%s\n' "${var-$*}"
134401 abcdef ghi jkl
134402 printf '%s\n' ${var-"$*"}
134403 abcdef ghi jkl
134404 printf '%s\n' ${var=$*}
134405 abcdef ghi jkl
134406 printf 'var=%s\n' "$var"
134407 var=abcdef ghi jkl
134408 unset var
134409 printf '%s\n' "${var=$*}"
134410 abcdef ghi jkl
134411 printf 'var=%s\n' "$var"
134412 var=abcdef ghi jkl
134413 printf '%s\n' "$@"
134414 abc
134415 def ghi
134416 jkl

134417 unset IFS
134418 printf '%s\n' "$*"
134419 abc def ghi jkl
134420 var=$*; printf '%s\n' "$var"
```

```

134421      abc def ghi jkl
134422      var="$*"; printf '%s\n' "$var"
134423      abc def ghi jkl
134424      unset var
134425      printf '%s\n' "${var-$*}
134426      abc
134427      def
134428      ghi
134429      jkl
134430      printf '%s\n' "${var-$*}"
134431      abc def ghi jkl
134432      printf '%s\n' "${var-$*}"
134433      abc def ghi jkl
134434      printf '%s\n' "${var=$*}"
134435      abc
134436      def
134437      ghi
134438      jkl
134439      printf 'var=%s\n' "$var"
134440      var=abc def ghi jkl
134441      unset var
134442      printf '%s\n' "${var=$*}"
134443      abc def ghi jkl
134444      printf 'var=%s\n' "$var"
134445      var=abc def ghi jkl
134446      printf '%s\n' "$@"
134447      abc
134448      def ghi
134449      jkl

134450      set one "" three
134451      printf ' [%s]\n' $*
134452      [one]
134453      [] (this line of output is optional)
134454      [three]
134455      printf ' [%s]\n' $@
134456      [one]
134457      [] (this line of output is optional)
134458      [three]

134459      set --
134460      printf ' [%s]\n' foo "$*"
134461      [foo]
134462      []
134463      printf ' [%s]\n' foo "$novar$*(echo)"
134464      [foo]
134465      []
134466      printf ' [%s]\n' foo $@
134467      [foo]
134468      printf ' [%s]\n' foo "$@"
134469      [foo]
134470      printf ' [%s]\n' foo '$@'
134471      [foo]
134472      []

```

```

134473 printf "[%s]\n" foo ""$@"
134474 [foo]
134475 [ ]
134476 printf "[%s]\n" foo "$novar${$(echo)}"
134477 [foo]
134478 [ ] (this line of output is optional)
134479 printf "[%s]\n" foo ""$novar${$(echo)}"
134480 [foo]
134481 [ ]

```

134482 In all of the following commands the results of the expansion of '@' (if performed) are  
 134483 unspecified:

```

134484 var=$@
134485 var=""$@"
134486 printf "%s\n" ${var=$@}
134487 printf "%s\n" "${var=$@}"
134488 printf "%s\n" ${var=""$@"}
134489 printf "%s\n" ${var?}$@}
134490 printf "%s\n" "${var?}$@"}
134491 printf "%s\n" ${var?"$@"}
134492 printf "%s\n" ${#@}
134493 printf "%s\n" "${#@}"
134494 printf "%s\n" ${@%foo}
134495 printf "%s\n" "${@%foo}"
134496 printf "%s\n" ${@#foo}
134497 printf "%s\n" "${@#foo}"
134498 printf "%s\n" ${var%$@}
134499 printf "%s\n" "${var%$@"}
134500 printf "%s\n" ${var%"$@"}
134501 printf "%s\n" ${var%%$@}
134502 printf "%s\n" "${var%%$@"}
134503 printf "%s\n" ${var%%%"$@"}
134504 printf "%s\n" ${var#$@}
134505 printf "%s\n" "${var#$@"}
134506 printf "%s\n" ${var#"$@"}
134507 printf "%s\n" ${var##$@}
134508 printf "%s\n" "${var##$@"}
134509 printf "%s\n" ${var##%"$@"}

```

134510 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0009 [888] is applied.

134511 Austin Group Defect 1039 is applied, clarifying the description of the '-' special parameter.

134512 Austin Group Defect 1052 is applied, clarifying that decimal valued special parameters expand  
 134513 to the shortest representation.

134514 Austin Group Defects 1150 and 1309 are applied, clarifying the description of the '?' special  
 134515 parameter.

134516 Austin Group Defect 1254 is applied, clarifying how the '?' and '!' special parameters are  
 134517 affected by job control.

## 134518 C.2.5.3 Shell Variables

134519 Since shell variables are parameters denoted by a name, the shell cannot initialize shell variables  
 134520 from environment variables that do not have a valid name. However, the shell may initialize  
 134521 parameters that do not have valid names from such environment variables.

134522 See the discussion of *IFS* in Section C.2.6.5 (on page 3889) and the RATIONALE for the *sh* utility.

134523 The prohibition on *LC\_CTYPE* changes affecting lexical processing protects the shell  
 134524 implementor (and the shell programmer) from the ill effects of changing the definition of  
 134525 <blank> or the set of alphabetic characters in the current environment. It would probably not be  
 134526 feasible to write a compiled version of a shell script without this rule. The rule applies only to  
 134527 the current invocation of the shell and its subshells—invoking a shell script or performing *exec*  
 134528 *sh* would subject the new shell to the changes in *LC\_CTYPE*.

134529 Other common environment variables used by historical shells are not specified by the Shell and  
 134530 Utilities volume of POSIX.1-2024, but they should be reserved for the historical uses.

134531 Tilde expansion for components of *PATH* in an assignment such as:

```
134532 PATH=~hlj/bin:~dwc/bin:$PATH
```

134533 is a feature of some historical shells and is allowed by the wording of XCU Section 2.6.1 (on page  
 134534 2485). Note that the <tilde> characters are expanded during the assignment to *PATH*, not when  
 134535 *PATH* is accessed during command search.

134536 The following entries represent additional information about variables included in the Shell and  
 134537 Utilities volume of POSIX.1-2024, or rationale for common variables in use by shells that have  
 134538 been excluded:

134539 — (Underscore.) While <underscore> is historical practice, its overloaded usage  
 134540 in the KornShell is confusing, and it has been omitted from the Shell and  
 134541 Utilities volume of POSIX.1-2024.

134542 *ENV* This variable can be used to set aliases and other items local to the invocation  
 134543 of a shell. The file referred to by *ENV* differs from **\$HOME/.profile** in that  
 134544 **.profile** is typically executed at session start-up, whereas the *ENV* file is  
 134545 executed at the beginning of each shell invocation. The *ENV* value is  
 134546 interpreted in a manner similar to a dot script, in that the commands are  
 134547 executed in the current environment and the file needs to be readable, but not  
 134548 executable. However, unlike dot scripts, no *PATH* searching is performed. This  
 134549 is used as a guard against Trojan Horse security breaches.

134550 *ERRNO* This variable was omitted from the Shell and Utilities volume of POSIX.1-2024  
 134551 because the values of error numbers are not defined in POSIX.1-2024 in a  
 134552 portable manner.

134553 *FCEDIT* Since this variable affects only the *fc* utility, it has been omitted from this more  
 134554 global place. The value of *FCEDIT* does not affect the command-line editing  
 134555 mode in the shell; see the description of *set -o vi* in the *set* built-in utility.

134556 *PS1* This variable is used for interactive prompts. Historically, the “superuser”  
 134557 has had a prompt of '#'. Since privileges are not required to be monolithic, it  
 134558 is difficult to define which privileges should cause the alternate prompt.  
 134559 However, a sufficiently powerful user should be reminded of that power by  
 134560 having an alternate prompt.

134561 *PS3* This variable is used by the KornShell for the *select* command. Since the POSIX  
 134562 shell does not include *select*, *PS3* was omitted.

134563 *PS4* This variable is used for shell debugging. For example, the following script:

```
134564 PS4=' [ ${LINENO} ] + '
134565 set -x
134566 echo Hello
```

134567 writes the following to standard error:

```
134568 [3]+ echo Hello
```

134569 *RANDOM* This pseudo-random number generator was not seen as being useful to  
134570 interactive users.

134571 *SECONDS* Although this variable is sometimes used with *PS1* to allow the display of the  
134572 current time in the prompt of the user, it is not one that would be manipulated  
134573 frequently enough by an interactive user to include in the Shell and Utilities  
134574 volume of POSIX.1-2024.

134575 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0002 [152] is applied.

134576 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0010 [888], XCU/TC2-2008/0011  
134577 [884], and XCU/TC2-2008/0012 [494] are applied.

134578 Austin Group Defect 953 is applied, clarifying how the *ENV* file is parsed.

134579 Austin Group Defect 1006 is applied, clarifying how the values of the *PS1*, *PS2*, and *PS4*  
134580 variables are expanded.

134581 Austin Group Defect 1441 is applied, requiring *PS4* to be used in non-interactive shells.

134582 Austin Group Defect 1511 is applied, making the description of *LINENO* consistent with other  
134583 variables as regards how they relate to the User Portability Utilities option.

134584 Austin Group Defect 1561 is applied, clarifying that shell variables are initialized only from  
134585 environment variables that have valid names.

## 134586 C.2.6 Word Expansions

134587 Some shells implement *brace expansion* which expands, for example, *file{A,B,C}.c* into the  
134588 fields *fileA.c*, *fileB.c*, and *fileC.c* or *file{1..3}.c* into the fields *file1.c*, *file2.c*,  
134589 and *file3.c*. This form of expansion is allowed but not required by this standard, but if  
134590 supported must be performed before all of the standard word expansions. A variant which some  
134591 shells implement whereby brace expansion is performed following field splitting was  
134592 considered by the standard developers and rejected because it causes surprising behavior if the  
134593 results of parameter expansion and command substitution happen to produce a valid brace  
134594 expansion. For example, if the shell variable *path* contains an arbitrary pathname, glob pattern  
134595 applications cannot rely on *some\_command -- \$path* passing a list of pathnames that match  
134596 the pattern to *some\_command*. Note that quoting the braces or commas prevents this form of  
134597 expansion, but quoting the periods need not prevent it.

134598 Step (2) refers to the “portions of fields generated by step (1)”. For example, if the word being  
134599 expanded were “*\$x+\$y*” and *IFS=+*, the word would be split only if “*\$x*” or “*\$y*” contained  
134600 ‘+’; the ‘+’ in the original word was not generated by step (1).

134601 *IFS* is used for performing field splitting on the results of parameter and command substitution;  
134602 it is not used for splitting all fields. Earlier versions of the shell used it for splitting all fields  
134603 during field splitting, but this has severe problems because the shell can no longer parse its own  
134604 script. There are also important security implications caused by this behavior. All useful  
134605 applications of *IFS* use it for parsing input of the *read* utility and for splitting the results of



134606 parameter and command substitution.

134607 The rule concerning expansion to a single field requires that if **foo=abc** and **bar=def**, that:

134608 `"$foo"$bar"`

134609 expands to the single field:

134610 `abcdef`

134611 The rule concerning empty fields can be illustrated by:

134612 `$ unset foo`

134613 `$ set $foo bar ' ' xyz "$foo" abc`

134614 `$ for i`

134615 `> do`

134616 `> echo "-$i-"`

134617 `> done`

134618 **-bar-**

134619 **--**

134620 **-xyz-**

134621 **--**

134622 **-abc-**

134623 Step (1) indicates that parameter expansion, command substitution, and arithmetic expansion

134624 are all processed simultaneously as they are scanned. For example, the following is valid

134625 arithmetic:

134626 `x=1`

134627 `echo $(( $(echo 3)+$x ))`

134628 An early proposal stated that tilde expansion preceded the other steps, but this is not the case in

134629 known historical implementations; if it were, and if a referenced home directory contained a '\$'

134630 character, expansions would result within the directory name.

134631 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0003 [49,430] is applied.

134632 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

134633 Austin Group Defect 985 is applied, clarifying that quote removal is not always performed.

134634 Austin Group Defect 1038 is applied, clarifying that a '\$' that is followed by a <space>, <tab>,

134635 or a <newline>, or is not followed by any character, is treated as a literal character.

134636 Austin Group Defect 1123 is applied, clarifying the environment in which expansions are

134637 performed and requirements relating to empty fields.

134638 Austin Group Defect 1193 is applied, adding optional brace expansion.

134639 **C.2.6.1 Tilde Expansion**

134640 Tilde expansion generally occurs only at the beginning of words, but an exception based on

134641 historical practice has been included:

134642 `PATH=/posix/bin:~djk/bin`

134643 This is eligible for tilde expansion because <tilde> follows a <colon> and none of the relevant

134644 characters is quoted. Consideration was given to prohibiting this behavior because any of the

134645 following are reasonable substitutes:

134646 `PATH=$(printf %s ~karels/bin : ~bostic/bin)`

```

134647     for Dir in ~maat/bin ~srb/bin ...
134648     do
134649         PATH=${PATH:+$PATH:}$Dir
134650     done

```

134651 In the first command, explicit <colon> characters are used for each directory. In all cases, the  
 134652 shell performs tilde expansion on each directory because all are separate words to the shell.

134653 Note that expressions in operands such as:

```

134654     make -k mumble LIBDIR=~chet/lib

```

134655 do not qualify as shell variable assignments, and tilde expansion is not performed (unless the  
 134656 command does so itself, which *make* does not).

134657 Because of the requirement that the word is not quoted, the following are not equivalent; only  
 134658 the last causes tilde expansion:

```

134659     \~hlj/   ~h\lj/   ~"hlj"/   ~hlj\   ~hlj/

```

134660 In an early proposal, tilde expansion occurred following any unquoted <equals-sign> or  
 134661 <colon>, but this was removed because of its complexity and to avoid breaking commands such  
 134662 as:

```

134663     rcp hostname:~marc/.profile .

```

134664 System administrators on systems where // has an implementation-defined meaning which is  
 134665 different to /, should not create users with a home directory of / or //, since this may lead to  
 134666 unexpected filename resolution on those systems.

134667 A suggestion was made that the special sequence "\$~" should be allowed to force tilde  
 134668 expansion anywhere. Since this is not historical practice, it has been left for future  
 134669 implementations to evaluate. (The description in XCU [Section 2.2](#) (on page 2472) requires that a  
 134670 <dollar-sign> be quoted to represent itself, so the "\$~" combination is already unspecified.)

134671 The results of giving <tilde> with an unknown login name are undefined because the KornShell  
 134672 "~+" and "~-" constructs make use of this condition, but in general it is an error to give an  
 134673 incorrect login name with <tilde>. The results of having *HOME* unset are unspecified because  
 134674 some historical shells treat this as an error.

134675 Historically, the Korn shell performed field splitting and pathname expansion on the results of  
 134676 tilde expansion, and earlier versions of this standard reflected this. However, tilde expansion  
 134677 results in a pathname, and performing field splitting and pathname expansion on something  
 134678 that is already a pathname is at best redundant and at worst will change the value from the  
 134679 correct pathname to one or more incorrect ones. Later versions of the Korn shell do not perform  
 134680 these expansions and POSIX.1-2024 has been updated to match. Note that although pathname  
 134681 expansion is not performed on the results of tilde expansion, this does not prevent other parts of  
 134682 the same word from being expanded. For example, ~/a\* expands to all files in *\$HOME*  
 134683 beginning with 'a'.

134684 Austin Group Defect 1172 is applied, clarifying how quoting affects tilde expansion.

134685 Austin Group Defect 1632 is applied, clarifying the treatment of <slash> characters in tilde  
 134686 expansion.

134687 C.2.6.2 *Parameter Expansion*

134688 The rule for finding the closing '}' in "\${...}" is the one used in the KornShell and is  
 134689 upwardly-compatible with the Bourne shell, which does not determine the closing '}' until the  
 134690 word is expanded. The advantage of this is that incomplete expansions, such as:

```
134691 ${foo
```

134692 can be determined during tokenization, rather than during expansion.

134693 Quote removal is performed when assigning the value in the `parameter:=word` form of  
 134694 expansion in order that a subsequent expansion of the same parameter produces the same value  
 134695 as the original expansion. That is, the commands:

```
134696 unset parameter
134697 foo=${parameter:=word}
134698 bar=${parameter}
```

134699 assign the same value to `foo` and `bar`. A consequence of this is that the expansions  
 134700 `parameter:=word` and `parameter:-word` can produce different results for the same `word`.  
 134701 For example, with `parameter` unset or empty:

```
134702 ${parameter:-a\ b}
```

134703 expands to a single field "a b", whereas:

```
134704 ${parameter:=a\ b}
```

134705 expands to two fields 'a' and 'b' (because `parameter` is assigned the value "a b" before its  
 134706 value is substituted).

134707 For rationale regarding expansion of "\${...}" within double-quotes, see [Section C.2.2.3](#) (on  
 134708 page 3867).

134709 The string length and substring capabilities were included because of the demonstrated need for  
 134710 them, based on their usage in other shells, such as C shell and KornShell.

134711 Historical versions of the KornShell have not performed tilde expansion on the word part of  
 134712 parameter expansion; however, it is more consistent to do so.

134713 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0004 [458], XCU/TC1-2008/0005  
 134714 [458], XCU/TC1-2008/0006 [457], XCU/TC1-2008/0007 [457], XCU/TC1-2008/0008 [417],  
 134715 XCU/TC1-2008/0009 [457], XCU/TC1-2008/0010 [457], XCU/TC1-2008/0011 [457],  
 134716 XCU/TC1-2008/0012 [457], XCU/TC1-2008/0013 [457], XCU/TC1-2008/0014 [457],  
 134717 XCU/TC1-2008/0015 [457], XCU/TC1-2008/0016 [457], XCU/TC1-2008/0017 [457], and  
 134718 XCU/TC1-2008/0018 [458] are applied.

134719 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0013 [888] and XCU/TC2-2008/0014  
 134720 [867] are applied.

134721 Austin Group Defect 221 is applied, removing a statement about counting brace levels and  
 134722 clarifying that quote removal is performed when expanding `word` in `parameter:=word`.

134723 Austin Group Defect 985 is applied, clarifying when quote removal is performed.

134724 Austin Group Defect 1052 is applied, clarifying the description of string length expansion.

134725 Austin Group Defect 1268 is applied, removing text relating to parameter expansion inside  
 134726 double-quotes.

134727 Austin Group Defect 1478 is applied, making explicitly unspecified the results of parameter  
 134728 expansions that test whether the parameter '\*' or '@' is unset or null.

134729 Austin Group Defect 1491 is applied, restructuring a paragraph that used "Otherwise" after two

134730 conditions.

134731 Austin Group Defect 1561 is applied, clarifying that the varieties of parameter expansion that  
134732 provide for substring processing process parameter values as characters.

134733 C.2.6.3 *Command Substitution*

134734 The "\$ ()" form of command substitution solves a problem of inconsistent behavior when using  
134735 backquotes. For example:

134736

134737

134738

134739

Command	Output
echo '\\$x'	\\$x
echo `echo '\\$x'`	\$x
echo \$(echo '\\$x')	\\$x

134740 Additionally, the backquoted syntax has historical restrictions on the contents of the embedded  
134741 command. While the newer "\$ ()" form can process any kind of valid embedded script (with a  
134742 few caveats; see below), the backquoted form cannot handle some valid scripts that include  
134743 backquotes. For example, these otherwise valid embedded scripts do not work in the left  
134744 column, but do work on the right:

134745

134746

134747

134748

134749

134750

134751

134752

134753

134754

134755

```

echo `
cat <<\eof
a here-doc with `
eof
`
echo `
echo abc # a comment with `
`
echo `
echo ` `
`
echo $(
cat <<\eof
a here-doc with )
eof
)
echo $(
echo abc # a comment with )
)
echo $(
echo ` `
)

```

134756 Because of these inconsistent behaviors, the backquoted variety of command substitution is not  
134757 recommended for new applications that nest command substitutions or attempt to embed  
134758 complex scripts.

134759 The KornShell feature:

134760

134761

134762

If the *commands* string is of the form `<word>`, *word* is expanded to generate a pathname, and the value of the command substitution is the contents of this file with any trailing `<newline>` characters deleted.

134763

134764

134765

was omitted from the Shell and Utilities volume of POSIX.1-2024 because `$(cat word)` is an appropriate substitute. However, to prevent breaking numerous scripts relying on this feature, it is unspecified to have a script within "\$ ()" that has only redirections.

134766

134767

134768

In IEEE Std 1003.2-1992 the `$(commands)` form of command substitution only had unspecified behavior for a *commands* string consisting solely of redirections. However, two additional unspecified cases have since been added with relation to aliases:

134769

134770

134771

134772

1. Implementations are permitted to parse the entire *commands* string before executing any of it, and in this case *alias* and *unalias* commands in *commands* have no effect during parsing. For example, the following commands:

```
alias foo='echo "hello globe"'
```

134773 `echo $(alias foo='echo "Hello World"';foo)`

134774 produce the output "hello globe" if the *commands* string is executed as an entire  
134775 command and produce the output "Hello World" if the *commands* string is executed  
134776 incrementally.

134777 2. Although existing aliases are required to be expanded when the shell parses the input  
134778 that follows the "\$ (" in order to find the terminating ') ' (see [Section 2.3](#), on page 2475),  
134779 it is unspecified whether the terminating ') ' can result from alias substitution. For  
134780 example, with this script:

```
134781 alias foo="echo foo )"
134782 echo $(foo ; echo bar
```

134783 some shells output lines containing "foo" and "bar" whereas other shells report a  
134784 syntax error because they do not find a terminating ') ' for the command substitution.

134785 Arithmetic expansions have precedence over command substitutions. That is, if the shell can  
134786 parse an expansion beginning with "\$ (" as an arithmetic expansion then it will do so. It will  
134787 only parse the expansion as a command substitution (that starts with a subshell) if it determines  
134788 that it cannot parse the expansion as an arithmetic expansion. If the syntax is valid for neither  
134789 type of expansion, then it is unspecified what kind of syntax error the shell reports.

134790 How well the shell performs this determination is a quality of implementation issue. Current  
134791 shell implementations use heuristics. In particular, the shell need not evaluate nested expansions  
134792 when determining whether it can parse an expansion beginning with "\$ (" as an arithmetic  
134793 expansion. For example:

```
134794 $( (a $op b) )
```

134795 is always an arithmetic expansion if "\$op" expands to, say, '+', but if "\$op" expands to '( '  
134796 then the shell might still parse the expansion as an arithmetic expansion (resulting in a syntax  
134797 error due to unbalanced parentheses) or it might perform a command substitution.

134798 This standard requires that conforming applications always separate the "\$ (" and '( '  
134799 with white space when a command substitution starts with a subshell. This is because  
134800 implementations may support extensions in arithmetic expressions which could result in the  
134801 shell parsing the input as an arithmetic expansion even though a minimally conforming shell  
134802 would not. For example, many shells support arrays with the array index (which can be an  
134803 expression) in square brackets. Therefore, the presence of "myfile[0-9]" within an expansion  
134804 beginning "\$ (" is no guarantee that it will be parsed as a command substitution.

134805 The ambiguity is not restricted to the simple case of a single subshell. More complicated  
134806 ambiguous cases are possible (even with just the standard shell syntax), such as:

```
134807 $( ( cat <<EOH
134808 + ( (
134809 EOH
134810 ) && ( cat <<EOH
134811 ) ) + 1 +
134812 EOH
134813 ) )
```

134814 This can be parsed as an arithmetic expansion, with *cat* and *EOH* as the names of shell variables.  
134815 Ambiguous cases also exist where the end of the expansion is at a different location for the  
134816 arithmetic expansion and the command substitution:

```
134817 $( (cat <<EOF
134818 + ((( (
```

```

134819 EOF
134820 ) && (
134821 cat <<EOF
134822 +
134823 EOF
134824 ) )

```

134825 This is an incomplete arithmetic expansion, but would have been a (complete) command  
 134826 substitution if it could not have been parsed as an arithmetic expansion. If this expansion occurs  
 134827 at the end of input then the shell reports a syntax error; it does not parse it as a command  
 134828 substitution.

134829 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/4 is applied, changing the text from: "If a  
 134830 command substitution occurs inside double-quotes, it shall not be performed on the results of  
 134831 the substitution." to: "If a command substitution occurs inside double-quotes, field splitting and  
 134832 pathname expansion shall not be performed on the results of the substitution.". The  
 134833 replacement text taken from the ISO POSIX-2:1993 standard is clearer about the items that are  
 134834 not performed.

134835 SD5-XCU-ERN-84 is applied, clarifying how the search for the matching backquote is satisfied.

134836 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0019 [217] is applied.

134837 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

134838 Austin Group Defect 953 is applied, clarifying how the commands in command substitutions are  
 134839 parsed.

134840 Austin Group Defect 1015 is applied, clarifying the handling of <backslash> when a backquoted  
 134841 command substitution is within double-quotes.

134842 Austin Group Defect 1268 is applied, removing text relating to command substitution inside  
 134843 double-quotes.

134844 Austin Group Defect 1342 is applied, clarifying the requirements for alias substitutions inside  
 134845 command substitutions.

134846 Austin Group Defect 1560 is applied, clarifying that the standard output of the command(s) in a  
 134847 command substitution is treated as a sequence of bytes.

#### 134848 C.2.6.4 Arithmetic Expansion

134849 The standard developers agreed that there was a strong desire for some kind of arithmetic  
 134850 evaluator to provide functionality similar to *expr*, that relating it to '\$' makes it work well with  
 134851 the standard shell language and provides access to arithmetic evaluation in places where  
 134852 accessing a utility would be inconvenient.

134853 The syntax and semantics for arithmetic were revised for the ISO/IEC 9945-2:1993 standard.  
 134854 The language represents a simple subset of the previous arithmetic language (which was  
 134855 derived from the KornShell "(())" construct). The syntax was changed from that of a  
 134856 command denoted by *((expression))* to an expansion denoted by *\$(expression)*. The new form is  
 134857 a dollar expansion ('\$') that evaluates the expression and substitutes the resulting value.  
 134858 Objections to the previous style of arithmetic included that it was too complicated, did not fit in  
 134859 well with the use of variables in the shell, and its syntax conflicted with subshells. The  
 134860 justification for the new syntax is that the shell is traditionally a macro language, and if a new  
 134861 feature is to be added, it should be accomplished by extending the capabilities presented by the  
 134862 current model of the shell, rather than by inventing a new one outside the model; adding a new  
 134863 dollar expansion was perceived to be the most intuitive and least destructive way to add such a

134864 new capability.

134865 The standard requires assignment operators to be supported (as listed in XCU [Section 1.1.2](#), on  
134866 page 2457), and since arithmetic expansions are not specified to be evaluated in a subshell  
134867 environment, changes to variables there have to be in effect after the arithmetic expansion, just  
134868 as in the parameter expansion "\$ {x=value} ".

134869 Note, however, that "\$ ( ( x=5 ) )" need not be equivalent to "\$ ( ( \$x=5 ) )". If the value of  
134870 the environment variable *x* is the string "y=", the expansion of "\$ ( ( x=5 ) )" would set *x* to 5  
134871 and output 5, but "\$ ( ( \$x=5 ) )" would output 0 if the value of the environment variable *y* is  
134872 not 5 and would output 1 if the environment variable *y* is 5. Similarly, if the value of the  
134873 environment variable is 4, the expansion of "\$ ( ( x=5 ) )" would still set *x* to 5 and output 5,  
134874 but "\$ ( ( \$x=5 ) )" (which would be equivalent to "\$ ( ( 4=5 ) )") would yield a syntax  
134875 error.

134876 In early proposals, a form `$(expression)` was used. It was functionally equivalent to the "\$ ( ) "  
134877 of the current text, but objections were lodged that the 1988 KornShell had already implemented  
134878 "\$ ( ) " and there was no compelling reason to invent yet another syntax. Furthermore, the  
134879 "\$ [ ] " syntax had a minor incompatibility involving the patterns in **case** statements.

134880 The portion of the ISO C standard arithmetic operations selected corresponds to the operations  
134881 historically supported in the KornShell. In addition to the exceptions listed in XCU [Section 2.6.4](#)  
134882 (on page 2490), the use of the following are explicitly outside the scope of the rules defined in  
134883 XCU [Section 1.1.2.1](#) (on page 2457):

- 134884 • The prefix operator '&' and the "[ ]", "->", and '.' operators.
- 134885 • Casts

134886 It was concluded that the *test* command (`(l)`) was sufficient for the majority of relational arithmetic  
134887 tests, and that tests involving complicated relational expressions within the shell are rare, yet  
134888 could still be accommodated by testing the value of "\$ ( ) " itself. For example:

```
134889 # a complicated relational expression
134890 while [ $(( ($x + $y)/($a * $b)) < ($foo*$bar) )) -ne 0 ]
```

134891 or better yet, the rare script that has many complex relational expressions could define a  
134892 function like this:

```
134893 val() {
134894     return $(!$1)
134895 }
```

134896 and complicated tests would be less intimidating:

```
134897 while val $(( ($x + $y)/($a * $b)) < ($foo*$bar) ))
134898 do
134899     # some calculations
134900 done
```

134901 A suggestion that was not adopted was to modify *true* and *false* to take an optional argument,  
134902 and *true* would exit true only if the argument was non-zero, and *false* would exit false only if the  
134903 argument was non-zero:

```
134904 while true $(( $x > 5 && $y <= 25 ))
```

134905 There is a minor portability concern with the new syntax. The example "\$ ( ( 2+2 ) )" could have  
134906 been intended to mean a command substitution of a utility named "2+2" in a subshell. The  
134907 standard developers considered this to be obscure and isolated to some KornShell scripts  
134908 (because "\$ ( ) " command substitution existed previously only in the KornShell). The text on

134909 command substitution requires that the "\$(" and '( ' be separate tokens if this usage is  
134910 needed.

134911 An example such as:

```
134912 echo $((echo hi);(echo there))
```

134913 should not be misinterpreted by the shell as arithmetic because attempts to balance the  
134914 parentheses pairs would indicate that they are subshells. However, as indicated by XBD [Section](#)  
134915 [3.85](#) (on page 44), a conforming application must separate two adjacent parentheses with white  
134916 space to indicate nested subshells.

134917 The standard is intentionally silent about how a variable's numeric value in an expression is  
134918 determined from its normal "sequence of bytes" value. It could be done as a text substitution, as  
134919 a conversion like that performed by *strtol()*, or even recursive evaluation. Therefore, the only  
134920 cases for which the standard is clear are those for which both conversions produce the same  
134921 result. The cases where they give the same result are those where the sequence of bytes form a  
134922 valid integer constant. Therefore, if a variable does not contain a valid integer constant, the  
134923 behavior is unspecified.

134924 For the commands:

```
134925 x=010; echo $((x += 1))
```

134926 the output must be 9.

134927 For the commands:

```
134928 x=' 1'; echo $((x += 1))
```

134929 the results are unspecified.

134930 For the commands:

```
134931 x=1+1; echo $((x += 1))
```

134932 the results are unspecified.

134933 Although the ISO C standard requires support for **long long** and allows extended integer types  
134934 with higher ranks, POSIX.1-2024 only requires arithmetic expansions to support **signed long**  
134935 integer arithmetic. Implementations are encouraged to support signed integer values at least as  
134936 large as the size of the largest file allowed on the implementation.

134937 Implementations are also allowed to perform floating-point evaluations as long as an  
134938 application won't see different results for expressions that would not overflow **signed long**  
134939 integer expression evaluation. (This includes appropriate truncation of results to integer values.)

134940 Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement  
134941 that the integer constant suffixes **l** and **L** had to be recognized. The ISO POSIX-2:1993 standard  
134942 did not require the **u**, **u1**, **uL**, **U**, **U1**, **UL**, **lu**, **lU**, **Lu**, and **LU** suffixes since only signed integer  
134943 arithmetic was required. Since all arithmetic expressions were treated as handling **signed long**  
134944 integer types anyway, the **l** and **L** suffixes were redundant. No known scripts used them and  
134945 some historic shells did not support them. When the ISO/IEC 9899:1999 standard was used as  
134946 the basis for the description of arithmetic processing, the **ll** and **LL** suffixes and combinations  
134947 were also not required. Implementations are still free to accept any or all of these suffixes, but  
134948 are not required to do so.

134949 There was also some confusion as to whether the shell was required to recognize character  
134950 constants. Syntactically, character constants were required to be recognized, but the  
134951 requirements for the handling of <backslash> and single-quote characters (needed to specify  
134952 character constants) within an arithmetic expansion were ambiguous. Furthermore, no known



134953 shells supported them. Changes made in response to IEEE PASC Interpretation 1003.2 #208  
 134954 removed the requirement to support them (if they were indeed required before). POSIX.1-2024  
 134955 clearly does not require support for character constants.

134956 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/3 is applied, clarifying arithmetic  
 134957 expressions.

134958 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0020 [50] is applied.

134959 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0015 [584] is applied.

#### 134960 C.2.6.5 *Field Splitting*

134961 The operation of field splitting using *IFS*, as described in early proposals, was based on the way  
 134962 the KornShell splits words, but it is incompatible with other common versions of the shell.  
 134963 However, each has merit, and so a decision was made to allow both. If the *IFS* variable is unset  
 134964 or is `<space><tab><newline>`, the operation is equivalent to the way the System V shell splits  
 134965 words. Using characters outside the `<space><tab><newline>` set yields the KornShell behavior,  
 134966 where each of the non-`<space><tab><newline>`s is significant. This behavior, which affords the  
 134967 most flexibility, was taken from the way the original *awk* handled field splitting.

134968 The different handling of white space and non-white-space characters in *IFS* can be summarized  
 134969 as a pseudo-ERE:

134970  $(s^*ns^* | s^+)$

134971 where *s* is an *IFS* white-space character and *n* is a character in the *IFS* that is not white space.  
 134972 Any string matching that ERE delimits a field, except that the *s+* form does not delimit fields at  
 134973 the beginning or the end of a line. For example, if *IFS* is `<space>/<comma>/<tab>`, the string:

134974 `<space><space>red<space><space>, <space>white<space>blue`

134975 yields the three colors as the delimited fields.

134976 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0016 [832] is applied.

134977 Austin Group Defect 1123 is applied, clarifying the requirements if no fields are delimited.

134978 Austin Group Defect 1560 is applied, clarifying that the results of word expansions are treated as  
 134979 a sequences of bytes when searching for (bytes that form) *IFS* characters.

134980 Austin Group Defect 1649 is applied, clarifying how field splitting is performed.

#### 134981 C.2.6.6 *Pathname Expansion*

134982 There is no additional rationale provided for this section.

#### 134983 C.2.6.7 *Quote Removal*

134984 The golden rule in quote removal is that if a quote character was treated as special in the original  
 134985 word, it is removed; if it was treated as a literal character, it is not removed.

134986 Austin Group Defect 221 is applied, clarifying the conditions under which quote characters are,  
 134987 or are not, removed.

134988 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

134989 **C.2.7 Redirection**

134990 In the System Interfaces volume of POSIX.1-2024, file descriptors are integers in the range  
134991 0–(`OPEN_MAX`–1). The file descriptors discussed in XCU [Section 2.7](#) (on page 2493) are that  
134992 same set of small integers.

134993 Having multi-digit file descriptor numbers for I/O redirection can cause some obscure  
134994 compatibility problems. Specifically, scripts that depend on an example command:

```
134995 echo 22>/dev/null
```

134996 echoing "2" to standard error or "22" to standard output are no longer portable. However, the  
134997 file descriptor number must still be delimited from the preceding text. For example:

```
134998 cat file2>foo
```

134999 writes the contents of **file2**, not the contents of **file**.

135000 The limitation to 9 file descriptors is overcome in some shells via a form of redirection whereby a  
135001 shell variable stores the file descriptor number. For example:

```
135002 exec {fdvar}> foo
```

135003 opens the file `foo` on a file descriptor greater than 9 and stores the file descriptor number in  
135004 shell variable `fdvar`. (This can later be closed using `exec {fdvar}>&-`.) This form of  
135005 redirection is allowed but not required by this standard.

135006 The "`>|`" format of output redirection was adopted from the KornShell. Along with the  
135007 *noclobber* option, set `-C`, it provides a safety feature to prevent inadvertent overwriting of  
135008 existing files. (See the RATIONALE for the *pathchk* utility for why this step was taken.) The  
135009 restriction on regular files is historical practice.

135010 The System V shell and the KornShell have differed historically on pathname expansion of *word*;  
135011 the former never performed it, the latter only when the result was a single field (file). As a  
135012 compromise, it was decided that the KornShell functionality was useful, but only as a shorthand  
135013 device for interactive users. No reasonable shell script would be written with a command such  
135014 as:

```
135015 cat foo > a*
```

135016 Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with  
135017 which they are most comfortable.

135018 The construct "`2>&1`" is often used to redirect standard error to the same file as standard  
135019 output. Since the redirections take place beginning to end, the order of redirections is significant.  
135020 For example:

```
135021 ls > foo 2>&1
```

135022 directs both standard output and standard error to file **foo**. However:

```
135023 ls 2>&1 > foo
```

135024 only directs standard output to file **foo** because standard error was duplicated as standard  
135025 output before standard output was directed to file **foo**.

135026 Applications should not use the `[n]<&-` or `[n]>&-` operators to execute a utility or application  
135027 with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, as  
135028 this might cause the executed program (or shell built-in) to misbehave. In order not to pass on  
135029 these file descriptors to an executed utility or application, applications should not just close  
135030 them but should reopen them on, for example, `/dev/null`. Some implementations may reopen  
135031 them automatically, but applications should not rely on this being done.

135032 The "<>" operator could be useful in writing an application that worked with several terminals,  
 135033 and occasionally wanted to start up a shell. That shell would in turn be unable to run  
 135034 applications that run from an ordinary controlling terminal unless it could make use of "<>"  
 135035 redirection. The specific example is a historical version of the pager *more*, which reads from  
 135036 standard error to get its commands, so standard input and standard output are both available  
 135037 for their usual usage. There is no way of saying the following in the shell without "<>":

```
135038 cat food | more - >/dev/tty03 2<>/dev/tty03
```

135039 Another example of "<>" is one that opens **/dev/tty** on file descriptor 3 for reading and writing:

```
135040 exec 3<> /dev/tty
```

135041 An example of creating a lock file for a critical code region:

```
135042 set -C
135043 until 2> /dev/null > lockfile
135044 do sleep 30
135045 done
135046 set +C
135047 perform critical function
135048 rm lockfile
```

135049 Since **/dev/null** is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

135050 Tilde expansion is not performed on a here-document because the data is treated as if it were  
 135051 enclosed in double-quotes.

135052 Austin Group Defect 1193 is applied, adding the optional redirection form `{location}redir-`  
 135053 `op word`.

135054 Austin Group Defect 1232 is applied, clarifying the allowed behaviors in an interactive shell  
 135055 when pathname expansion on the word following a redirection operator would result in more  
 135056 than one word.

135057 Austin Group Defect 1493 is applied, moving some information from this section to the  
 135058 definition of "file descriptor" in XBD [Section 3.141](#) (on page 51).

#### 135059 C.2.7.1 *Redirecting Input*

135060 There is no additional rationale provided for this section.

#### 135061 C.2.7.2 *Redirecting Output*

135062 Earlier versions of this standard did not require redirection using '>' when *noclobber* is set to  
 135063 perform the file creation step as an atomic operation. Historical shells just called *stat()* to check  
 135064 if a regular file existed and then called *creat()*. The operation thus involved a race condition  
 135065 which meant that it could not be used for reliable creation of lock files. Many shell  
 135066 implementations improved on this by using *open()* with the `O_CREAT` and `O_EXCL` flags set as  
 135067 one step in a multi-step process which still meant that an existing non-regular file (for example  
 135068 **/dev/null**, **/dev/tty**, or a FIFO) was opened successfully. However, the methods employed still  
 135069 involved a race condition and could produce misleading diagnostics if there is concurrent  
 135070 creation or removal of files.

135071 An ideal solution would be an `O_NOCLOBBER` flag for *open()* which the shell could use in  
 135072 order to perform the entire operation atomically, and implementations are encouraged to adopt  
 135073 this solution, adding the flag as described in the FUTURE DIRECTIONS section of *open()* (on  
 135074 page 1515), and using it in the implementation's POSIX shell and in other shells. Authors of

135075 portable shells should make use of `#ifdef O_NOCLOSE` so that it is used on  
135076 implementations that provide it.

135077 If `O_NOCLOSE` is not used, shells can use one of the following methods:

135078 1. The “`stat` first” method.

135079 a. Call `stat()` and if the file exists and is a regular file, the redirection fails. Otherwise:

135080 b. Call `open()` without `O_CREAT` or `O_TRUNC` to open an existing file. If the open  
135081 succeeds, use `fstat()` to check whether the opened file is a regular file. If it is, close  
135082 it and fail the redirection. If it is a non-regular file, the redirection succeeds.  
135083 Otherwise:

135084 c. Call `open()` with `O_CREAT|O_EXCL`. The redirection succeeds or fails depending  
135085 on whether the open succeeds or fails.

135086 2. The “exclusive create first” method.

135087 a. Call `open()` with `O_CREAT|O_EXCL`. If the open succeeds, the redirection  
135088 succeeds. If the open fails with `[EMFILE]` or `[ENFILE]`, use `stat()` to check whether  
135089 a regular file exists; if it does, fail the redirection. Otherwise:

135090 b. Call `open()` without `O_CREAT` or `O_TRUNC` to open an existing file. If the open  
135091 succeeds, use `fstat()` to check whether the opened file is a regular file. If it is, close  
135092 it and fail the redirection. If it is a non-regular file, the redirection succeeds. If the  
135093 second open fails, the redirection fails with a diagnostic based on the `errno` value  
135094 set by the first open.

135095 (A minor variation of this method could also be used whereby step 2.b is only done if the  
135096 `open()` in step 2.a fails with `[EEXIST]`.)

135097 Method 1 is in widespread use. Method 2 has not been observed exactly as described, although  
135098 an implementation which omits the `stat()` in step 2.a has been observed. Without the `stat()`, this  
135099 method has a problem in that if a regular file exists but the `open()` fails with `[EMFILE]` or  
135100 `[ENFILE]` instead of `[EEXIST]` (which is to be expected if those conditions exist, because  
135101 detecting `[EEXIST]` is more expensive), then the shell will give an incorrect diagnostic.  
135102 (Reporting that no file descriptors are available implies that a non-regular file exists, because the  
135103 shell tried to open the file and it is not supposed to open an existing regular file.)

135104 A variant of method 1 which omits the initial `stat()` call has also been observed; this has the  
135105 same problem with `[EMFILE]` and `[ENFILE]`. With the `stat()`, this misleading diagnostic can also  
135106 happen, but only if a regular file is created in the timing window between steps 1.a and 1.b,  
135107 which makes it an allowed case. (The standard allows a misleading diagnostic when there is  
135108 concurrent creation or removal of files.)

135109 Both methods have cases where a misleading diagnostic is given when a non-regular file is  
135110 concurrently created or removed. With method 1 it occurs if no file exists at steps 1.a and 1.b,  
135111 and a non-regular file is created before step 1.c. With method 2 it occurs if a non-regular file  
135112 exists at step 2.a and is removed before step 2.b. (In both cases, the diagnostic misleadingly  
135113 implies that a regular file exists).

135114 Both methods differ from historical shell behavior in that the redirection fails if there is an  
135115 existing symbolic link whose target does not exist, instead of the link’s target being created as a  
135116 regular file. The standard developers consider reliable lock file creation to be more important  
135117 than the creation of symbolic link targets.

135118 Creation of lock files and unique (often temporary) files with `noclobber` set is only reliable  
135119 provided neither non-regular files nor symbolic links to non-regular files exist or are created in  
135120 the same directory with the same names, and no other processes delete the files while still in use.

- 135121 If a directory such as `/tmp` is used for lock files, then another process could accidentally or  
135122 maliciously create a FIFO (or a special file, given sufficient privilege) with the same name,  
135123 causing multiple processes to simultaneously open the same lock file instead of one succeeding  
135124 and the others failing.
- 135125 Austin Group Defects 1016 and 1364 are applied, changing the requirements when the `noclobber`  
135126 option is set.
- 135127 C.2.7.3 *Appending Redirected Output*
- 135128 Note that when a file is opened (even with the `O_APPEND` flag set), the initial file offset for that  
135129 file is set to the beginning of the file. Some historic shells set the file offset to the current end-of-  
135130 file when **append** mode shell redirection was used, but this is not allowed by POSIX.1-2024.
- 135131 Austin Group Defect 1016 is applied, changing ``with the `O_APPEND` flag`` to ``with the  
135132 `O_APPEND` flag set``.
- 135133 C.2.7.4 *Here-Document*
- 135134 Historical shell behavior was to treat the end of input as being equivalent to the delimiter of a  
135135 here-document, terminating the here-document, usually without any indication, and continuing  
135136 as if the delimiter had been recognized. This can cause problems where the delimiter had been  
135137 intended to occur much earlier in the script, but was incorrectly entered—a mistake which for  
135138 many other errors would have resulted in a syntax error, and an aborted script, instead simply  
135139 generates incorrect results. Because of this some shell implementations have changed to  
135140 reporting an undelimited here-document as a syntax error. Other implementations are  
135141 encouraged to do the same.
- 135142 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0017 [890], XCU/TC2-2008/0018  
135143 [583], and XCU/TC2-2008/0019 [580] are applied.
- 135144 Austin Group Defect 1036 is applied, clarifying how here-documents are parsed.
- 135145 Austin Group Defect 1411 is applied, adding a paragraph break.
- 135146 C.2.7.5 *Duplicating an Input File Descriptor*
- 135147 The file descriptor duplication redirection operators, `[n]<&word` and `[n]>&word`, make a copy  
135148 of one file descriptor as another. If the operation is successful, the new file descriptor has the  
135149 same access mode as the source (old) file descriptor, because the access mode is determined by  
135150 the open file description to which both file descriptors point. To avoid a redirection error,  
135151 applications need to ensure that they use the appropriate redirection operator for the access  
135152 mode of the file descriptor being duplicated.
- 135153 Austin Group Defect 1536 is applied, making it optional whether attempting to duplicate an  
135154 open file descriptor that is not open for input results in a redirection error.
- 135155 C.2.7.6 *Duplicating an Output File Descriptor*
- 135156 See [Section C.2.7.5](#).
- 135157 Austin Group Defect 1536 is applied, making it optional whether attempting to duplicate an  
135158 open file descriptor that is not open for output results in a redirection error.

135159 C.2.7.7 *Open File Descriptors for Reading and Writing*

135160 There is no additional rationale provided for this section.

## 135161 C.2.8 Exit Status and Errors

135162 There is no additional rationale provided for this section.

### 135163 C.2.8.1 *Consequences of Shell Errors*

135164 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0020 [882] and XCU/TC2-2008/0021  
135165 [717,882] are applied.

135166 Austin Group Defect 914 is applied, requiring that the shell does not exit when a redirection  
135167 error occurs with compound commands or with function execution.

135168 Austin Group Defect 1427 is applied, changing this section to account for the effect of the  
135169 *command* utility when it is used to execute a special built-in utility.

135170 Austin Group Defect 1629 is applied, requiring that the shell exits if an unrecoverable read error  
135171 occurs when reading commands.

### 135172 C.2.8.2 *Exit Status for Commands*

135173 There is a historical difference in *sh* and *ksh* non-interactive error behavior. When a command  
135174 named in a script is not found, some implementations of *sh* exit immediately, but *ksh* continues  
135175 with the next command. Thus, the Shell and Utilities volume of POSIX.1-2024 says that the shell  
135176 “may” exit in this case. This puts a small burden on the programmer, who has to test for  
135177 successful completion following a command if it is important that the next command not be  
135178 executed if the previous command was not found. If it is important for the command to have  
135179 been found, it was probably also important for it to complete successfully. The test for successful  
135180 completion would not need to change.

135181 Historically, shells have returned an exit status of  $128+n$ , where  $n$  represents the signal number.  
135182 Since signal numbers are not standardized, there is no portable way to determine which signal  
135183 caused the termination. Also, it is possible for a command to exit with a status in the same range  
135184 of numbers that the shell would use to report that the command was terminated by a signal.  
135185 Implementations are encouraged to choose exit values greater than 256 to indicate programs that  
135186 terminate by a signal so that the exit status cannot be confused with an exit status generated by a  
135187 normal termination. However, the use of exit values greater than 256 poses a problem for the  
135188 shell’s own exit status. Historically this was the exit status of the last command invoked by the  
135189 shell, but if the last command was terminated by a signal and was assigned an exit status greater  
135190 than 256 by the shell, this value would be truncated to eight bits in the shell’s exit status.  
135191 Likewise truncation would occur with use of

135192 `exit $?`

135193 `or`

135194 `ret=$?`

135195 `....`

135196 `exit $ret`

135197 in shell scripts. To avoid this truncation, shells which assign exit statuses greater than 256 are  
135198 required to propagate the wait status of the last command to the shell’s own wait status (by  
135199 sending itself the same signal), and to handle exit values greater than 256 passed to the *exit*

135200 builtin by mimicking the wait status that would give rise to assignment of that exit status in the  
 135201 shell. Note that this requirement does not apply to signals that do not cause termination, such as  
 135202 SIGCHLD, since the shell can never actually assign a corresponding exit status greater than 256,  
 135203 and the requirement is worded in terms of this assignment.

135204 Historical shells make the distinction between “utility not found” and “utility found but cannot  
 135205 execute” in their error messages. By specifying two seldomly used exit status values for these  
 135206 cases, 127 and 126 respectively, this gives an application the opportunity to make use of this  
 135207 distinction without having to parse an error message that would probably change from locale to  
 135208 locale. The *command*, *env*, *nohup*, and *xargs* utilities in the Shell and Utilities volume of  
 135209 POSIX.1-2024 have also been specified to use this convention.

135210 When a command fails during word expansion or redirection, most historical implementations  
 135211 exit with a status of 1. However, there was some sentiment that this value should probably be  
 135212 much higher so that an application could distinguish this case from the more normal exit status  
 135213 values. Thus, the language “greater than zero” was selected to allow either method to be  
 135214 implemented.

135215 If a C application calls `exit(256)`, the command’s exit status in the shell becomes zero due to  
 135216 the modulo 256 operation. Since zero is interpreted as “true” or “success” for `if` statements,  
 135217 AND and OR lists, `set -e`, and so on, applications should be careful to avoid exiting with a  
 135218 value that is a multiple of 256 unless the value is intended to be interpreted as true or success.

135219 To avoid ambiguity caused by the modulo 256 operation, applications are encouraged to avoid  
 135220 using a count or the result of a computation as the exit value unless the value is guaranteed to be  
 135221 non-negative and less than 256.

135222 The ambiguity caused by the modulo 256 operation is unfortunate, but required due to historical  
 135223 implementation behavior. A future version of this standard may change the definition of exit  
 135224 status to remove the modulo 256 requirement and use all bits of the value passed to `exit()` (or  
 135225 equivalent), and may introduce a way to select whether the special parameter ‘?’ contains the  
 135226 exit status modulo 256 or the full exit status.

135227 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0022 [717] is applied.

135228 Austin Group Defect 51 is applied, clarifying the exit status when a command is terminated due  
 135229 to the receipt of a signal.

135230 Austin Group Defect 947 is applied, clarifying the exit status of commands.

## 135231 C.2.9 Shell Commands

135232 A description of an “empty command” was removed from an early proposal because it is only  
 135233 relevant in the cases of `sh -c ""`, `system("")`, or an empty shell-script file (such as the  
 135234 implementation of *true* on some historical systems). Since it is no longer mentioned in the Shell  
 135235 and Utilities volume of POSIX.1-2024, it falls into the silently unspecified category of behavior  
 135236 where implementations can continue to operate as they have historically, but conforming  
 135237 applications do not construct empty commands. (However, note that *sh* does explicitly state an  
 135238 exit status for an empty string or file.) In an interactive session or a script with other commands,  
 135239 extra <newline> or <semicolon> characters, such as:

```
135240 $ false
135241 $
135242 $ echo $?
135243 1
```

135244 would not qualify as the empty command described here because they would be consumed by

- 135245 other parts of the grammar.
- 135246 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0023 [473] is applied.
- 135247 C.2.9.1 *Simple Commands*
- 135248 Austin Group Defect 1224 is applied, correcting a mismatch between the description of simple  
135249 commands and the formal *simple\_command* grammar.
- 135250 Austin Group Defect 1227 is applied, inserting additional subsection headings.
- 135251 **Order of Processing**
- 135252 The enumerated list is used only when the command is actually going to be executed. For  
135253 example, in:
- 135254 `true || $foo *`
- 135255 no expansions are performed.
- 135256 Expansion of words in an assignment context following the command name can only occur for  
135257 declaration utilities, and only when the word can be used as a variable assignment in isolation.
- 135258 For example, this code sequence exports the single variable *a* with the value "1 b=2", but  
135259 invokes *make* with the macro *a* set to '1' and *b* set to '2', since *make* is not a declaration utility:
- 135260 `set '1 b=2'`  
135261 `export a=$1`  
135262 `make a=$1`
- 135263 Conversely, this code sequence exports two variables, *a* set to '1' and *b* set to '2', because the  
135264 use of quoting means that the word could not be recognized as a variable assignment, and  
135265 regular expansion rules require that field splitting occurs on the unquoted expansion of \$1:
- 135266 `set '1 b=2'`  
135267 `export \a=$1`
- 135268 Likewise, this code sequence will not be parsed in assignment context, but is still required to  
135269 export the variable named *foo* with the value '1':
- 135270 `var=foo`  
135271 `export $var=1`
- 135272 Implementations are permitted to provide extensions that serve as declaration utilities, such as  
135273 *typeset* or *local*, or even a way to define a function that can behave as a declaration utility.
- 135274 Declaration utilities are only required to be recognized via lexical analysis; if any expansions are  
135275 required before the command name is known, or before the first argument to the *command* utility  
135276 is known, then it is unspecified whether subsequent arguments will be treated with an  
135277 assignment context during expansion. For example, it is unspecified whether
- 135278 `var=export; $var a=~`
- 135279 sets the variable *a* to a literal <tilde> or to the value of \$HOME, since lexical analysis sees "\$var"  
135280 rather than "export" as the command name.
- 135281 Austin Group Defects 351 and 1535 are applied, adding requirements relating to declaration  
135282 utilities.



135283 **Variable Assignments**

135284 The following example illustrates both how a variable assignment without a command name  
 135285 affects the current execution environment, and how an assignment with a command name only  
 135286 affects the execution environment of the command:

```
135287 $ x=red
135288 $ echo $x
135289 red
135290 $ export x
135291 $ sh -c 'echo $x'
135292 red
135293 $ x=blue sh -c 'echo $x'
135294 blue
135295 $ echo $x
135296 red
```

135297 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0021 [255] is applied.

135298 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0024 [654] is applied.

135299 Austin Group Defect 1009 is applied, clarifying the behavior when a special built-in utility is  
 135300 executed with a variable assignment.

135301 **Commands with no Command Name**

135302 This next example illustrates that redirections without a command name are still performed:

```
135303 $ ls foo
135304 ls: foo: no such file or directory
135305 $ > foo
135306 $ ls foo
135307 foo
```

135308 A command without a command name, but one that includes a command substitution, has an  
 135309 exit status of the last command substitution that the shell performed. For example:

```
135310 if      x=$(command)
135311 then    ...
135312 fi
```

135313 An example of redirections without a command name being performed in a subshell shows that  
 135314 the here-document does not disrupt the standard input of the **while** loop:

```
135315 IFS=:
135316 while read a b
135317 do     echo $a
135318       <<-eof
135319       Hello
135320       eof
135321 done </etc/passwd
```

135322 Following are examples of commands without command names in AND-OR lists:

```
135323 > foo || {
135324     echo "error: foo cannot be created" >&2
135325     exit 1
135326 }
135327 # set saved if /vmunix.save exists
```

135328 `test -f /vmunix.save && saved=1`

135329 Command substitution and redirections without command names both occur in subshells, but  
135330 they are not necessarily the same ones. For example, in:

135331 `exec 3> file`  
135332 `var=$(echo foo >&3) 3>&1`

135333 it is unspecified whether **foo** is echoed to the file or to standard output.

135334 Austin Group Defect 1150 is applied, clarifying the exit status of a command that has no  
135335 command name and has more than one command substitution.

### 135336 **Command Search and Execution**

135337 This description requires that the shell can execute shell scripts directly, even if the underlying  
135338 system does not support the common "#!" interpreter convention. That is, if file **foo** contains  
135339 shell commands and is executable, the following executes **foo**:

135340 `./foo`

135341 The command search shown here does not match all historical implementations. A more typical  
135342 sequence has been:

- 135343 • Any built-in (special or regular)
- 135344 • Functions
- 135345 • Path search for executable files

135346 But there are problems with this sequence. Since the programmer has no idea in advance which  
135347 utilities might have been built into the shell, a function cannot be used to override portably a  
135348 utility of the same name. (For example, a function named *cd* cannot be written for many  
135349 historical systems.) Furthermore, the *PATH* variable is partially ineffective in this case, and only  
135350 a pathname with a <slash> can be used to ensure a specific executable file is invoked.

135351 After the *execve()* failure described, the shell normally executes the file as a shell script. Some  
135352 implementations, however, attempt to detect whether the file is actually a script and not an  
135353 executable from some other architecture. The method used by the KornShell is allowed by the  
135354 text that indicates non-text files may be bypassed.

135355 The sequence selected for the Shell and Utilities volume of POSIX.1-2024 acknowledges that  
135356 special built-ins cannot be overridden, but gives the programmer full control over which  
135357 versions of other utilities are executed (with some exceptions). It provides a means of  
135358 suppressing function lookup (via the *command* utility) for the user's own functions and, with the  
135359 exception of the intrinsic utilities (see XCU Section 1.7, on page 2470), ensures that any regular  
135360 built-ins or functions provided by the implementation are under the control of the path search.  
135361 The mechanisms for associating non-intrinsic built-ins or functions with executable files in the  
135362 path are not specified by the Shell and Utilities volume of POSIX.1-2024, but the wording  
135363 requires that if either is implemented, the application is not able to distinguish a function or  
135364 built-in from an executable (other than in terms of performance, presumably). The  
135365 implementation ensures that all effects specified by the Shell and Utilities volume of  
135366 POSIX.1-2024 resulting from the invocation of the regular built-in or function (interaction with  
135367 the environment, variables, traps, and so on) are identical to those resulting from the invocation  
135368 of an executable file.

135369 Various historical implementations have used the names in item 1.b. as built-ins or reserved  
135370 words. This standard does not specify their behavior, but their existence means that it is  
135371 important for portable applications to avoid giving functions (or utilities in *PATH*) those names  
135372 because the function (or utility in *PATH*) might not be executed as expected.

135373 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/4 is applied, updating the case where  
135374 *execve*() fails due to an error equivalent to the [ENOEXEC] error.

135375 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0022 [168], XCU/TC1-2008/0023  
135376 [168], XCU/TC1-2008/0024 [168], XCU/TC1-2008/0025 [168], XCU/TC1-2008/0026 [168,430],  
135377 XCU/TC1-2008/0027 [168,430], and XCU/TC1-2008/0028 [173] are applied.

135378 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0025 [935] and XCU/TC2-2008/0026  
135379 [705] are applied.

135380 Austin Group Defect 465 is applied, adding *compound*, *enum*, *float*, *integer*, and *nameref* to the  
135381 table of command names for which the results are unspecified.

135382 Austin Group Defect 854 is applied, adding intrinsic utilities.

135383 Austin Group Defect 1391 is applied, clarifying the execution of a standard utility provided by  
135384 the implementation in the form of a function.

### 135385 Standard File Descriptors

135386 There is no additional rationale provided for this section.

### 135387 Non-built-in Utility Execution

135388 Austin Group Defect 1157 is applied, clarifying the execution of non-built-in utilities.

135389 Austin Group Defects 1226 and 1435 are applied, clarifying the circumstances under which the  
135390 shell may bypass execution of a non-built-in utility as a shell script.

### 135391 Examples

135392 Consider three versions of the *ls* utility:

- 135393 1. The application includes a shell function named *ls*.
- 135394 2. The user writes a utility named *ls* and puts it in **/fred/bin**.
- 135395 3. The example implementation provides *ls* as a regular shell built-in that is invoked (either  
135396 by the shell or directly by *exec*) when the path search reaches the directory **/posix/bin**.

135397 If *PATH*=**/posix/bin**, various invocations yield different versions of *ls*:

135398	Invocation	Version of <i>ls</i>
135399	<i>ls</i> (from within application script)	(1) function
135400	<i>command ls</i> (from within application script)	(3) built-in
135401	<i>ls</i> (from within makefile called by application)	(3) built-in
135402	<i>system("ls")</i>	(3) built-in
135403	<i>PATH="/fred/bin:\$PATH" ls</i>	(2) user's version

### 135404 C.2.9.2 Pipelines

135405 Because pipeline assignment of standard input or standard output or both takes place before  
135406 redirection, it can be modified by redirection. For example:

135407 **\$** *command1* 2>&1 | *command2*

135408 sends both the standard output and standard error of *command1* to the standard input of  
135409 *command2*.

135410 The reserved word **!** allows more flexible testing using AND and OR lists. The behavior of **!(** is

135411 unspecified because in the Korn Shell this introduces a negated pathname expansion. Portable  
 135412 applications need to separate the ! and ( to ensure the command is treated as a negated subshell.

135413 It was suggested that it would be better to return a non-zero value if any command in the  
 135414 pipeline terminates with non-zero status (perhaps the bitwise-inclusive OR of all return values).  
 135415 However, the choice of the last-specified command semantics are historical practice and would  
 135416 cause applications to break if changed. An example of historical behavior:

```
135417 $ sleep 5 | (exit 4)
135418 $ echo $?
135419 4
135420 $ (exit 4) | sleep 5
135421 $ echo $?
135422 0
```

135423 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0029 [205] is applied.

135424 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0027 [521] is applied.

### 135425 **Exit Status**

135426 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0030 [52] is applied.

135427 Austin Group Defect 789 is applied, adding the *pipefail* option.

### 135428 C.2.9.3 *Lists*

135429 The equal precedence of "&&" and "||" is historical practice. The standard developers  
 135430 evaluated the model used more frequently in high-level programming languages, such as C, to  
 135431 allow the shell logical operators to be used for complex expressions in an unambiguous way, but  
 135432 they could not allow historical scripts to break in the subtle way unequal precedence might  
 135433 cause. Some arguments were posed concerning the "{}" or "()" groupings that are required  
 135434 historically. There are some disadvantages to these groupings:

- 135435 • The "()" can be expensive, as they spawn other processes on some implementations. This  
 135436 performance concern is primarily an implementation issue.
- 135437 • The "{}" braces are not operators (they are reserved words) and require a trailing  
 135438 <space> after each '{', and a <semicolon> before each '}'. Most programmers (and  
 135439 certainly interactive users) have avoided braces as grouping constructs because of the  
 135440 problematic syntax required. Braces were not changed to operators because that would  
 135441 generate compatibility issues even greater than the precedence question; braces appear  
 135442 outside the context of a keyword in many shell scripts.

135443 IEEE PASC Interpretation 1003.2 #204 is applied, clarifying that the operators "&&" and "||"  
 135444 are evaluated with left associativity.

135445 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0031 [45] and XCU/TC1-2008/0032  
 135446 [45] are applied.

135447 **Asynchronous AND-OR Lists**

135448 Unless the implementation has an internal limit, such as {CHILD\_MAX}, on the retained process  
 135449 IDs, it would require unbounded memory for the following example:

```
135450 while true
135451 do     foo & echo $!
135452 done
```

135453 The treatment of the signals SIGINT and SIGQUIT with asynchronous AND-OR lists is  
 135454 described in XCU [Section 2.12](#) (on page 2521).

135455 Since the connection of the input to the equivalent of /dev/null is considered to occur before  
 135456 redirections, the following script would produce no output:

```
135457 exec < /etc/passwd
135458 cat <&0 &
135459 wait
```

135460 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0028 [760] is applied.

135461 Austin Group Defect 1254 is applied, replacing the *Asynchronous Lists* section with an  
 135462 *Asynchronous AND-OR Lists* section.

135463 **Sequential AND-OR Lists**

135464 Austin Group Defect 1254 is applied, replacing the *Sequential Lists* section with a *Sequential AND-OR Lists* section.

135466 **AND Lists**

135467 There is no additional rationale provided for this section.

135468 **OR Lists**

135469 There is no additional rationale provided for this section.

135470 C.2.9.4 *Compound Commands*

135471 Austin Group Defect 1309 is applied, clarifying the exit status of the **for**, **case**, **if**, **while**, and  
 135472 **until** compound commands.

135473 **Grouping Commands**

135474 The semicolon shown in {*compound-list*; } is an example of a control operator delimiting the }  
 135475 reserved word. Other delimiters are possible, as shown in XCU [Section 2.10](#) (on page 2512);  
 135476 <newline> is frequently used.

135477 A proposal was made to use the <do-done> construct in all cases where command grouping in  
 135478 the current process environment is performed, identifying it as a construct for the grouping  
 135479 commands, as well as for shell functions. This was not included because the shell already has a  
 135480 grouping construct for this purpose ("{}"), and changing it would have been counter-  
 135481 productive.

135482 The requirement for conforming applications to separate two leading ' ( ' characters with white  
 135483 space if a grouping command would be parsed as an arithmetic expansion if preceded by a '\$'  
 135484 is to allow shells which implement the "( arithmetic expression )" extension to  
 135485 apply the same disambiguation rules consistently to \$(...) and (...). See [Section C.2.6.3](#)  
 135486 (on page 3884).

135487 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0033 [217] is applied.

135488 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] is applied.

### 135489 For Loop

135490 The format is shown with generous usage of <newline> characters. See the grammar in XCU  
135491 [Section 2.10](#) (on page 2512) for a precise description of where <newline> and <semicolon>  
135492 characters can be interchanged.

135493 Some historical implementations support '{ ' and '}' as substitutes for **do** and **done**. The  
135494 standard developers chose to omit them, even as an obsolescent feature. (Note that these  
135495 substitutes were only for the **for** command; the **while** and **until** commands could not use them  
135496 historically because they are followed by compound-lists that may contain "{ . . . }" grouping  
135497 commands themselves.)

135498 The reserved word pair **do** ... **done** was selected rather than **do** ... **od** (which would have  
135499 matched the spirit of **if** ... **fi** and **case** ... **esac**) because *od* is already the name of a standard  
135500 utility.

135501 PASC Interpretation 1003.2 #169 has been applied changing the grammar.

### 135502 Case Conditional Construct

135503 An optional <left-parenthesis> before *pattern* was added to allow numerous historical KornShell  
135504 scripts to conform. At one time, using the leading parenthesis was required if the **case** statement  
135505 was to be embedded within a "\$ ( )" command substitution; this is no longer the case with the  
135506 POSIX shell. Nevertheless, many historical scripts use the <left-parenthesis>, if only because it  
135507 makes matching-parenthesis searching easier in *vi* and other editors. This is a relatively simple  
135508 implementation change that is upwards-compatible for all scripts.

135509 Consideration was given to requiring *break* inside the *compound-list* to prevent falling through to  
135510 the next pattern action list. This was rejected as being nonexistent practice. Instead, the standard  
135511 now requires a feature first added in KornShell that using ";" &" instead of ";;" as a terminator  
135512 causes the exact opposite behavior—the flow of control continues with the next *compound-list*.

135513 Although the standard is explicit that the order of side-effects due to pattern expansion within a  
135514 single clause is unspecified, it is clear that patterns are expanded in clause order, and that no  
135515 further pattern expansions are attempted after the first match. That is, the following example is  
135516 required to output "1.0":

```
135517 x=0 y=1
135518 case 1 in
135519     $ ((y=0)) ) ;;
135520     $ ((x=1)) ) ;&
135521     $ ((x=2)) ) echo $x.$y ;;
135522 esac
```

135523 Some implementations of the shell also allow ";" &" as a terminator which falls through to the  
135524 next matching pattern (regardless of the choice of terminator in any intermediate non-matching  
135525 clauses), in contrast to ";" &" falling through to the next clause (regardless of the pattern  
135526 guarding that clause). This is an allowed extension, but is not required by the standard at this  
135527 time.

135528 The pattern '\*', given as the last pattern in a **case** construct, is equivalent to the default case in  
135529 a C-language **switch** statement.

135530 The grammar shows that reserved words can be used as patterns, even if one is the first word on  
135531 a line. Obviously, the reserved word **esac** cannot be used in this manner.

135532 Some historical shells would fall back to doing a byte to byte comparison with each pattern if the  
 135533 pattern matching rules did not produce a match. That behavior is not allowed by this standard  
 135534 because it allows user input to bypass input validations like:

```
135535 case $1 in
135536   [0123456789]) : OK;;
135537   *) echo >&2 not a decimal digit; exit 1;;
135538 esac
```

135539 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] is applied.

135540 Austin Group Defect 449 is applied, adding ; & as a **case** clause terminator.

135541 Austin Group Defect 1454 is applied, clarifying that a **case** statement with no patterns is valid  
 135542 syntax.

### 135543 **If Conditional Construct**

135544 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2512).

### 135545 **While Loop**

135546 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2512).

### 135547 **Until Loop**

135548 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2512).

### 135549 C.2.9.5 *Function Definition Command*

135550 The description of functions in an early proposal was based on the notion that functions should  
 135551 behave like miniature shell scripts; that is, except for sharing variables, most elements of an  
 135552 execution environment should behave as if they were a new execution environment, and  
 135553 changes to these should be local to the function. For example, traps and options should be reset  
 135554 on entry to the function, and any changes to them do not affect the traps or options of the caller.  
 135555 There were numerous objections to this basic idea, and the opponents asserted that functions  
 135556 were intended to be a convenient mechanism for grouping common commands that were to be  
 135557 executed in the current execution environment, similar to the execution of the *dot* special  
 135558 built-in.

135559 It was also pointed out that the functions described in that early proposal did not provide a local  
 135560 scope for everything a new shell script would, such as the current working directory, or *umask*,  
 135561 but instead provided a local scope for only a few select properties. The basic argument was that  
 135562 if a local scope is needed for the execution environment, the mechanism already existed: the  
 135563 application can put the commands in a new shell script and call that script. All historical shells  
 135564 that implemented functions, other than the KornShell, have implemented functions that operate  
 135565 in the current execution environment. Because of this, traps and options have a global scope  
 135566 within a shell script. Local variables within a function were considered and included in another  
 135567 early proposal (controlled by the special built-in *local*), but were removed because they do not fit  
 135568 the simple model developed for functions and because there was some opposition to adding yet  
 135569 another new special built-in that was not part of historical practice. Implementations should  
 135570 reserve the identifier *local* (as well as *typeset*, as used in the KornShell) in case this local variable  
 135571 mechanism is adopted in a future version of this standard.

135572 A separate issue from the execution environment of a function is the availability of that function  
 135573 to child shells. A few objectors maintained that just as a variable can be shared with child shells  
 135574 by exporting it, so should a function. In early proposals, the *export* command therefore had a *-f*

135575 flag for exporting functions. Functions that were exported were to be put into the environment  
 135576 as *name()*=*value* pairs, and upon invocation, the shell would scan the environment for these and  
 135577 automatically define these functions. This facility was strongly opposed and was omitted. Some  
 135578 of the arguments against exportable functions were as follows:

- 135579 • There was little historical practice. The Ninth Edition shell provided them, but there was  
 135580 controversy over how well it worked.
- 135581 • There are numerous security problems associated with functions appearing in the  
 135582 environment of a user and overriding standard utilities or the utilities owned by the  
 135583 application.
- 135584 • There was controversy over requiring *make* to import functions, where it has historically  
 135585 used an *exec* function for many of its command line executions.
- 135586 • Functions can be big and the environment is of a limited size. (The counter-argument was  
 135587 that functions are no different from variables in terms of size: there can be big ones, and  
 135588 there can be small ones—and just as one does not export huge variables, one does not  
 135589 export huge functions. However, this might not apply to the average shell-function writer,  
 135590 who typically writes much larger functions than variables.)

135591 As far as can be determined, the functions in the Shell and Utilities volume of POSIX.1-2024  
 135592 match those in System V. Earlier versions of the KornShell had two methods of defining  
 135593 functions:

```
135594 function fname { compound-list }
```

135595 and:

```
135596 fname() { compound-list }
```

135597 The latter used the same definition as the Shell and Utilities volume of POSIX.1-2024, but  
 135598 differed in semantics, as described previously. The current edition of the KornShell aligns the  
 135599 latter syntax with the Shell and Utilities volume of POSIX.1-2024 and keeps the former as is.

135600 Some shells accept simple commands (see XCU Section 2.9.1, on page 2500) after *fname()* in  
 135601 addition to compound commands (see XCU Section 2.9.4, on page 2508); however this standard  
 135602 only requires support for compound commands.

135603 The name space for functions is limited to that of a *name* because of historical practice.  
 135604 Complications in defining the syntactic rules for the function definition command and in  
 135605 dealing with known extensions such as the "*@()*" usage in the KornShell prevented the name  
 135606 space from being widened to a *word*. Using functions to support synonyms such as the "*!!*"  
 135607 and "*%*" usage in the C shell is thus disallowed to conforming applications, but acceptable as an  
 135608 extension. For interactive users, the aliasing facilities in the Shell and Utilities volume of  
 135609 POSIX.1-2024 should be adequate for this purpose. It is recognized that the name space for  
 135610 utilities in the file system is wider than that currently supported for functions, if the portable  
 135611 filename character set guidelines are ignored, but it did not seem useful to mandate extensions  
 135612 in systems for so little benefit to conforming applications.

135613 The "*()*" in the function definition command consists of two operators. Therefore, intermixing  
 135614 <blank> characters with the *fname*, '*(*', and '*)*' is allowed, but unnecessary.

135615 An example of how a function definition can be used wherever a simple command is allowed:

```
135616 # If variable i is equal to "yes",
135617 # define function foo to be ls -l
135618 #
135619 [ "$i" = yes ] && foo() {
135620     ls -l
```



135621 }  
 135622 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0034 [383] and XCU/TC1-2008/0035  
 135623 [214] are applied.  
 135624 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] and XCU/TC2-2008/0030  
 135625 [654] are applied.

## 135626 C.2.10 Shell Grammar

135627 There are several subtle aspects of this grammar where conventional usage implies rules about  
 135628 the grammar that in fact are not true.

135629 For *compound\_list*, only the forms that end in a *separator* allow a reserved word to be recognized,  
 135630 so usually only a *separator* can be used where a compound list precedes a reserved word (such as  
 135631 **Then, Else, Do, and Rbrace**). Explicitly requiring a separator would disallow such valid (if rare)  
 135632 statements as:

```
135633 if (false) then (echo x) else (echo y) fi
```

135634 See the Note under special grammar rule (1).

135635 Concerning the third sentence of rule (1) (“Also, if the parser ...”):

- 135636 • This sentence applies rather narrowly: when a compound list is terminated by some clear  
 135637 delimiter (such as the closing **fi** of an inner **if\_clause**) then it would apply; where the  
 135638 compound list might continue (as in after a **;**), rule (7a) (and consequently the first  
 135639 sentence of rule (1)) would apply. In many instances the two conditions are identical, but  
 135640 this part of rule (1) does not give license to treating a **WORD** as a reserved word unless it  
 135641 is in a place where a reserved word has to appear.
- 135642 • The statement is equivalent to requiring that when the LR(1) lookahead set contains  
 135643 exactly one reserved word, it must be recognized if it is present. (Here “LR(1)” refers to the  
 135644 theoretical concepts, not to any real parser generator.)

135645 For example, in the construct below, and when the parser is at the point marked with **^**,  
 135646 the only next legal token is **then** (this follows directly from the grammar rules):

```
135647 if if...fi then ... fi
135648         ^
```

135649 At that point, the **then** must be recognized as a reserved word.

135650 (Depending on the parser generator actually used, “extra” reserved words may be in some  
 135651 lookahead sets. It does not really matter if they are recognized, or even if any possible  
 135652 reserved word is recognized in that state, because if it is recognized and is not in the  
 135653 (theoretical) LR(1) lookahead set, an error is ultimately detected. In the example above, if  
 135654 some other reserved word (for example, **while**) is also recognized, an error occurs later.

135655 This is approximately equivalent to saying that reserved words are recognized after other  
 135656 reserved words (because it is after a reserved word that this condition occurs), but avoids  
 135657 the “except for ...” list that would be required for **case**, **for**, and so on. (Reserved words  
 135658 are of course recognized anywhere a *simple\_command* can appear, as well. Other rules take  
 135659 care of the special cases of non-recognition, such as rule (4) for **case** statements.)

135660 Note that the body of here-documents are handled by token recognition (see XCU [Section 2.3](#), on  
 135661 page 2475) and do not appear in the grammar directly. (However, the here-document I/O  
 135662 redirection operator is handled as part of the grammar.)

135663 The optional redirection syntax:

135664 `{location}redir-op word`  
 135665 (see XCU [Section 2.7](#), on page 2493) is accommodated in the grammar rules by the optional  
 135666 **IO\_LOCATION** token identifier and two correspondingly optional elements in `io_redirect`.  
 135667 Without these, the grammar would not permit this form of redirection because it would require  
 135668 that, for example, `echo {var}> foo` is parsed such that `{var}` is a **WORD** to be expanded  
 135669 and passed to `echo`. The grammar does not restrict the location given between the `'{'` and `'}'`  
 135670 in these forms (other than requiring it to be non-empty) since shells may parse an invalid  
 135671 location as part of an `io_redirect` and later treat the invalid location as an error.

#### 135672 C.2.10.1 Shell Grammar Lexical Conventions

135673 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0031 [648] and XCU/TC2-2008/0032  
 135674 [574,646] are applied.

135675 Austin Group Defect 1193 is applied, adding the optional **IO\_LOCATION** token identifier.

135676 Austin Group Defect 1454 is applied, clarifying how to convert the token identifier type of the  
 135677 **TOKEN** when rule 1 applies.

#### 135678 C.2.10.2 Shell Grammar Rules

135679 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0036 [44] is applied.

135680 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0033 [643,839], XCU/TC2-2008/0034  
 135681 [643], XCU/TC2-2008/0035 [648], XCU/TC2-2008/0036 [736], XCU/TC2-2008/0037 [737],  
 135682 XCU/TC2-2008/0038 [581], and XCU/TC2-2008/0039 [735] are applied.

135683 Austin Group Defect 249 is applied, adding the dollar-single-quotes quoting mechanism.

135684 Austin Group Defect 449 is applied, adding `;&` as a **case** clause terminator.

135685 Austin Group Defect 1193 is applied, adding the optional **IO\_LOCATION** token identifier.

135686 Austin Group Defects 1276 and 1279 are applied, clarifying rule 7.

135687 Austin Group Defect 1454 is applied, clarifying how rule 4 applies.

### 135688 C.2.11 Job Control

135689 See also [Job Control](#) (on page 3656).

135690 Shell implementations differ regarding how much of a foreground job is retained when it is  
 135691 converted to a suspended job. For example, given this foreground job:

135692 `sleep 10; echo foo; echo bar &`

135693 if this is suspended during execution of the `sleep`, *ksh93* retains all of the commands in the  
 135694 suspended job and executes them when `fg` is used:

135695 `^Z[1] + Stopped` `sleep 10; echo foo; echo bar &`

135696 `$ jobs`

135697 `[1] + Stopped` `sleep 10; echo foo; echo bar &`

135698 `$ fg`

135699 `sleep 10; echo foo; echo bar`

135700 `foo`

135701 `[1] 30686`

135702 `bar`

135703 §

135704 However, some other shells create a suspended job containing only the `sleep 10` command.

135705 Some historical shells did not handle suspending a foreground AND-OR list well. They would  
 135706 treat the wait status of a process that indicated it had stopped as if it was a non-zero exit status  
 135707 and (if the next operator in the AND-OR list was `|`) would execute the remainder of the AND-  
 135708 OR list at that point. This behavior is not allowed by the standard for two reasons:

135709 1. It does not meet the fundamental requirement of an AND-OR list that the decision on  
 135710 whether to execute each part (except the first) is made based on the exit status of the  
 135711 previous part when it completes.

135712 2. It can lead to data loss. For example, consider a user who often runs this command:

135713 `generate_report > report.out || rm report.out`

135714 with the intention that the incomplete results from a failed `generate_report` run are never  
 135715 retained in order that they cannot be mistaken for a complete set of results. If one day the  
 135716 user decides to check on the progress of the command by stopping it and examining what  
 135717 has been written so far, they will find that the **report.out** file has already been removed.

135718 Austin Group Defects 1254 and 1675 are applied, adding this section.

### 135719 C.2.12 Signals and Error Handling

135720 Historically, some shell implementations silently ignored attempts to use `trap` to set SIGINT or  
 135721 SIGQUIT to the default action or to set a trap for them after they have been set to be ignored by  
 135722 the shell when it executes an asynchronous subshell (and job control is disabled). This behavior  
 135723 is not conforming. For example, if a shell script containing the following line is run in the  
 135724 foreground at a terminal:

135725 `(trap - INT; exec sleep 10) & wait`

135726 and is then terminated by typing the interrupt character, this standard requires that the `sleep`  
 135727 command is terminated by the SIGINT signal.

135728 SD5-XCU-ERN-93 is applied, updating the first paragraph of XCU [Section 2.12](#) (on page 2521).

135729 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0040 [750] is applied.

### 135730 C.2.13 Shell Execution Environment

135731 Some implementations have implemented the last stage of a pipeline in the current environment  
 135732 so that commands such as:

135733 `command | read foo`

135734 set variable **foo** in the current environment. This extension is allowed, but not required;  
 135735 therefore, a shell programmer should consider a pipeline to be in a subshell environment, but  
 135736 not depend on it.

135737 In early proposals, the description of execution environment failed to mention that each  
 135738 command in a multiple command pipeline could be in a subshell execution environment. For  
 135739 compatibility with some historical shells, the wording was phrased to allow an implementation  
 135740 to place any or all commands of a pipeline in the current environment. However, this means that  
 135741 a POSIX application must assume each command is in a subshell environment, but not depend  
 135742 on it.

- 135743 The wording about shell scripts is meant to convey the fact that describing “trap actions” can  
 135744 only be understood in the context of the shell command language. Outside of this context, such  
 135745 as in a C-language program, signals are the operative condition, not traps.
- 135746 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0037 [238] is applied.
- 135747 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0041 [706] is applied.
- 135748 Austin Group Defect 1247 is applied, changing “signal traps” to “traps” and changing “All other  
 135749 commands” to “Except where otherwise stated, all other commands”.
- 135750 Austin Group Defect 1254 is applied, changing the list item relating to process IDs “known to  
 135751 this shell environment”.
- 135752 Austin Group Defect 1384 is applied, changing the requirements for subshells of interactive  
 135753 shells.
- 135754 Austin Group Defect 1580 is applied, adding a list item about environment variables with  
 135755 invalid names.

## 135756 C.2.14 Pattern Matching Notation

- 135757 Pattern matching is a simpler concept and has a simpler syntax than REs, as the former is  
 135758 generally used for the manipulation of filenames, which are relatively simple collections of  
 135759 characters, while the latter is generally used to manipulate arbitrary text strings of potentially  
 135760 greater complexity. However, some of the basic concepts are the same, so this section points  
 135761 liberally to the detailed descriptions in XBD [Chapter 9](#) (on page 179).
- 135762 Austin Group Defect 1443 is applied, adding non-shell uses to the description of what shell  
 135763 pattern matching notation is used for.
- 135764 Austin Group Defect 1564 is applied, clarifying that pattern matching notation is used for  
 135765 matching character strings (not arbitrary byte strings), and that if an attempt is made to use  
 135766 pattern matching notation to match a string that contains one or more bytes that do not form  
 135767 part of a valid character, the behavior is unspecified.

### 135768 C.2.14.1 Patterns Matching a Single Character

- 135769 Both quoting and escaping are described here because pattern matching must work in three  
 135770 separate circumstances:
- 135771 1. Calling directly upon the shell, such as in pathname expansion or in a **case** statement. All  
 135772 of the following match the string or file **abc**:  
 135773 `abc "abc" a"b" c a\bc a[b]c a["b"]c a[\b]c a["\b"]c a?c a*c`  
 135774 The following do not:  
 135775 `"a?c" a*c a\b]c`
  - 135776 2. Calling a utility or function without going through a shell, as described for *find* and the  
 135777 *fnmatch()* and *glob()* functions defined in the System Interfaces volume of POSIX.1-2024,  
 135778 or pattern matching in the shell in situations where the pattern is specified indirectly  
 135779 instead of directly to the shell, such as:  
 135780 `ls -ld -- $pattern`  
 135781 `or`  
 135782 `case $var in ($pattern) ...`

135783 3. Calling utilities such as *find*, *cpio*, *tar*, or *pax* through the shell command line. In this case,  
135784 shell quote removal is performed before the utility sees the argument. For example, in:

```
135785 find /bin -name "e\c[\h]o" -print
```

135786 after quote removal, the `<backslash>` characters are presented to *find* and it treats them as  
135787 escape characters. Both precede ordinary characters, so the *c* and *h* represent themselves  
135788 and *echo* would be found on many historical systems (that have it in */bin*). To find a  
135789 filename that contained shell special characters or pattern characters, both quoting and  
135790 escaping are required, such as:

```
135791 pax -r ... "*a(\?"
```

135792 to extract a filename ending with "a(?".

135793 The wording “In a pattern, or part of one, where a shell-quoting `<backslash>` cannot be used to  
135794 preserve the literal value of a character that would otherwise be treated as special” has been  
135795 carefully crafted so that for the shell it only applies to certain contexts. In particular:

- 135796 • The use of “or part of one” is needed because a single pattern can be produced partly from  
135797 characters directly included in a word and partly from characters that result from one or  
135798 more of the word expansions. For example, in the following command the `<backslash>`  
135799 escapes the `'?'` character:

```
135800 dir='abc\?'  
135801 ls -l -- $dir/*.c
```

- 135802 • The reference to “a shell-quoting `<backslash>`” rather than just using “where shell quoting  
135803 cannot be used” is because there are ways that other types of shell quoting can be used  
135804 where a shell-quoting `<backslash>` cannot, such as placing an expansion within double-  
135805 quotes as in this example:

```
135806 dir='abc?'  
135807 ls -l -- "$dir"/*.c
```

- 135808 • The use of “that would otherwise be treated as special” is needed because otherwise the  
135809 condition would apply to `<backslash>` in single-quotes. For example, in the following  
135810 command the `<backslash>` is not treated as escaping the `'?'` because the `'?'` would not  
135811 be treated as special anyway:

```
135812 ls -l 'abc\?'/*.c
```

135813 In patterns specified indirectly to the shell, it is unspecified whether or not `<backslash>` is  
135814 special inside bracket expressions. This is because there are two mutually exclusive consistency  
135815 aims and neither is considered more important than the other. One is consistency with direct  
135816 patterns, where `<backslash>` is special inside bracket expressions (which is, in turn, for  
135817 consistency with the way single-quotes and double-quotes preserve the literal value of  
135818 characters inside bracket expressions); the other is consistency with regular expressions, *find*,  
135819 *pax*, *fnmatch()*, and *glob()*, where `<backslash>` is not special inside bracket expressions (not  
135820 counting the extra C-string escaping in EREs in *awk*).

135821 Earlier versions of this standard allowed two behaviors when a pattern ends with an unescaped  
135822 `<backslash>`: it could match nothing or be treated as an invalid pattern. However, a third  
135823 behavior has since been observed, where the ending `<backslash>` is treated as a literal  
135824 `<backslash>`, and therefore this standard now simply states that the behavior is unspecified.

135825 Earlier versions of this standard included the statement “The shell special characters always  
135826 require quoting” in XCU [Section 2.14.1](#) (on page 2523). It is unclear what was intended by this,  
135827 since there are pattern matching contexts in which it is not possible to quote those characters,  
135828 such as:

135829 `execlp("find", "find", ".", "-name", "*[()]*", (char *)0);`

135830 where the parentheses cannot be escaped with a <backslash> because <backslash> is not special  
 135831 in bracket expressions in that context. The statement is thought to have been a warning to  
 135832 application writers and interactive shell users that shell special characters (sometimes called  
 135833 metacharacters) always need quoting in patterns that appear directly in shell code; for example,  
 135834 this code:

```
135835 case $char in
135836 [()]) ... ;;
135837 esac
```

135838 is incorrect because the parentheses are parsed as operators—they need to be quoted in order to  
 135839 be treated as part of the pattern. This standard now simply requires instead that applications  
 135840 quote or escape any character that would otherwise be treated as special, in order for it to be  
 135841 matched as an ordinary character. If shell special characters are used without this protection in  
 135842 contexts where they are treated as special, syntax errors can result or implementation extensions  
 135843 can be triggered. Some shells support a series of extensions based on parentheses in patterns  
 135844 that are valid extensions in these contexts because they would otherwise cause syntax errors.  
 135845 However, this means that they are not allowed by this standard to be recognized in contexts  
 135846 where those syntax errors would not occur anyway, such as in:

```
135847 pattern='a*(b)'; ls -- $pattern
```

135848 which this standard requires to list files with names beginning 'a' and ending "(b)". It is  
 135849 recommended that implementations do not extend pattern matching in the shell in ways that are  
 135850 only valid extensions because they would otherwise be syntax errors, in order to avoid  
 135851 inconsistency between different pattern matching contexts. One way to provide an extension  
 135852 that is consistent between different pattern matching contexts in the shell (although still not  
 135853 consistent with *find -name*, *fnmatch()*, etc.) is to enable the extension only when a non-standard  
 135854 shell option is set, or when the shell is executed using a command name other than *sh*.  
 135855 Consistency with non-shell contexts can then be achieved by enabling equivalent extensions in  
 135856 those other contexts by use of non-standard utility options or non-standard FNM\_\* and GLOB\_\*  
 135857 flags.

135858 The restriction on a <circumflex> in a bracket expression is to allow implementations that  
 135859 support pattern matching using the <circumflex> as the negation character in addition to the  
 135860 <exclamation-mark>. A conforming application must use something like "[\^!]" to match  
 135861 either character.

135862 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0042 [806] is applied.

135863 Austin Group Defect 985 is applied, changing the description of the '[' special character.

135864 Austin Group Defect 1234 is applied, clarifying how <backslash> is handled in patterns.

#### 135865 C.2.14.2 *Patterns Matching Multiple Characters*

135866 Since each <asterisk> matches zero or more occurrences, the patterns "a\*b" and "a\*\*b" have  
 135867 identical functionality.

135868 **Examples**

- 135869 `a[bc]` Matches the strings "ab" and "ac".
- 135870 `a*d` Matches the strings "ad", "abd", and "abcd", but not the string "abc".
- 135871 `a*d*` Matches the strings "ad", "abcd", "abcdef", "aaaad", and "adddd".
- 135872 `*a*d` Matches the strings "ad", "abcd", "efabcd", "aaaad", and "adddd".

135873 C.2.14.3 *Patterns Used for Filename Expansion*

135874 The caveat about a `<slash>` within a bracket expression is derived from historical practice. The  
 135875 pattern `"a[b/c]d"` does not match such pathnames as **abd** or **a/d**. On some implementations  
 135876 (including those conforming to the Single UNIX Specification), it matched a pathname of  
 135877 literally `"a[b/c]d"`. On other systems, it produced an undefined condition (an unescaped `'[`  
 135878 used outside a bracket expression). In this version, the XSI behavior is now required.

135879 Filenames beginning with a `<period>` historically have been specially protected from view on  
 135880 UNIX systems. A proposal to allow an explicit `<period>` in a bracket expression to match a  
 135881 leading `<period>` was considered; it is allowed as an implementation extension, but a  
 135882 conforming application cannot make use of it. If this extension becomes popular in the future, it  
 135883 will be considered for a future version of the Shell and Utilities volume of POSIX.1-2024.

135884 Patterns are matched against existing filenames and pathnames only when the pattern contains  
 135885 a `'*'`, `'?'` or `'['` character that will be treated as special. This prevents accidental removal of  
 135886 `<backslash>` characters in variable expansions where generating a list of matching files is not  
 135887 intended and a (usually oddly named) file with a matching name happens to exist. For example,  
 135888 a shell script that tries to be portable to systems that predate the introduction of functions and  
 135889 `printf` might use this on POSIX systems:

135890 `myecho='printf %s\n'`

135891 to be used as:

135892 `$myecho args...`

135893 If `%s\n` were to be matched against existing files, this would not work if a file called `%sn`  
 135894 happened to exist.

135895 Historical systems have varied in their permissions requirements. To match `f*/bar` has required  
 135896 read permissions on the `f*` directories in the System V shell, but the Shell and Utilities volume of  
 135897 POSIX.1-2024, the C shell, and KornShell require only search permissions. If read or search  
 135898 permission is denied, shells do not report an error but treat this as a successful "no match"  
 135899 condition. Error conditions that are related to file system contents and occur when attempting to  
 135900 read or search a directory are also required to be treated the same way because they imply that  
 135901 there are no matches (that are accessible to the process). For example, if the pattern is `foo/*bar`  
 135902 and attempting to open the directory `foo` fails because it does not exist or is not a directory, then  
 135903 there can be no matching pathnames. The error conditions listed in XSH Section 2.3 (on page  
 135904 507) that are related to file system contents and could occur when attempting to open or search a  
 135905 directory are [EACCES], [ELOOP], [ENAMETOOLONG], [ENOENT], and [ENOTDIR]. Error  
 135906 conditions that are not related to file system contents or which occur when reading a directory,  
 135907 notably [EMFILE] and [ENFILE] but also things like [EIO], [ENOMEM], and [EOVERFLOW],  
 135908 can either be treated as errors or be treated the same way as when permission is denied. Treating  
 135909 them as errors is seen as desirable, because to do otherwise would mean the shell could execute  
 135910 a command with an unchanged pattern when pathnames matching the pattern exist, but it is not  
 135911 historical practice. Implementations that handle the two categories of error differently should  
 135912 also handle non-standard error conditions appropriately, if encountered, depending on which

- 135913 category they fit into.
- 135914 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0043 [963] is applied.
- 135915 Austin Group Defect 1070 is applied, requiring that when the matching filenames or pathnames  
135916 are sorted, any that collate equally are further compared byte-by-byte using the collating  
135917 sequence for the POSIX locale.
- 135918 Austin Group Defect 1228 is applied, allowing directory entries for dot and dot-dot to be  
135919 ignored when matching patterns against existing filenames.
- 135920 Austin Group Defect 1234 is applied, changing the behavior of patterns used for filename  
135921 expansion such that a pattern is matched against existing filenames and pathnames only when it  
135922 contains a '\*', '?', or '[' character that will be treated as special.
- 135923 Austin Group Defects 1273 and 1275 are applied, clarifying how errors are treated when  
135924 attempting to open or search a pathname as a directory or attempting to read an opened  
135925 directory.

### 135926 C.2.15 Special Built-In Utilities

- 135927 See the RATIONALE sections on the individual reference pages.
- 135928 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0044 [882] and XCU/TC2-2008/0045  
135929 [654] are applied.
- 135930 Austin Group Defect 1009 is applied, clarifying the behavior when a special built-in utility is  
135931 executed with a variable assignment.
- 135932 Austin Group Defect 1445 is applied, changing text relating to the term "built-in".

## 135933 C.3 Utilities

- 135934 For the utilities included in POSIX.1-2024, see the RATIONALE sections on the individual  
135935 reference pages.

### 135936 C.3.1 Utilities Removed in this Version

- 135937 The following utilities were removed in this version of this standard:

135938	<i>qalter</i>	<i>qmove</i>	<i>qrls</i>	<i>qstat</i>
135939	<i>qdel</i>	<i>qmsg</i>	<i>qselect</i>	<i>qsub</i>
135940	<i>qhold</i>	<i>qrerun</i>	<i>qsig</i>	

### 135941 C.3.2 Utilities Removed in the Previous Version

- 135942 None.



135943 **C.3.3 Exclusion of Utilities**

135944 The set of utilities contained in POSIX.1-2024 is drawn from the base documents for IEEE Std  
135945 1003.2-1992, with one addition: the *c17* utility. This section contains rationale for some of the  
135946 deliberations that led to this set of utilities, and why certain utilities were excluded.

135947 Many utilities were evaluated by the standard developers; more historical utilities were  
135948 excluded from the base documents for IEEE Std 1003.2-1992 than included. The following list  
135949 contains many common UNIX system utilities that were not included as mandatory utilities, in  
135950 the User Portability Utilities option, in the XSI option, or in one of the software development  
135951 groups. It is logistically difficult for this rationale to distribute correctly the reasons for not  
135952 including a utility among the various utility options. Therefore, this section covers the reasons  
135953 for all utilities not included in POSIX.1-2024.

135954 This rationale is limited to a discussion of only those utilities actively or indirectly evaluated by  
135955 the IEEE Std 1003.2-1992 standard developers, rather than the list of all known UNIX utilities  
135956 from all its variants.

135957 *adb* The intent of the various software development utilities was to assist in the  
135958 installation (rather than the actual development and debugging) of applications.  
135959 This utility is primarily a debugging tool. Furthermore, many useful aspects of *adb*  
135960 are very hardware-specific.

135961 *as* Assemblers are hardware-specific and are included implicitly as part of the  
135962 compilers in POSIX.1-2024.

135963 *banner* The only known use of this command is as part of the *lp* printer header pages. It  
135964 was decided that the format of the header is implementation-defined, so this utility  
135965 is superfluous to application portability.

135966 *calendar* This reminder service program is not useful to conforming applications.

135967 *cancel* The *lp* (line printer spooling) system specified is the most basic possible and did  
135968 not need this level of application control.

135969 *chroot* This is primarily of administrative use, requiring superuser privileges.

135970 *col* No utilities defined in POSIX.1-2024 produce output requiring such a filter. The  
135971 *nroff* text formatter is present on many historical systems and will continue to  
135972 remain as an extension; *col* is expected to be shipped by all the systems that ship  
135973 *nroff*.

135974 *cpio* This has been replaced by *pax*, for reasons explained in the rationale for that utility.

135975 *cpp* This is subsumed by *c17*.

135976 *cu* This utility is terminal-oriented and is not useful from shell scripts or typical  
135977 application programs.

135978 *dc* The functionality of this utility can be provided by the *bc* utility; *bc* was selected  
135979 because it was easier to use and had superior functionality. Although the historical  
135980 versions of *bc* are implemented using *dc* as a base, POSIX.1-2024 prescribes the  
135981 interface and not the underlying mechanism used to implement it.

135982 *dircmp* Although a useful concept, the historical output of this directory comparison  
135983 program is not suitable for processing in application programs. Also, the *diff -r*  
135984 command gives equivalent functionality.


135985 *dis* Disassemblers are hardware-specific.

135986	<i>emacs</i>	The community of <i>emacs</i> editing enthusiasts was adamant that the full <i>emacs</i> editor not be included in IEEE Std 1003.2-1992 because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original <i>emacs</i> program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include <i>emacs</i> , or included it without supporting it, but there were very few that did not include and support <i>vi</i> .	
135987			
135988			
135989			
135990			
135991			
135992			
135993			
135994	<i>ld</i>	This is subsumed by <i>c17</i> .	
135995	<i>line</i>	The functionality of <i>line</i> can be provided with <i>read</i> .	
135996	<i>lint</i>	This technology is partially subsumed by <i>c17</i> . It is also hard to specify the degree of checking for possible error conditions in programs in any compiler, and specifying what <i>lint</i> would do in these cases is equally difficult.	
135997			
135998			
135999		It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, any description of <i>lint</i> is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, one cannot require <i>lint</i> to detect all errors of any given class.	
136000			
136001			
136002			
136003			
136004			
136005			
136006			
136007			
136008			
136009		Additionally, some vendors complained that since many compilers are distributed in a binary form without a <i>lint</i> facility (because the ISO C standard does not require one), implementing the standard as a stand-alone product will be much harder. Rather than being able to build upon a standard compiler component (simply by providing <i>c17</i> as an interface), source to that compiler would most likely need to be modified to provide the <i>lint</i> functionality. This was considered a major burden on system providers for a very small gain to developers (users).	
136010			
136011			
136012	<i>login</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.	
136013			
136014	<i>lorder</i>	This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.	
136015			
136016	<i>lpstat</i>	The <i>lp</i> system specified is the most basic possible and did not need this level of application control.	
136017			
136018	<i>mail</i>	This utility was omitted in favor of <i>mailx</i> because there was a considerable functionality overlap between the two.	
136019			
136020	<i>mknod</i>	This was omitted in favor of <i>mkfifo</i> , as <i>mknod</i> has too many implementation-defined functions.	
136021			
136022	<i>news</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.	
136023			
136024	<i>pack</i>	This compression program was considered inferior to <i>compress</i> .	
136025	<i>passwd</i>	This utility was proposed in an early draft of the IEEE Std 1003.2-1992 UPE but met with too many objections to be included. There were various reasons:	
136026			
136027			• Changing a password should not be viewed as a command, but as part of the login sequence. Changing a password should only be done while a trusted path is in effect.
136028			
136029			

136030		<ul style="list-style-type: none"> <li>• Even though the text in early drafts was intended to allow a variety of implementations to conform, the security policy for one site may differ from another site running with identical hardware and software. One site might use password authentication while the other did not. Vendors could not supply a <i>passwd</i> utility that would conform to POSIX.1-2024 for all sites using their system.</li> <li>• This is really a subject for a system administration working group or a security working group.</li> </ul>
136031		
136032		
136033		
136034		
136035		
136036		
136037		
136038	<i>pcat</i>	This compression program was considered inferior to <i>zcat</i> .
136039	<i>pg</i>	This duplicated many of the features of the <i>more</i> pager, which was preferred by the standard developers.
136040		
136041	<i>prof</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
136042		
136043		
136044	RCS	RCS was originally considered as part of a version control utilities portion of the scope. However, this aspect was abandoned by the standard developers. SCCS is now included as an optional part of the XSI option.
136045		
136046		
136047	<i>red</i>	Restricted editor. This was not considered by the standard developers because it never provided the level of security restriction required.
136048		
136049	<i>rsh</i>	Restricted shell. This was not considered by the standard developers because it does not provide the level of security restriction that is implied by historical documentation.
136050		
136051		
136052	<i>sdb</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, some useful aspects of <i>sdb</i> are very hardware-specific.
136053		
136054		
136055		
136056	<i>sdiff</i>	The “side-by-side <i>diff</i> ” utility from System V was omitted because it is used infrequently, and even less so by conforming applications. Despite being in System V, it is not in the SVID or XPG.
136057		
136058		
136059	<i>shar</i>	Any of the numerous “shell archivers” were excluded because they did not meet the requirement of existing practice.
136060		
136061	<i>shl</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs. The job control aspects of the shell command language are generally more useful.
136062		
136063		
136064	<i>size</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
136065		
136066		
136067	<i>spell</i>	This utility is not useful from shell scripts or typical application programs. The <i>spell</i> utility was considered, but was omitted because there is no known technology that can be used to make it recognize general language for user-specified input without providing a complete dictionary along with the input file.
136068		
136069		
136070		
136071	<i>su</i>	This utility is not useful from shell scripts or typical application programs. (There was also sentiment to avoid security-related utilities.)
136072		

136073	<i>sum</i>	This utility was renamed <i>cksum</i> .
136074	<i>tar</i>	This has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.
136075	<i>unpack</i>	This compression program was considered inferior to <i>uncompress</i> .
136076	<i>wall</i>	This utility is terminal-oriented and is not useful in shell scripts or typical applications. It is generally used only by system administrators.
136077		

136078

 *Rationale (Informative)*

136079

**Part D:**

136080

**Portability Considerations**

136081

*The Open Group*

136082

*The Institute of Electrical and Electronics Engineers, Inc.*



136083

136084

## Portability Considerations (Informative)

136085

This section contains information to satisfy various international requirements:

136086

- [Section D.1](#) describes perceived user requirements.

136087

- [Section D.2](#) (on page 3923) indicates how the facilities of POSIX.1-2024 satisfy those requirements.

136088

136089

- [Section D.3](#) (on page 3931) offers guidance to writers of profiles on how the configurable options, limits, and optional behavior of POSIX.1-2024 should be cited in profiles.

136090

### D.1 User Requirements

136092

This section describes the user requirements that were perceived by the standard developers. The primary source for these requirements was an analysis of historical practice in widespread use, as typified by the base documents for the ISO POSIX-1: 1996 standard.

136093

136094

136095

POSIX.1-2024 addresses the needs of users requiring open systems solutions for source code portability of applications. It currently addresses users requiring open systems solutions for source-code portability of applications involving multi-programming and process management (creating processes, signaling, and so on); access to files and directories in a hierarchy of file systems (opening, reading, writing, deleting files, and so on); access to asynchronous communications ports and other special devices; access to information about other users of the system; facilities supporting applications requiring bounded (realtime) response.

136096

136097

136098

136099

136100

136101

136102

The following users are identified for POSIX.1-2024:

136103

- Those employing applications written in high-level languages, such as C, Ada, or FORTRAN.

136104

136105

- Users who desire conforming applications that do not necessarily require the characteristics of high-level languages (for example, the speed of execution of compiled languages or the relative security of source code intellectual property inherent in the compilation process).

136106

136107

136108

136109

- Users who desire conforming applications that can be developed quickly and can be modified readily without the use of compilers and other system components that may be unavailable on small systems or those without special application development capabilities.

136110

136111

136112

136113

- Users who interact with a system to achieve general-purpose time-sharing capabilities common to most business or government offices or academic environments: editing, filing, inter-user communications, printing, and so on.

136114

136115

136116

- Users who develop applications for POSIX-conformant systems.

136117

- Users who develop applications for UNIX systems.

136118

An acknowledged restriction on applicable users is that they are limited to the group of individuals who are familiar with the style of interaction characteristic of historically-derived systems based on one of the UNIX operating systems (as opposed to other historical systems with different models, such as MS/DOS, Macintosh, VMS, MVS, and so on). Typical users

136119

136120

136121

- 136122 would include program developers, engineers, or general-purpose time-sharing users.
- 136123 The requirements of users of POSIX.1-2024 can be summarized as a single goal: *application source portability*. The requirements of the user are stated in terms of the requirements of portability of applications. This in turn becomes a requirement for a standardized set of syntax and semantics for operations commonly found on many operating systems.
- 136124
- 136125
- 136126
- 136127 The following sections list the perceived requirements for application portability.

#### 136128 **D.1.1 Configuration Interrogation**

- 136129 An application must be able to determine whether and how certain optional features are provided and to identify the system upon which it is running, so that it may appropriately adapt to its environment.
- 136130
- 136131
- 136132 Applications must have sufficient information to adapt to varying behaviors of the system.

#### 136133 **D.1.2 Process Management**

- 136134 An application must be able to manage itself, either as a single process or as multiple processes. Applications must be able to manage other processes when appropriate.
- 136135
- 136136 Applications must be able to identify, control, create, and delete processes, and there must be communication of information between processes and to and from the system.
- 136137
- 136138 Applications must be able to use multiple flows of control with a process (threads) and synchronize operations between these flows of control.
- 136139

#### 136140 **D.1.3 Access to Data**

- 136141 Applications must be able to operate on the data stored on the system, access it, and transmit it to other applications. Information must have protection from unauthorized or accidental access or modification.
- 136142
- 136143

#### 136144 **D.1.4 Access to the Environment**

- 136145 Applications must be able to access the external environment to communicate their input and results.
- 136146

#### 136147 **D.1.5 Access to Determinism and Performance Enhancements**

- 136148 Applications must have sufficient control of resource allocation to ensure the timeliness of interactions with external objects.
- 136149



**136150 D.1.6 Operating System-Dependent Profile**

136151 The capabilities of the operating system may make certain optional characteristics of the base  
136152 language in effect no longer optional, and this should be specified.

**136153 D.1.7 I/O Interaction**

136154 The interaction between the C language I/O subsystem (*stdio*) and the I/O subsystem of  
136155 POSIX.1-2024 must be specified.

**136156 D.1.8 Internationalization Interaction**

136157 The effects of the environment of POSIX.1-2024 on the internationalization facilities of the C  
136158 language must be specified.

**136159 D.1.9 C-Language Extensions**

136160 Certain functions in the C language must be extended to support the additional capabilities  
136161 provided by POSIX.1-2024.

**136162 D.1.10 Command Language**

136163 Users should be able to define procedures that combine simple tools and/or applications into  
136164 higher-level components that perform to the specific needs of the user. The user should be able  
136165 to store, recall, use, and modify these procedures. These procedures should employ a powerful  
136166 command language that is used for recurring tasks in conforming applications (scripts) in the  
136167 same way that it is used interactively to accomplish one-time tasks. The language and the  
136168 utilities that it uses must be consistent between systems to reduce errors and retraining.

**136169 D.1.11 Interactive Facilities**

136170 Use the system to accomplish individual tasks at an interactive terminal. The interface should be  
136171 consistent, intuitive, and offer usability enhancements to increase the productivity of terminal  
136172 users, reduce errors, and minimize retraining costs. Online documentation or usage assistance  
136173 should be available.

**136174 D.1.12 Accomplish Multiple Tasks Simultaneously**

136175 Access applications and interactive facilities from a single terminal without requiring serial  
136176 execution: switch between multiple interactive tasks; schedule one-time or periodic background  
136177 work; display the status of all work in progress or scheduled; influence the priority scheduling  
136178 of work, when authorized.

**136179 D.1.13 Complex Data Manipulation**

136180 Manipulate data in files in complex ways: sort, merge, compare, translate, edit, format, pattern  
136181 match, select subsets (strings, columns, fields, rows, and so on). These facilities should be  
136182 available to both conforming applications and interactive users.

**136183 D.1.14 File Hierarchy Manipulation**

136184 Create, delete, move/rename, copy, backup/archive, and display files and directories. These  
136185 facilities should be available to both conforming applications and interactive users.

**136186 D.1.15 Locale Configuration**

136187 Customize applications and interactive sessions for the cultural and language conventions of the  
136188 user. Employ a wide variety of standard character encodings. These facilities should be available  
136189 to both conforming applications and interactive users.

**136190 D.1.16 Inter-User Communication**

136191 Send messages or transfer files to other users on the same system or other systems on a network.  
136192 These facilities should be available to both conforming applications and interactive users.

**136193 D.1.17 System Environment**

136194 Display information about the status of the system (activities of users and their interactive and  
136195 background work, file system utilization, system time, configuration, and presence of optional  
136196 facilities) and the environment of the user (terminal characteristics, and so on). Inform the  
136197 system operator/administrator of problems. Control access to user files and other resources.

**136198 D.1.18 Printing**

136199 Output files on a variety of output device classes, accessing devices on local or network-  
136200 connected systems. Control (or influence) the formatting, priority scheduling, and output  
136201 distribution of work. These facilities should be available to both conforming applications and  
136202 interactive users.

**136203 D.1.19 Software Development**

136204 Develop (create and manage source files, compile/interpret, debug) portable open systems  
136205 applications and package them for distribution to, and updating of, other systems.

## 136206 D.2 Portability Capabilities

136207 This section describes the significant portability capabilities of POSIX.1-2024 and indicates how  
 136208 the user requirements listed in [Section D.1](#) (on page 3919) are addressed. The capabilities are  
 136209 listed in the same format as the preceding user requirements; they are summarized below:

- 136210 • Configuration Interrogation
- 136211 • Process Management
- 136212 • Access to Data
- 136213 • Access to the Environment
- 136214 • Access to Determinism and Performance Enhancements
- 136215 • Operating System-Dependent Profile
- 136216 • I/O Interaction
- 136217 • Internationalization Interaction
- 136218 • C-Language Extensions
- 136219 • Command Language
- 136220 • Interactive Facilities
- 136221 • Accomplish Multiple Tasks Simultaneously
- 136222 • Complex Data Manipulation
- 136223 • File Hierarchy Manipulation
- 136224 • Locale Configuration
- 136225 • Inter-User Communication
- 136226 • System Environment
- 136227 • Printing
- 136228 • Software Development

### 136229 D.2.1 Configuration Interrogation

136230 The *uname()* operation provides basic identification of the system. The *sysconf()*, *pathconf()*, and  
 136231 *fpathconf()* functions and the *getconf* utility provide means to interrogate the implementation to  
 136232 determine how to adapt to the environment in which it is running. These values can be either  
 136233 static (indicating that all instances of the implementation have the same value) or dynamic  
 136234 (indicating that different instances of the implementation have the different values, or that the  
 136235 value may vary for other reasons, such as reconfiguration).

### 136236 Unsatisfied Requirements

136237 None directly. However, as new areas are added, there will be a need for additional capability in  
 136238 this area.

## 136239 D.2.2 Process Management

136240 The *fork()*, *exec* family, *posix\_spawn()*, and *posix\_spawnp()* functions provide for the creation of  
 136241 new processes or the insertion of new applications into existing processes. The *\_Exit()*, *\_exit()*,  
 136242 *exit()*, and *abort()* functions allow for the termination of a process by itself. The *wait()*, *waitid()*,  
 136243 and *waitpid()* functions allow one process to deal with the termination of another.

136244 The *times()* function allows for basic measurement of times used by a process. Various  
 136245 functions, including *fstat()*, *getegid()*, *geteuid()*, *getgid()*, *getgrgid()*, *getgrnam()*, *getlogin()*,  
 136246 *getpid()*, *getppid()*, *getpwnam()*, *getpwuid()*, *getuid()*, *lstat()*, and *stat()*, provide for access to the  
 136247 identifiers of processes and the identifiers and names of owners of processes (and files).

136248 The various functions operating on environment variables provide for communication of  
 136249 information (primarily user-configurable defaults) from a parent to child processes.

136250 The operations on the current working directory control and interrogate the directory from  
 136251 which relative pathname searches start. The *umask()* function controls the default protections  
 136252 applied to files created by the process.

136253 The *alarm()*, *pause()*, *sleep()*, *ualarm()*, and *usleep()* operations allow the process to suspend until  
 136254 a timer has expired or to be notified when a period of time has elapsed. The *time()* operation  
 136255 interrogates the current time and date.

136256 The signal mechanism provides for communication of events either from other processes or  
 136257 from the environment to the application, and the means for the application to control the effect  
 136258 of these events. The mechanism provides for external termination of a process and for a process  
 136259 to suspend until an event occurs. The mechanism also provides for a value to be associated with  
 136260 an event.

136261 Job control provides a means to group processes and control them as groups, and to control their  
 136262 access to the function between the user and the system (the “controlling terminal”). It also  
 136263 provides the means to suspend and resume processes.

136264 The Process Scheduling option provides control of the scheduling and priority of a process.

136265 The Message Passing option provides a means for interprocess communication involving small  
 136266 amounts of data.

136267 The Memory Management facilities provide control of memory resources and for the sharing of  
 136268 memory. This functionality is mandatory on POSIX-conforming systems.

136269 The Threads facilities provide multiple flows of control with a process (threads),  
 136270 synchronization between threads (including mutexes, barriers, and spin locks), association of  
 136271 data with threads, and controlled cancellation of threads.

136272 The XSI interprocess communications functionality provide an alternate set of facilities to  
 136273 manipulate semaphores, message queues, and shared memory. These are provided on XSI-  
 136274 conformant systems to support conforming applications developed to run on UNIX systems.

## 136275 D.2.3 Access to Data

136276 The *open()*, *close()*, *fclose()*, *fopen()*, *freopen()*, *pipe()*, and *pipe2()* functions provide for access to  
 136277 files and data. Such files may be regular files, interprocess data channels (pipes), or devices.  
 136278 Additional types of objects in the file system are permitted and are being contemplated for  
 136279 standardization.

136280 The *access()*, *chmod()*, *chown()*, *dup()*, *dup2()*, *dup3()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*,  
 136281 *futimens()*, *lstat()*, *readlink()*, *realpath()*, *stat()*, and *utimensat()* functions allow for control and

136282 interrogation of file and file-related objects (including symbolic links), and their ownership,  
136283 protections, and timestamps.

136284 The *fgetc()*, *fputc()*, *fread()*, *fseek()*, *fsetpos()*, *fwrite()*, *getc()*, *getchar()*, *lseek()*, *putchar()*, *putc()*,  
136285 *read()*, and *write()* functions provide for data transfer from the application to files (in all their  
136286 forms).

136287 The *closedir()*, *link()*, *mkdir()*, *opendir()*, *readdir()*, *rename()*, *rmdir()*, *rewinddir()*, and *unlink()*  
136288 functions provide for a complete set of operations on directories. Directories can arbitrarily  
136289 contain other directories, and a single file can be mentioned in more than one directory.

136290 The *faccessat()*, *openat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*, *renameat()*, *readlinkat()*,  
136291 *symlinkat()*, and *unlinkat()* functions allow for race-free and thread-safe file access. The  
136292 motivation for the introduction of these functions was as follows:

- 136293 • Interfaces taking a pathname may be limited by the maximum length of a pathname  
136294 (`{PATH_MAX}`). The absolute path of files can far exceed this length. The alternative  
136295 solution of changing the working directory and using relative pathnames is not thread-  
136296 safe.
- 136297 • A second motivation is that files accessed outside the current working directory are subject  
136298 to attacks caused by the race condition created by changing any of the elements of the  
136299 pathnames used.
- 136300 • A third motivation is to allow application code which makes use of a virtual current  
136301 working directory for each individual thread. In the alternative model there is only one  
136302 current working directory for all threads.

136303 The file-locking mechanisms provide for advisory locking (protection during transactions) of  
136304 ranges of bytes (in effect, records) in a file.

136305 The *confstr()*, *fpathconf()*, *pathconf()*, and *sysconf()* functions provide for enquiry as to the  
136306 behavior of the system where variability is permitted.

136307 The asynchronous input and output functions *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*,  
136308 *aio\_return()*, *aio\_suspend()*, *aio\_write()*, and *lio\_listio()* provide for initiation and control of  
136309 asynchronous data transfers.

136310 The Synchronized Input and Output option provides for assured commitment of data to media.

#### 136311 D.2.4 Access to the Environment

136312 The operations and types in XBD are provided for access to asynchronous serial devices. The  
136313 primary intended use for these is the controlling terminal for the application (the interaction  
136314 point between the user and the system). They are general enough to be used to control any  
136315 asynchronous serial device. The functions are also general enough to be used with many other  
136316 device types as a user interface when some emulation is provided.

136317 Less detailed access is provided for other device types, but in many instances an application  
136318 need not know whether an object in the file system is a device or a regular file to operate  
136319 correctly.

136320 **Unsatisfied Requirements**

136321 Detailed control of common device classes, specifically magnetic tape, is not provided.

136322 **D.2.5 Bounded (Realtime) Response**136323 The realtime signal functions *sigqueue()*, *sigtimedwait()*, and *sigwaitinfo()* provide queued signals  
136324 and the prioritization of the handling of signals.136325 The SCHED\_FIFO, SCHED\_SPORADIC, and SCHED\_RR scheduling policies provide control  
136326 over processor allocation.136327 The semaphore functions *sem\_clockwait()*, *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*,  
136328 *sem\_open()*, *sem\_post()*, *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*, and *sem\_wait()* provide  
136329 high-performance synchronization.136330 The memory management functions provide memory locking for control of memory allocation,  
136331 file mapping for high performance, and shared memory for high-performance interprocess  
136332 communication. The Message Passing option provides for interprocess communication without  
136333 being dependent on shared memory.136334 The timers functions *clock\_getres()*, *clock\_gettime()*, *clock\_settime()*, *nanosleep()*, *timer\_create()*,  
136335 *timer\_delete()*, *timer\_getoverrun()*, *timer\_gettime()*, and *timer\_settime()* provide functionality to  
136336 manipulate clocks and timers and include a high resolution function called *nanosleep()* with a  
136337 finer resolution than the *sleep()* function.136338 The timeout functions — *pthread\_mutex\_clocklock()*, *pthread\_mutex\_timedlock()*,  
136339 *pthread\_rwlock\_clockrdlock()*, *pthread\_rwlock\_clockwrlock()*, *pthread\_rwlock\_timedrdlock()*,  
136340 *pthread\_rwlock\_timedwrlock()*, *sem\_clockwait()*, and *sem\_timedwait()* — the Typed Memory  
136341 Objects option and the Monotonic Clock facility provide further facilities for applications to use  
136342 to obtain predictable bounded response.136343 **D.2.6 Operating System-Dependent Profile**136344 POSIX.1-2024 makes no distinction between text and binary files. The values of EXIT\_SUCCESS  
136345 and EXIT\_FAILURE are further defined.136346 **Unsatisfied Requirements**136347 None known, but the ISO C standard may contain some additional options that could be  
136348 specified.136349 **D.2.7 I/O Interaction**136350 POSIX.1-2024 defines how each of the ISO C standard *stdio* functions interact with the POSIX.1  
136351 operations, typically specifying the behavior in terms of POSIX.1 operations.

136352 **Unsatisfied Requirements**

136353 None.

136354 **D.2.8 Internationalization Interaction**

136355 The POSIX.1-2024 environment operations provide a means to define the environment for  
 136356 *setlocale()* and time functions such as *ctime()*. The *tzset()* function is provided to set time  
 136357 conversion information.

136358 The *nl\_langinfo()* function is provided to query locale-specific cultural settings.

136359 The multiple concurrent locale functions *duplocale()*, *freelocale()*, *is\*\_l()*, *newlocale()*,  
 136360 *strcasemp\_l()*, *strcoll\_l()*, *strfmon\_l()*, *strncasemp\_l()*, *strxfrm\_l()*, *tolower\_l()*, *toupper\_l()*,  
 136361 *towctrans\_l()*, *towlower()*, *towupper()*, *uselocale()*, *wscasemp\_l()*, *wscoll\_l()*, *wscncasemp\_l()*,  
 136362 *wcsxfrm\_l()*, *wctrans\_l()*, and *wctype\_l()* are provide to support per-thread locale information.

136363 **Unsatisfied Requirements**

136364 None.

136365 **D.2.9 C-Language Extensions**

136366 The *setjmp()* and *longjmp()* functions are not defined to be cognizant of the signal masks defined  
 136367 for POSIX.1. The *sigsetjmp()* and *siglongjmp()* functions are provided to fill this gap.

136368 **Unsatisfied Requirements**

136369 None.

136370 **D.2.10 Command Language**

136371 The shell command language, as described in XCU [Chapter 2](#) (on page 2472), is a common  
 136372 language useful in batch scripts, through an API to high-level languages (for the C-Language  
 136373 Binding option, *system()* and *popen()*) and through an interactive terminal (see the *sh* utility).  
 136374 The shell language has many of the characteristics of a high-level language, but it has been  
 136375 designed to be more suitable for user terminal entry and includes interactive debugging  
 136376 facilities. Through the use of pipelining, many complex commands can be constructed from  
 136377 combinations of data filters and other common components. Shell scripts can be created, stored,  
 136378 recalled, and modified by the user with simple editors.

136379 In addition to the basic shell language, the following utilities offer features that simplify and  
 136380 enhance programmatic access to the utilities and provide features normally found only in high-  
 136381 level languages: *basename*, *bc*, *command*, *dirname*, *echo*, *env*, *expr*, *false*, *printf*, *read*, *sleep*, *tee*, *test*,  
 136382 *time\**,<sup>10</sup> *true*, *wait*, *xargs*, and all of the special built-in utilities in XCU [Section 2.15](#) (on page 2526).

---

136383 10. The utilities listed with an asterisk here and later in this section are present only on systems which support the User Portability Utilities  
 136384 option. There may be further restrictions on the utilities offered with various configuration option combinations; see the individual utility  
 136385 descriptions.

136386 **Unsatisfied Requirements**

136387 None.

136388 **D.2.11 Interactive Facilities**

136389 The utilities offer a common style of command-line interface through conformance to the Utility  
 136390 Syntax Guidelines (see XBD [Section 12.2](#), on page 215) and the common utility defaults (see XCU  
 136391 [Section 1.4](#), on page 2462). The *sh* utility offers an interactive command-line history and editing  
 136392 facility.

136393 The following utilities can be used interactively as well as by scripts; *alias*, *fc*, *mailx*, *unalias*, and  
 136394 *write*.

136395 The following utilities in the User Portability Utilities option provide for interactive use: *ex*, *more*,  
 136396 and *vi*; the *man* utility offers online access to system documentation.

136397 **Unsatisfied Requirements**

136398 The command line interface to individual utilities is as intuitive and consistent as historical  
 136399 practice allows. Work underway based on graphical user interfaces may be more suitable for  
 136400 novice or occasional users of the system.

136401 **D.2.12 Accomplish Multiple Tasks Simultaneously**

136402 The shell command language offers background processing through the asynchronous list  
 136403 command form; see XCU [Section 2.9](#) (on page 2499).

136404 The *nohup* utility makes background processing more robust and usable.

136405 The *kill* utility can terminate background jobs.

136406 The following utilities support periodic job scheduling, control, and display: *at*, *batch*, *crontab*,  
 136407 *nice*, *ps*, and *renice*.

136408 When the User Portability Utilities option is supported, the following utilities allow  
 136409 manipulation of jobs: *bg*, *fg*, and *jobs*.

136410 **Unsatisfied Requirements**

136411 Terminals with multiple windows may be more suitable for some multi-tasking interactive uses  
 136412 than the job control approach in POSIX.1-2024. See the comments on graphical user interfaces in  
 136413 [Section D.2.11](#). The *nice* and *renice* utilities do not necessarily take advantage of complex system  
 136414 scheduling algorithms that are supported by the realtime options within POSIX.1-2024.

136415 **D.2.13 Complex Data Manipulation**

136416 The following utilities address user requirements in this area: *asa*, *awk*, *bc*, *cmp*, *comm*, *csplit*, *cut*,  
 136417 *dd*, *diff*, *ed*, *ex\**, *expand*, *expr*, *find*, *fold*, *grep*, *head*, *join*, *od*, *paste*, *pr*, *printf*, *sed*, *sort*, *split*, *tabs*, *tail*, *tr*,  
 136418 *unexpand*, *uniq*, *uudecode*, *uuencode*, and *wc*.



136419 **Unsatisfied Requirements**

136420 Sophisticated text formatting utilities, such as *troff* or *TeX*, are not included. Standards work in  
136421 the area of SGML may satisfy this.

136422 **D.2.14 File Hierarchy Manipulation**

136423 The following utilities address user requirements in this area: *basename*, *cd*, *chgrp*, *chmod*, *chown*,  
136424 *cksum*, *cp*, *dd*, *df*, *diff*, *dirname*, *du*, *find*, *ls*, *ln*, *mkdir*, *mkfifo*, *mv*, *patch*, *pathchk*, *pax*, *pwd*, *rm*, *rmdir*,  
136425 *test*, and *touch*.

136426 **Unsatisfied Requirements**

136427 Some graphical user interfaces offer more intuitive file manager components that allow file  
136428 manipulation through the use of icons for novice users.

136429 **D.2.15 Locale Configuration**

136430 The standard utilities are affected by the various *LC\_* variables to achieve locale-dependent  
136431 operation: character classification, collation sequences, regular expressions and shell pattern  
136432 matching, date and time formats, numeric formatting, and monetary formatting. When the  
136433 POSIX2\_LOCALEDEF option is supported, applications can provide their own locale definition  
136434 files.

136435 The following utilities address user requirements in this area: *date*, *ed*, *ex\**, *find*, *grep*, *locale*,  
136436 *localedef*, *more\**, *sed*, *sh*, *sort*, *tr*, *uniq*, and *vi\**.

136437 The *iconv()*, *iconv\_close()*, and *iconv\_open()* functions are available to allow an application to  
136438 convert character data between supported character sets.

136439 The *genccat* utility and the *catopen()*, *catclose()*, and *catgets()* functions provide for message  
136440 catalog manipulation.

136441 **Unsatisfied Requirements**

136442 Some aspects of multi-byte character and state-encoded character encodings have not yet been  
136443 addressed. The C-language functions, such as *getopt()*, are generally limited to single-byte  
136444 characters. The effect of the *LC\_MESSAGES* variable on message formats is only suggested at  
136445 this time.

136446 **D.2.16 Inter-User Communication**

136447 The following utilities address user requirements in this area: *cksum*, *mailx*, *mesg*, *patch*, *pax*, *talk*,  
136448 *uudecode*, *uuencode*, *who*, and *write*.

136449 The historical UUCP utilities are included as a separate UUCP Utilities option.

136450 **Unsatisfied Requirements**

136451 None.

136452 **D.2.17 System Environment**136453 The following utilities address user requirements in this area: *chgrp*, *chmod*, *chown*, *df*, *du*, *env*,  
136454 *getconf*, *id*, *logger*, *logname*, *mesg*, *newgrp*, *ps*, *stty*, *tput*, *tty*, *umask*, *uname*, and *who*.136455 The *closelog()*, *openlog()*, *setlogmask()*, and *syslog()* functions provide system logging facilities on  
136456 XSI-conformant systems; these are analogous to the *logger* utility.136457 **Unsatisfied Requirements**

136458 None.

136459 **D.2.18 Printing**136460 The following utilities address user requirements in this area: *pr* and *lp*.136461 **Unsatisfied Requirements**

136462 There are no features to control the formatting or scheduling of the print jobs.

136463 **D.2.19 Software Development**136464 The following utilities address user requirements in this area: *ar*, *asa*, *awk*, *c17*, *ctags*, *getconf*,  
136465 *getopts*, *lex*, *localedef*, *make*, *nm*, *od*, *patch*, *pax*, *strings*, *strip*, *time*, and *yacc*.136466 The *system()*, *popen()*, *pclose()*, *regcomp()*, *regexec()*, *regerror()*, *regfree()*, *fnmatch()*, *getopt()*,  
136467 *glob()*, *globfree()*, *wordexp()*, and *wordfree()* functions allow C-language programmers to access  
136468 some of the interfaces used by the utilities, such as argument processing, regular expressions,  
136469 and pattern matching.136470 The SCCS source-code control system utilities are available on systems supporting the XSI  
136471 Development option.136472 **Unsatisfied Requirements**136473 There are no language-specific development tools related to languages other than C. There is no  
136474 data dictionary or other CASE-like development tools.136475 **D.2.20 Future Growth**136476 It is arguable whether or not all functionality to support applications is potentially within the  
136477 scope of POSIX.1-2024. As a simple matter of practicality, it cannot be. Areas such as graphics,  
136478 application domain-specific functionality, windowing, and so on, should be in unique standards.  
136479 As such, they are properly “Unsatisfied Requirements” in terms of providing fully conforming  
136480 applications, but ones which are outside the scope of POSIX.1-2024.136481 However, as the standards evolve, certain functionality once considered “exotic” enough to be  
136482 part of a separate standard become common enough to be included in a core standard such as  
136483 this. Realtime and networking, for example, have both moved from separate standards (with  
136484 much difficult cross-referencing) into this standard over time, and although no specific areas

136485 have been identified for inclusion in a future version, such inclusions seem likely.

### 136486 **D.3 Profiling Considerations**

136487 This section offers guidance to writers of profiles on how the configurable options, limits, and  
 136488 optional behavior of POSIX.1-2024 should be cited in profiles. Profile writers should consult the  
 136489 general guidance in POSIX.0 when writing POSIX Standardized Profiles.

136490 The information in this section is an inclusive list of features that should be considered by profile  
 136491 writers. Subsetting of POSIX.1-2024 should follow XBD [Section 2.1.5.1](#) (on page 20). A set of  
 136492 profiling options is described in [Appendix E](#) (on page 3943).

#### 136493 **D.3.1 Configuration Options**

136494 There are two set of options suggested by POSIX.1-2024: those for POSIX-conforming systems  
 136495 and those for X/Open System Interface (XSI) conformance. The requirements for XSI  
 136496 conformance are documented in the Base Definitions volume of POSIX.1-2024 and not discussed  
 136497 further here, as they superset the POSIX conformance requirements.

#### 136498 **D.3.2 Configuration Options (Shell and Utilities)**

136499 There are three broad optional configurations for the Shell and Utilities volume of POSIX.1-2024:  
 136500 basic execution system, development system, and user portability interactive system. The  
 136501 options to support these, and other minor configuration options, are listed in XBD [Chapter 2](#) (on  
 136502 page 15). Profile writers should consult the following list and the comments concerning user  
 136503 requirements addressed by various components in [Section D.2](#) (on page 3923).

136504 POSIX2\_UPE

136505 The system supports the User Portability Utilities option.

136506 This option is a requirement for a user portability interactive system. It is required  
 136507 frequently except for those systems, such as embedded realtime or dedicated application  
 136508 systems, that support little or no interactive time-sharing work by users or operators. XSI-  
 136509 conformant systems support this option.

136510 POSIX2\_SW\_DEV

136511 The system supports the Software Development Utilities option.

136512 This option is required by many systems, even those in which actual software development  
 136513 does not occur. The *make* utility, in particular, is required by many application software  
 136514 packages as they are installed onto the system. If POSIX2\_C\_DEV is supported,  
 136515 POSIX2\_SW\_DEV is almost a mandatory requirement because of *ar* and *make*.

136516 POSIX2\_C\_BIND

136517 The system supports the C-Language Bindings option.

136518 This option is required on some implementations developing complex C applications or on  
 136519 any system installing C applications in source form that require the functions in this option.  
 136520 The *system()* and *popen()* functions, in particular, are widely used by applications; the  
 136521 others are rather more specialized.

136522 POSIX2\_C\_DEV

136523 The system supports the C-Language Development Utilities option.

136524 This option is required by many systems, even those in which actual C-language software

136525 development does not occur. The *c17* utility, in particular, is required by many application  
 136526 software packages as they are installed onto the system. The *lex* and *yacc* utilities are used  
 136527 less frequently.

#### 136528 POSIX2\_FORT\_RUN

136529 The system supports the FORTRAN Runtime Utilities option.

136530 This option is required for some FORTRAN applications that need the *asa* utility to convert  
 136531 Hollerith printing statement output. It is unknown how frequently this occurs.

#### 136532 POSIX2\_LOCALEDEF

136533 The system supports the creation of locales.

136534 This option is needed if applications require their own customized locale definitions to  
 136535 operate. It is presently unknown whether many applications are dependent on this.  
 136536 However, the option is virtually mandatory for systems in which internationalized  
 136537 applications are developed.

136538 XSI-conformant systems support this option.

#### 136539 POSIX2\_CHAR\_TERM

136540 The system supports at least one terminal type capable of all operations described in  
 136541 POSIX.1-2024.

136542 On systems with POSIX2\_UPE, this option is almost always required. It was developed  
 136543 solely to allow certain specialized vendors and user applications to bypass the requirement  
 136544 for general-purpose asynchronous terminal support. For example, an application and  
 136545 system that was suitable for block-mode terminals would not need this option.

136546 XSI-conformant systems support this option.

### 136547 D.3.3 Configurable Limits

136548 Very few of the limits need to be increased for profiles. No profile can cite lower values.

136549 {POSIX2\_BC\_BASE\_MAX}

136550 {POSIX2\_BC\_DIM\_MAX}

136551 {POSIX2\_BC\_SCALE\_MAX}

136552 {POSIX2\_BC\_STRING\_MAX}

136553 No increase is anticipated for any of these *bc* values, except for very specialized applications  
 136554 involving huge numbers.

136555 {POSIX2\_COLL\_WEIGHTS\_MAX}

136556 Some natural languages with complex collation requirements require an increase from the  
 136557 default 2 to 4; no higher numbers are anticipated.

136558 {POSIX2\_EXPR\_NEST\_MAX}

136559 No increase is anticipated.

136560 {POSIX2\_LINE\_MAX}

136561 This number is much larger than most historical applications have been able to use. At some  
 136562 future time, applications may be rewritten to take advantage of even larger values.

136563 {POSIX2\_RE\_DUP\_MAX}

136564 No increase is anticipated.

136565 {POSIX2\_VERSION}

136566 This is actually not a limit, but a standard version stamp. Generally, a profile should specify  
 136567 XCU [Chapter 2](#) (on page 2472) by name in the normative references section, not this value.

### 136568 D.3.4 Configuration Options (System Interfaces)

136569 {NGROUPS\_MAX}

136570 A non-zero value indicates that the implementation supports supplementary groups.

136571 This option is needed where there is a large amount of shared use of files, but where a  
 136572 certain amount of protection is needed. Many profiles<sup>11</sup> are known to require this option; it  
 136573 should only be required if needed, but it should never be prohibited.

136574 \_POSIX\_ADVISORY\_INFO

136575 The system provides advisory information for file management.

136576 This option allows the application to specify advisory information that can be used to  
 136577 achieve better or even deterministic response time in file manager or input and output  
 136578 operations.

136579 \_POSIX\_ASYNCHRONOUS\_IO

136580 Support for asynchronous input and output is mandatory in POSIX.1-2024.

136581 \_POSIX\_BARRIERS

136582 Support for barrier synchronization is mandatory in POSIX.1-2024.

136583 This facility allows efficient synchronization of multiple parallel threads in multi-processor  
 136584 systems in which the operation is supported in part by the hardware architecture.

136585 \_POSIX\_CHOWN\_RESTRICTED

136586 The system restricts the right to “give away” files to other users. It is mandatory that an  
 136587 implementation be able to support this facility in POSIX.1-2024; however, it is recognized  
 136588 that implementations need not enable the functionality by default.

136589 Some applications expect that they can change the ownership of files in this way. It is  
 136590 provided where either security or system account requirements cause this ability to be a  
 136591 problem. It is also known to be specified in many profiles.

136592 \_POSIX\_CLOCK\_SELECTION

136593 Support for clock selection is mandatory in POSIX.1-2024.

136594 This facility allows applications to request a high resolution sleep in order to suspend a  
 136595 thread during a relative time interval, or until an absolute time value, using the desired  
 136596 clock. It also allows the application to select the clock used in a *pthread\_cond\_timedwait()*  
 136597 function call.

136598 \_POSIX\_CPUTIME

136599 The system supports the Process CPU-Time Clocks option.

136600 This option allows applications to use a new clock that measures the execution times of  
 136601 processes or threads, and the possibility to create timers based upon these clocks, for  
 136602 runtime detection (and treatment) of execution time overruns.

136603 \_POSIX\_FSYNC

136604 The system supports file synchronization requests.

136605 This option was created to support historical systems that did not provide the feature.  
 136606 Applications that are expecting guaranteed completion of their input and output operations  
 136607 should require the \_POSIX\_SYNC\_IO option. This option should never be prohibited.

136608 XSI-conformant systems support this option.

---

136609 11. There are no formally approved profiles of POSIX.1-2024 at the time of publication; the reference here is to various profiles generated by  
 136610 private bodies or governments.

136611	<code>_POSIX_IPV6</code>	
136612		The system supports facilities related to Internet Protocol Version 6 (IPv6).
136613		This option was created to allow systems to transition to IPv6.
136614	<code>_POSIX_JOB_CONTROL</code>	
136615		Support for job control is mandatory in POSIX.1-2024.
136616		Most applications that use it can run when it is not present, although with a degraded level
136617		of user convenience.
136618	<code>_POSIX_MAPPED_FILES</code>	
136619		Support for memory mapped files is mandatory in POSIX.1-2024.
136620		This facility provides for the mapping of regular files into the process address space.
136621		Both this facility and the Shared Memory Objects option provide shared access to memory
136622		objects in the process address space. The <code>mmap()</code> and <code>munmap()</code> functions provide the
136623		functionality of existing practice for mapping regular files. This functionality was deemed
136624		unnecessary, if not inappropriate, for embedded systems applications and is expected to be
136625		optional in subprofiles.
136626	<code>_POSIX_MEMLOCK</code>	
136627		The system supports the locking of the address space.
136628		This option was created to support historical systems that did not provide the feature. It
136629		should only be required if needed, but it should never be prohibited.
136630	<code>_POSIX_MEMLOCK_RANGE</code>	
136631		The system supports the locking of specific ranges of the address space.
136632		For applications that have well-defined sections that need to be locked and others that do
136633		not, POSIX.1-2024 supports an optional set of functions to lock or unlock a range of process
136634		addresses. The following are two reasons for having a means to lock down a specific range:
136635		1. An asynchronous event handler function that must respond to external events in a
136636		deterministic manner such that page faults cannot be tolerated
136637		2. An input/output “buffer” area that is the target for direct-to-process I/O, and the
136638		overhead of implicit locking and unlocking for each I/O call cannot be tolerated
136639		It should only be required if needed, but it should never be prohibited.
136640	<code>_POSIX_MEMORY_PROTECTION</code>	
136641		Support for memory protection is mandatory in POSIX.1-2024.
136642		The provision of this facility typically imposes additional hardware requirements.
136643	<code>_POSIX_PRIORITIZED_IO</code>	
136644		The system provides prioritization for input and output operations.
136645		The use of this option may interfere with the ability of the system to optimize input and
136646		output throughput. It should only be required if needed, but it should never be prohibited.
136647	<code>_POSIX_MESSAGE_PASSING</code>	
136648		The system supports the passing of messages between processes.
136649		This option was created to support historical systems that did not provide the feature. The
136650		functionality adds a high-performance XSI interprocess communication facility for local
136651		communication. It should only be required if needed, but it should never be prohibited.

- 136652        \_POSIX\_MONOTONIC\_CLOCK  
 136653        Support for a monotonic clock is mandatory in POSIX.1-2024.
- 136654        This facility allows realtime applications to rely on a monotonically increasing clock that  
 136655        does not jump backwards, and whose value does not change except for the regular ticking  
 136656        of the clock.
- 136657        \_POSIX\_PRIORITY\_SCHEDULING  
 136658        The system provides priority-based process scheduling.
- 136659        Support of this option provides predictable scheduling behavior, allowing applications to  
 136660        determine the order in which processes that are ready to run are granted access to a  
 136661        processor. It should only be required if needed, but it should never be prohibited.
- 136662        \_POSIX\_REALTIME\_SIGNALS  
 136663        Support for realtime signals is mandatory in POSIX.1-2024.
- 136664        This facility provides prioritized, queued signals with associated data values.
- 136665        \_POSIX\_REGEX  
 136666        Support for regular expression facilities is mandatory in POSIX.1-2024.
- 136667        \_POSIX\_SAVED\_IDS  
 136668        Support for this feature is mandatory in POSIX.1-2024.
- 136669        Certain classes of applications rely on it for proper operation, and there is no alternative  
 136670        short of giving the application root privileges on most implementations that did not provide  
 136671        \_POSIX\_SAVED\_IDS.
- 136672        \_POSIX\_SEMAPHORES  
 136673        Support for counting semaphores is mandatory in POSIX.1-2024.
- 136674        \_POSIX\_SHARED\_MEMORY\_OBJECTS  
 136675        The system supports the mapping of shared memory objects into the process address space.
- 136676        Both this option and the Memory Mapped Files option provide shared access to memory  
 136677        objects in the process address space. The functions defined under this option provide the  
 136678        functionality of existing practice for shared memory objects. This functionality was deemed  
 136679        appropriate for embedded systems applications and, hence, is provided under this option.  
 136680        It should only be required if needed, but it should never be prohibited.
- 136681        \_POSIX\_SHELL  
 136682        Support for the *sh* utility command line interpreter is mandatory in POSIX.1-2024.
- 136683        \_POSIX\_SPAWN  
 136684        The system supports the spawn option.
- 136685        This option provides applications with an efficient mechanism to spawn execution of a new  
 136686        process.
- 136687        \_POSIX\_SPINLOCKS  
 136688        Support for spin locks is mandatory in POSIX.1-2024.
- 136689        This facility provides a simple and efficient synchronization mechanism for threads  
 136690        executing in multi-processor systems.
- 136691        \_POSIX\_SPORADIC\_SERVER  
 136692        The system supports the sporadic server scheduling policy.
- 136693        This option provides applications with a new scheduling policy for scheduling aperiodic  
 136694        processes or threads in hard realtime applications.

- 136695 `_POSIX_SYNCHRONIZED_IO`  
136696 The system supports guaranteed file synchronization.
- 136697 This option was created to support historical systems that did not provide the feature.  
136698 Applications that are expecting guaranteed completion of their input and output operations  
136699 should require this option, rather than the File Synchronization option. It should only be  
136700 required if needed, but it should never be prohibited.
- 136701 `_POSIX_THREADS`  
136702 Support for multiple threads of control within a single process is mandatory in  
136703 POSIX.1-2024.
- 136704 `_POSIX_THREAD_ATTR_STACKADDR`  
136705 The system supports specification of the stack address for a created thread.
- 136706 Applications may take advantage of support of this option for performance benefits, but  
136707 dependence on this feature should be minimized. This option should never be prohibited.
- 136708 XSI-conformant systems support this option.
- 136709 `_POSIX_THREAD_ATTR_STACKSIZE`  
136710 The system supports specification of the stack size for a created thread.
- 136711 Applications may require this option in order to ensure proper execution, but such usage  
136712 limits portability and dependence on this feature should be minimized. It should only be  
136713 required if needed, but it should never be prohibited.
- 136714 XSI-conformant systems support this option.
- 136715 `_POSIX_THREAD_PRIORITY_SCHEDULING`  
136716 The system provides priority-based thread scheduling.
- 136717 Support of this option provides predictable scheduling behavior, allowing applications to  
136718 determine the order in which threads that are ready to run are granted access to a processor.  
136719 It should only be required if needed, but it should never be prohibited.
- 136720 `_POSIX_THREAD_PRIO_INHERIT`  
136721 The system provides mutual-exclusion operations with priority inheritance.
- 136722 Support of this option provides predictable scheduling behavior, allowing applications to  
136723 determine the order in which threads that are ready to run are granted access to a processor.  
136724 It should only be required if needed, but it should never be prohibited.
- 136725 `_POSIX_THREAD_PRIO_PROTECT`  
136726 The system supports a priority ceiling emulation protocol for mutual-exclusion operations.
- 136727 Support of this option provides predictable scheduling behavior, allowing applications to  
136728 determine the order in which threads that are ready to run are granted access to a processor.  
136729 It should only be required if needed, but it should never be prohibited.
- 136730 `_POSIX_THREAD_PROCESS_SHARED`  
136731 The system provides shared access among multiple processes to synchronization objects.
- 136732 This option was created to support historical systems that did not provide the feature. It  
136733 should only be required if needed, but it should never be prohibited.
- 136734 XSI-conformant systems support this option.
- 136735 `_POSIX_THREAD_SAFE_FUNCTIONS`  
136736 Support for thread-safe functions is mandatory in POSIX.1-2024.



- 136737 `_POSIX_THREAD_SPORADIC_SERVER`  
 136738 The system supports the thread sporadic server scheduling policy.
- 136739 Support for this option provides applications with a new scheduling policy for scheduling  
 136740 aperiodic threads in hard realtime applications.
- 136741 `_POSIX_TIMEOUTS`  
 136742 Support for timeouts for some blocking services is mandatory in POSIX.1-2024.
- 136743 `_POSIX_TIMERS`  
 136744 Support for higher resolution clocks with multiple timers per process is mandatory in  
 136745 POSIX.1-2024.
- 136746 This facility is appropriate for applications requiring higher resolution timestamps or  
 136747 needing to control the timing of multiple activities.
- 136748 `_POSIX_TYPED_MEMORY_OBJECTS`  
 136749 The system supports the Typed Memory Objects option.
- 136750 This option was created to allow realtime applications to access different kinds of physical  
 136751 memory, and allow processes in these applications to share portions of this memory.

### 136752 D.3.5 Configurable Limits

- 136753 In general, the configurable limits in the `<limits.h>` header defined in the Base Definitions  
 136754 volume of POSIX.1-2024 have been set to minimal values; many applications or implementations  
 136755 may require larger values. No profile can cite lower values.
- 136756 `{AIO_LISTIO_MAX}`  
 136757 The current minimum is likely to be inadequate for most applications. It is expected that  
 136758 this value will be increased by profiles requiring support for list input and output  
 136759 operations.
- 136760 `{AIO_MAX}`  
 136761 The current minimum is likely to be inadequate for most applications. It is expected that  
 136762 this value will be increased by profiles requiring support for asynchronous input and  
 136763 output operations.
- 136764 `{AIO_PRIO_DELTA_MAX}`  
 136765 The functionality associated with this limit is needed only by sophisticated applications. It  
 136766 is not expected that this limit would need to be increased under a general-purpose profile.
- 136767 `{ARG_MAX}`  
 136768 The current minimum is likely to need to be increased for profiles, particularly as larger  
 136769 amounts of information are passed through the environment. Many implementations are  
 136770 believed to support larger values.
- 136771 `{CHILD_MAX}`  
 136772 The current minimum is suitable only for systems where a single user is not running  
 136773 applications in parallel. It is significantly too low for any system also requiring windows,  
 136774 and if `_POSIX_JOB_CONTROL` is specified, it should be raised.
- 136775 `{CLOCKRES_MIN}`  
 136776 It is expected that profiles will require a finer granularity clock, perhaps as fine as 1  $\mu$ s,  
 136777 represented by a value of 1 000 for this limit.
- 136778 `{DELAYTIMER_MAX}`  
 136779 It is believed that most implementations will provide larger values.


136780	{LINK_MAX}
136781	For most applications and usage, the current minimum is adequate. Many implementations
136782	have a much larger value, but this should not be used as a basis for raising the value unless
136783	the applications to be used require it.
136784	{LOGIN_NAME_MAX}
136785	This is not actually a limit, but an implementation parameter. No profile should impose a
136786	requirement on this value.
136787	{MAX_CANON}
136788	For most purposes, the current minimum is adequate. Unless high-speed burst serial
136789	devices are used, it should be left as is.
136790	{MAX_INPUT}
136791	See {MAX_CANON}.
136792	{MQ_OPEN_MAX}
136793	The current minimum should be adequate for most profiles.
136794	{MQ_PRIO_MAX}
136795	The current minimum corresponds to the required number of process scheduling priorities.
136796	Many realtime practitioners believe that the number of message priority levels ought to be
136797	the same as the number of execution scheduling priorities.
136798	{NAME_MAX}
136799	Many implementations now support larger values, and many applications and users
136800	assume that larger names can be used. Many existing profiles also specify a larger value.
136801	Specifying this value will reduce the number of conforming implementations, although this
136802	might not be a significant consideration over time. Values greater than 255 should not be
136803	required.
136804	{NGROUPS_MAX}
136805	The value selected will typically be 8 or larger.
136806	{OPEN_MAX}
136807	The historically common value for this has been 20. Many implementations support larger
136808	values. If applications that use larger values are anticipated, an appropriate value should be
136809	specified.
136810	{PAGESIZE}
136811	This is not actually a limit, but an implementation parameter. No profile should impose a
136812	requirement on this value.
136813	{PATH_MAX}
136814	Historically, the minimum has been either 1024 or indefinite, depending on the
136815	implementation. Few applications actually require values larger than 256, but some users
136816	may create file hierarchies that must be accessed with longer paths. This value should only
136817	be changed if there is a clear requirement.
136818	{PIPE_BUF}
136819	The current minimum is adequate for most applications. Historically, it has been larger. If
136820	applications that write single transactions larger than this are anticipated, it should be
136821	increased. Applications that write lines of text larger than this probably do not need it
136822	increased, as the text line is delimited by a <newline>.
136823	{POSIX_VERSION}
136824	This is actually not a limit, but a standard version stamp. Generally, a profile should specify
136825	POSIX.1-2024 by a name in the normative references section, not this value.

136826	{PTHREAD_DESTRUCTOR_ITERATIONS}
136827	It is unlikely that applications will need larger values to avoid loss of memory resources.
136828	{PTHREAD_KEYS_MAX}
136829	The current value should be adequate for most profiles.
136830	{PTHREAD_STACK_MIN}
136831	This should not be treated as an actual limit, but as an implementation parameter. No
136832	profile should impose a requirement on this value.
136833	{PTHREAD_THREADS_MAX}
136834	It is believed that most implementations will provide larger values.
136835	{RTSIG_MAX}
136836	The current limit was chosen so that the set of POSIX.1 signal numbers can fit within a
136837	32-bit field. It is recognized that most existing implementations define many more signals
136838	than are specified in POSIX.1 and, in fact, many implementations have already exceeded 32
136839	signals (including the ``null signal’’). Support of {_POSIX_RTSIG_MAX} additional signals
136840	may push some implementations over the single 32-bit word line, but is unlikely to push
136841	any implementations that are already over that line beyond the 64 signal line.
136842	{SEM_NSEMS_MAX}
136843	The current value should be adequate for most profiles.
136844	{SEM_VALUE_MAX}
136845	The current value should be adequate for most profiles.
136846	{SSIZE_MAX}
136847	This limit reflects fundamental hardware characteristics (the size of an integer), and should
136848	not be specified unless it is clearly required. Extreme care should be taken to assure that
136849	any value that might be specified does not unnecessarily eliminate implementations
136850	because of accidents of hardware design.
136851	{STREAM_MAX}
136852	This limit is very closely related to {OPEN_MAX}. It should never be larger than
136853	{OPEN_MAX}, but could reasonably be smaller for application areas where most files are
136854	not accessed through <i>stdio</i> . Some implementations may limit {STREAM_MAX} to 20 but
136855	allow {OPEN_MAX} to be considerably larger. Such implementations should be allowed for
136856	if the applications permit.
136857	{TIMER_MAX}
136858	The current limit should be adequate for most profiles, but it may need to be larger for
136859	applications with a large number of asynchronous operations.
136860	{TTY_NAME_MAX}
136861	This is not actually a limit, but an implementation parameter. No profile should impose a
136862	requirement on this value.
136863	{TZNAME_MAX}
136864	The minimum has been historically adequate, but if longer timezone names are anticipated
136865	(particularly such values as UTC-1), this should be increased.

**D.3.6 Optional Behavior**

136867 In POSIX.1-2024, there are no instances of the terms unspecified, undefined, implementation-  
136868 defined, or with the verbs “may” or “need not”, that the standard developers anticipate or  
136869 sanction as suitable for profile or test method citation. All of these are merely warnings to  
136870 conforming applications to avoid certain areas that can vary from system to system, and even  
136871 over time on the same system. In many cases, these terms are used explicitly to support  
136872 extensions, but profiles should not anticipate and require such extensions; future versions of this  
136873 standard may do so.

136874

 *Rationale (Informative)*

136875

**Part E:**

136876

**Subprofiling Considerations**

136877

*The Open Group*

136878

*The Institute of Electrical and Electronics Engineers, Inc.*



## Subprofiling Considerations (Informative)

136881 This section contains further information to satisfy the requirement that the project scope enable  
 136882 subprofiling of POSIX.1-2024. The approach taken is to include a general requirement in  
 136883 normative text regarding subprofiling and to include an informative section (here) containing a  
 136884 proposed set of subprofiling options.

### 136885 E.1 Subprofiling Option Groups

136886 The following Option Groups<sup>12</sup> are defined to support profiling. Systems claiming support to  
 136887 POSIX.1-2024 need not implement these options apart from the requirements stated in XBD  
 136888 Section 2.1.3 (on page 17). These Option Groups allow profiles to subset the System Interfaces  
 136889 volume of POSIX.1-2024 by collecting sets of related functions and generic functions.

136890 POSIX\_ASYNCHRONOUS\_IO: Asynchronous Input and Output Functions

136891 *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*, *aio\_return()*, *aio\_suspend()*, *aio\_write()*,  
 136892 *lio\_listio()*

136893 POSIX\_BARRIERS: Barriers

136894 *pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*,  
 136895 *pthread\_barrierattr\_destroy()*, *pthread\_barrierattr\_getpshared()*, *pthread\_barrierattr\_init()*,  
 136896 *pthread\_barrierattr\_setpshared()*

136897 POSIX\_C\_LANG\_ATOMICS: ISO C Atomic Operations

136898 *atomic\_compare\_exchange\_strong()*, *atomic\_compare\_exchange\_strong\_explicit()*,  
 136899 *atomic\_compare\_exchange\_weak()*, *atomic\_compare\_exchange\_weak\_explicit()*, *atomic\_exchange()*,  
 136900 *atomic\_exchange\_explicit()*, *atomic\_fetch\_add()*, *atomic\_fetch\_add\_explicit()*, *atomic\_fetch\_and()*,  
 136901 *atomic\_fetch\_and\_explicit()*, *atomic\_fetch\_or()*, *atomic\_fetch\_or\_explicit()*, *atomic\_fetch\_sub()*,  
 136902 *atomic\_fetch\_sub\_explicit()*, *atomic\_fetch\_xor()*, *atomic\_fetch\_xor\_explicit()*, *atomic\_flag\_clear()*,  
 136903 *atomic\_flag\_clear\_explicit()*, *atomic\_flag\_test\_and\_set()*, *atomic\_flag\_test\_and\_set\_explicit()*,  
 136904 *atomic\_init()*, *atomic\_is\_lock\_free()*, *atomic\_load()*, *atomic\_load\_explicit()*, *atomic\_signal\_fence()*,  
 136905 *atomic\_thread\_fence()*, *atomic\_store()*, *atomic\_store\_explicit()*, *kill\_dependency()*

136906 POSIX\_C\_LANG\_JUMP: Jump Functions

136907 *longjmp()*, *setjmp()*

136908 POSIX\_C\_LANG\_MATH: Maths Library

136909 *CMPLX()*, *CMPLXF()*, *CMPLXL()*, *acos()*, *acosf()*, *acosh()*, *acoshf()*, *acoshl()*, *acosl()*, *asin()*,  
 136910 *asinf()*, *asinh()*, *asinhf()*, *asinhf()*, *asinl()*, *atan()*, *atan2()*, *atan2f()*, *atan2l()*, *atanf()*, *atanh()*,  
 136911 *atanhf()*, *atanhl()*, *atanl()*, *cabs()*, *cabsf()*, *cabsl()*, *cacos()*, *cacosf()*, *cacosh()*, *cacoshf()*,  
 136912 *cacoshl()*, *cacosl()*, *carg()*, *cargf()*, *cargl()*, *casin()*, *casinf()*, *casinh()*, *casinhf()*, *casinhl()*,  
 136913 *casinl()*, *catan()*, *catanf()*, *catanh()*, *catanhf()*, *catanhf()*, *catanhl()*, *catanl()*, *cbrt()*, *cbrtf()*, *cbrtl()*, *ccos()*,  
 136914 *ccosf()*, *ccosh()*, *ccoshf()*, *ccoshl()*, *ccosl()*, *ceil()*, *ceilf()*, *ceill()*, *cexp()*, *cexpf()*, *cexpl()*, *cimag()*,  
 136915 *cimagf()*, *cimagl()*, *clog()*, *clogf()*, *clogl()*, *conj()*, *conjf()*, *conjl()*, *copysign()*, *copysignf()*,  
 136916 *copysignl()*, *cos()*, *cosf()*, *cosh()*, *coshf()*, *coshl()*, *cosl()*, *cpow()*, *cpowf()*, *cpowl()*, *cproj()*,  
 136917 *cprojf()*, *cprojl()*, *creal()*, *crealf()*, *creall()*, *csin()*, *csinf()*, *csinh()*, *csinhf()*, *csinhl()*, *csinl()*,  
 136918 *csqrt()*, *csqrtf()*, *csqrtl()*, *ctan()*, *ctanf()*, *ctanh()*, *ctanhf()*, *ctanhf()*, *ctanhl()*, *ctanl()*, *erf()*, *erfc()*, *erfcf()*,

136919 12. These are modeled on the Units of Functionality from IEEE Std 1003.13-1998.

136920 *erfc1(), erfff(), erfl(), exp(), exp2(), exp2f(), exp2l(), expf(), expl(), expm1(), expm1f(), expm1l(),*  
 136921 *fabs(), fabsf(), fabsl(), fdim(), fdimf(), fdiml(), floor(), floorf(), floorl(), fma(), fmaf(), fnal(),*  
 136922 *fmax(), fmaxf(), fmaxl(), fmin(), fminf(), fminl(), fmod(), fmodf(), fmodl(), fpclassify(), frexp(),*  
 136923 *frexpf(), frexpl(), hypot(), hypotf(), hypotl(), ilogb(), ilogbf(), ilogbl(), isfinite(), isgreater(),*  
 136924 *isgreaterequal(), isinf(), isless(), islessequal(), islessgreater(), isnan(), isnormal(), isunordered(),*  
 136925 *ldexp(), ldexpf(), ldexpl(), lgamma(), lgammaf(), lgammal(), llrint(), llrintf(), llrintl(),*  
 136926 *llround(), llroundf(), llroundl(), log(), log10(), log10f(), log10l(), log1p(), log1pf(), log1pl(),*  
 136927 *log2(), log2f(), log2l(), logb(), logbf(), logbl(), logf(), logl(), lrint(), lrintf(), lrintl(), lround(),*  
 136928 *lroundf(), lroundl(), modf(), modff(), modfl(), nan(), nanf(), nanl(), nearbyint(), nearbyintf(),*  
 136929 *nearbyintl(), nextafter(), nextafterf(), nextafterl(), nexttoward(), nexttowardf(), nexttowardl(),*  
 136930 *pow(), powf(), powl(), remainder(), remainderf(), remainderl(), remquo(), remquoof(), remquoofl(),*  
 136931 *rint(), rintf(), rintl(), round(), roundf(), roundl(), scalbln(), scalblnf(), scalblnl(), scalbn(),*  
 136932 *scalbnf(), scalbnl(), signbit(), sin(), sinf(), sinh(), sinhf(), sinhl(), sinl(), sqrt(), sqrtf(), sqrtl(),*  
 136933 *tan(), tanf(), tanh(), tanhf(), tanhl(), tanl(), tgamma(), tgammaf(), tgammaof(), trunc(),*  
 136934 *truncf(), truncf()*

#### 136935 POSIX\_C\_LANG\_SUPPORT: General ISO C Library

136936 *abs(), aligned\_alloc(), asctime(), atof(), atoi(), atol(), atoll(), bsearch(), calloc(), ctime(),*  
 136937 *difftime(), div(), feclearexcept(), fegetenv(), fegetexceptflag(), fegetround(), feholdexcept(),*  
 136938 *feraiseexcept(), fesetenv(), fesetexceptflag(), fesetround(), fetestexcept(), feupdateenv(), free(),*  
 136939 *gmtime(), imaxabs(), imaxdiv(), isalnum(), isalpha(), isblank(), iscntrl(), isdigit(), isgraph(),*  
 136940 *islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit(), labs(), ldiv(), llabs(), lldiv(),*  
 136941 *localeconv(), localtime(), malloc(), memchr(), memcmp(), memcpy(), memmove(), memset(),*  
 136942 *mktime(), qsort(), rand(), realloc(), setlocale(), snprintf(), sprintf(), srand(), sscanf(), strcat(),*  
 136943 *strchr(), strcmp(), strcoll(), strcpy(), strcspn(), strerror(), strptime(), strlen(), strncat(),*  
 136944 *strncmp(), strncpy(), strpbrk(), strrchr(), strspn(), strstr(), strtod(), strtodf(), strtointmax(),*  
 136945 *strtok(), strtol(), strtold(), strtoll(), strtoul(), strtoull(), strtoumax(), strxfrm(), time(),*  
 136946 *timespec\_get(), tolower(), toupper(), tzname, tzset(), va\_arg(), va\_copy(), va\_end(), va\_start(),*  
 136947 *vsprintf(), vsprintf(), vsscanf()*

#### 136948 POSIX\_C\_LANG\_SUPPORT\_R: Thread-Safe General ISO C Library

136949 *gmtime\_r(), localtime\_r(), qsort\_r(), strerror\_r(), strtok\_r()*

#### 136950 POSIX\_C\_LANG\_THREADS: ISO C Threads

136951 *call\_once(), cnd\_broadcast(), cnd\_signal(), cnd\_destroy(), cnd\_init(), cnd\_timedwait(),*  
 136952 *cnd\_wait(), mtx\_destroy(), mtx\_init(), mtx\_lock(), mtx\_timedlock(), mtx\_trylock(),*  
 136953 *mtx\_unlock(), thrd\_create(), thrd\_current(), thrd\_detach(), thrd\_equal(), thrd\_exit(),*  
 136954 *thrd\_join(), thrd\_sleep(), thrd\_yield(), tss\_create(), tss\_delete(), tss\_get(), tss\_set()*

#### 136955 POSIX\_C\_LANG\_UCHAR: ISO C Unicode Utilities

136956 *c16rtomb(), c32rtomb(), mbrtoc16(), mbrtoc32()*

#### 136957 POSIX\_C\_LANG\_WIDE\_CHAR: Wide-Character ISO C Library

136958 *btowc(), iswalnum(), iswalnum(), iswalpha(), iswblank(), iswcntrl(), iswctype(), iswdigit(), iswgraph(),*  
 136959 *iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), mblen(), mbrlen(),*  
 136960 *mbrtowc(), mbsinit(), mbsrtowcs(), mbstowcs(), mbtowc(), swprintf(), swscanf(), towctrans(),*  
 136961 *towlower(), towupper(), vswprintf(), vswscanf(), wcrctomb(), wcscat(), wcschr(), wcsncmp(),*  
 136962 *wcscoll(), wcscpy(), wcscspn(), wcsftime(), wcslen(), wcsncat(), wcsncmp(), wcsncpy(),*  
 136963 *wcspbrk(), wcsrchr(), wcsrtombs(), wcspn(), wcsstr(), wcstod(), wcstof(), wcstointmax(),*  
 136964 *wcstok(), wcstol(), wcstold(), wcstoll(), wcstombs(), wcstoul(), wcstoull(), wcstoumax(),*  
 136965 *wcsxfrm(), wctob(), wctomb(), wctrans(), wctype(), wmemchr(), wmemcmp(), wmemcpy(),*  
 136966 *wmemmove(), wmemset()*

#### 136967 POSIX\_C\_LANG\_WIDE\_CHAR\_EXT: Extended Wide-Character ISO C Library

136968 *mbsnrtowcs(), wcpcpy(), wcpncpy(), wcscasecmp(), wcsdup(), wcslcat(), wcslcpy(),*  
 136969 *wcsncasecmp(), wcsnlen(), wcsnrtombs()*



136970	POSIX_C_LIB_EXT: General C Library Extension
136971	<i>fnmatch()</i> , <i>getentropy()</i> , <i>getopt()</i> , <i>getsubopt()</i> , <i>memmem()</i> , <i>optarg</i> , <i>opterr</i> , <i>optind</i> , <i>optopt</i> ,
136972	<i>reallocarray()</i> , <i>stpncpy()</i> , <i>stpncpy()</i> , <i>strncasecmp()</i> , <i>strdup()</i> , <i>strfmon()</i> , <i>strlcat()</i> , <i>strncpy()</i> ,
136973	<i>strncasecmp()</i> , <i>strndup()</i> , <i>strlen()</i>
136974	POSIX_CLOCK_SELECTION: Clock Selection
136975	<i>clock_nanosleep()</i> , <i>pthread_condattr_getclock()</i> , <i>pthread_condattr_setclock()</i>
136976	POSIX_DEVICE_IO: Device Input and Output
136977	<i>FD_CLR()</i> , <i>FD_ISSET()</i> , <i>FD_SET()</i> , <i>FD_ZERO()</i> , <i>clearerr()</i> , <i>close()</i> , <i>fclose()</i> , <i>fdopen()</i> , <i>feof()</i> ,
136978	<i>ferror()</i> , <i>fflush()</i> , <i>fgetc()</i> , <i>fgets()</i> , <i>fileno()</i> , <i>fopen()</i> , <i>fprintf()</i> , <i>fputc()</i> , <i>fputs()</i> , <i>fread()</i> , <i>freopen()</i> ,
136979	<i>fscanf()</i> , <i>fwrite()</i> , <i>getc()</i> , <i>getchar()</i> , <i>open()</i> , <i>perror()</i> , <i>poll()</i> , <i>posix_close()</i> , <i>ppoll()</i> , <i>printf()</i> ,
136980	<i>pread()</i> , <i>pselect()</i> , <i>putc()</i> , <i>putchar()</i> , <i>puts()</i> , <i>pwrite()</i> , <i>read()</i> , <i>scanf()</i> , <i>select()</i> , <i>setbuf()</i> ,
136981	<i>setvbuf()</i> , <i>stderr</i> , <i>stdin</i> , <i>stdout</i> , <i>ungetc()</i> , <i>vfprintf()</i> , <i>vscanf()</i> , <i>vprintf()</i> , <i>vscanf()</i> , <i>write()</i>
136982	POSIX_DEVICE_IO_EXT: Extended Device Input and Output
136983	<i>asprintf()</i> , <i>dprintf()</i> , <i>fmemopen()</i> , <i>open_memstream()</i> , <i>vasprintf()</i> , <i>vdprintf()</i>
136984	POSIX_DEVICE_SPECIFIC: General Terminal
136985	<i>cfgetispeed()</i> , <i>cfgetospeed()</i> , <i>cfsetispeed()</i> , <i>cfsetospeed()</i> , <i>ctermid()</i> , <i>isatty()</i> , <i>tcdrain()</i> , <i>tcflow()</i> ,
136986	<i>tcflush()</i> , <i>tcgetattr()</i> , <i>tcgetwinsize()</i> , <i>tcsendbreak()</i> , <i>tcsetattr()</i> , <i>tcsetwinsize()</i> , <i>ttynamename()</i>
136987	POSIX_DEVICE_SPECIFIC_R: Thread-Safe General Terminal
136988	<i>ttynamename_r()</i>
136989	POSIX_DYNAMIC_LINKING: Dynamic Linking
136990	<i>dladdr()</i> , <i>dlclose()</i> , <i>dlderror()</i> , <i>dlopen()</i> , <i>dlsym()</i>
136991	POSIX_FD_MGMT: File Descriptor Management
136992	<i>dup()</i> , <i>dup2()</i> , <i>dup3()</i> , <i>fcntl()</i> , <i>fgetpos()</i> , <i>fseek()</i> , <i>fseeko()</i> , <i>fsetpos()</i> , <i>ftell()</i> , <i>ftello()</i> , <i>ftruncate()</i> ,
136993	<i>lseek()</i> , <i>rewind()</i>
136994	POSIX_FIFO: FIFO
136995	<i>mkfifo()</i>
136996	POSIX_FIFO_FD: FIFO File Descriptor Routines
136997	<i>mkfifoat()</i> , <i>mknodat()</i>
136998	POSIX_FILE_ATTRIBUTES: File Attributes
136999	<i>chmod()</i> , <i>chown()</i> , <i>fchmod()</i> , <i>fchown()</i> , <i>umask()</i>
137000	POSIX_FILE_ATTRIBUTES_FD: File Attributes File Descriptor Routines
137001	<i>fchmodat()</i> , <i>fchownat()</i>
137002	POSIX_FILE_LOCKING: Thread-Safe Stdio Locking
137003	<i>flockfile()</i> , <i>ftrylockfile()</i> , <i>funlockfile()</i> , <i>getc_unlocked()</i> , <i>getchar_unlocked()</i> , <i>putc_unlocked()</i> ,
137004	<i>putchar_unlocked()</i>
137005	POSIX_FILE_SYSTEM: File System
137006	<i>access()</i> , <i>chdir()</i> , <i>closedir()</i> , <i>creat()</i> , <i>fchdir()</i> , <i>fpathconf()</i> , <i>fstat()</i> , <i>fstatvfs()</i> , <i>futimens()</i> , <i>getcwd()</i> ,
137007	<i>link()</i> , <i>mkdir()</i> , <i>mkostemp()</i> , <i>mkstemp()</i> , <i>opendir()</i> , <i>pathconf()</i> , <i>posix_getdents()</i> , <i>readdir()</i> ,
137008	<i>remove()</i> , <i>rename()</i> , <i>rewinddir()</i> , <i>rmdir()</i> , <i>stat()</i> , <i>statvfs()</i> , <i>tmpfile()</i> , <i>tmpnam()</i> , <i>truncate()</i> ,
137009	<i>unlink()</i>
137010	POSIX_FILE_SYSTEM_EXT: File System Extensions
137011	<i>alphasort()</i> , <i>dirfd()</i> , <i>getdelim()</i> , <i>getline()</i> , <i>mkdtemp()</i> , <i>scandir()</i>
137012	POSIX_FILE_SYSTEM_FD: File System File Descriptor Routines
137013	<i>faccessat()</i> , <i>fdopendir()</i> , <i>fstatat()</i> , <i>linkat()</i> , <i>mkdirat()</i> , <i>openat()</i> , <i>renameat()</i> , <i>unlinkat()</i> ,
137014	<i>utimensat()</i>

137015	POSIX_FILE_SYSTEM_GLOB: File System Glob Expansion
137016	<i>glob()</i> , <i>globfree()</i>
137017	POSIX_FILE_SYSTEM_R: Thread-Safe File System
137018	<i>readdir_r()</i>
137019	POSIX_I18N: Internationalization
137020	<i>bind_textdomain_codeset()</i> , <i>bindtextdomain()</i> , <i>catclose()</i> , <i>catgets()</i> , <i>catopen()</i> , <i>dcgettext()</i> ,
137021	<i>dcgettext_l()</i> , <i>dcngettext()</i> , <i>dcngettext_l()</i> , <i>dgettext()</i> , <i>dgettext_l()</i> , <i>dngettext()</i> , <i>dngettext_l()</i> ,
137022	<i>gettext()</i> , <i>gettext_l()</i> , <i>iconv()</i> , <i>iconv_close()</i> , <i>iconv_open()</i> , <i>ngettext()</i> , <i>ngettext_l()</i> ,
137023	<i>nl_langinfo()</i> , <i>nl_langinfo_l()</i> , <i>textdomain()</i>
137024	POSIX_JOB_CONTROL: Job Control
137025	<i>setpgid()</i> , <i>tcgetpgrp()</i> , <i>tcsetpgrp()</i> , <i>tcgetsid()</i>
137026	POSIX_MAPPED_FILES: Memory Mapped Files
137027	<i>mmap()</i> , <i>munmap()</i>
137028	POSIX_MEMORY_PROTECTION: Memory Protection
137029	<i>mprotect()</i>
137030	POSIX_MULTI_CONCURRENT_LOCALES: Multiple Concurrent Locales
137031	<i>duplocale()</i> , <i>freelocale()</i> , <i>getlocalename_l()</i> , <i>isalnum_l()</i> , <i>isalpha_l()</i> , <i>isblank_l()</i> , <i>iscntrl_l()</i> ,
137032	<i>isdigit_l()</i> , <i>isgraph_l()</i> , <i>islower_l()</i> , <i>isprint_l()</i> , <i>ispunct_l()</i> , <i>isspace_l()</i> , <i>isupper_l()</i> ,
137033	<i>iswalnum_l()</i> , <i>iswalphal_l()</i> , <i>iswblank_l()</i> , <i>iswcntrl_l()</i> , <i>iswctype_l()</i> , <i>iswdigit_l()</i> , <i>iswgraph_l()</i> ,
137034	<i>iswlower_l()</i> , <i>iswprint_l()</i> , <i>iswpunct_l()</i> , <i>iswspace_l()</i> , <i>iswupper_l()</i> , <i>iswxdigit_l()</i> , <i>isxdigit_l()</i> ,
137035	<i>newlocale()</i> , <i>strcaselmp_l()</i> , <i>strcoll_l()</i> , <i>strerror_l()</i> , <i>strfmon_l()</i> , <i>strftime_l()</i> , <i>strncaselmp_l()</i> ,
137036	<i>strxfrm_l()</i> , <i>tolower_l()</i> , <i>toupper_l()</i> , <i>towctrans_l()</i> , <i>towlower_l()</i> , <i>towupper_l()</i> , <i>uselocale()</i> ,
137037	<i>wscaselmp_l()</i> , <i>wscoll_l()</i> , <i>wscncaselmp_l()</i> , <i>wcsxfrm_l()</i> , <i>wctrans_l()</i> , <i>wctype_l()</i>
137038	POSIX_MULTI_PROCESS: Multiple Processes
137039	<i>_Exit()</i> , <i>_Fork()</i> , <i>_exit()</i> , <i>assert()</i> , <i>at_quick_exit()</i> , <i>atexit()</i> , <i>clock()</i> , <i>execl()</i> , <i>execle()</i> , <i>execlp()</i> ,
137040	<i>execv()</i> , <i>execve()</i> , <i>execvp()</i> , <i>exit()</i> , <i>fork()</i> , <i>getpgrp()</i> , <i>getpgid()</i> , <i>getpid()</i> , <i>getppid()</i> , <i>getrlimit()</i> ,
137041	<i>getsid()</i> , <i>quick_exit()</i> , <i>setrlimit()</i> , <i>setsid()</i> , <i>sleep()</i> , <i>times()</i> , <i>wait()</i> , <i>waitid()</i> , <i>waitpid()</i>
137042	POSIX_MULTI_PROCESS_FD: Multiple Processes File Descriptor Routines
137043	<i>fexecve()</i>
137044	POSIX_NETWORKING: Networking
137045	<i>accept()</i> , <i>accept4()</i> , <i>be16toh()</i> , <i>be32toh()</i> , <i>be64toh()</i> , <i>bind()</i> , <i>connect()</i> , <i>endhostent()</i> , <i>endnetent()</i> ,
137046	<i>endprotoent()</i> , <i>endseroent()</i> , <i>freeaddrinfo()</i> , <i>gai_strerror()</i> , <i>getaddrinfo()</i> , <i>gethostent()</i> ,
137047	<i>gethostname()</i> , <i>getnameinfo()</i> , <i>getnetbyaddr()</i> , <i>getnetbyname()</i> , <i>getnetent()</i> , <i>getpeername()</i> ,
137048	<i>getprotobyname()</i> , <i>getprotobynumber()</i> , <i>getprotoent()</i> , <i>getservbyname()</i> , <i>getservbyport()</i> ,
137049	<i>getservent()</i> , <i>getsockname()</i> , <i>getsockopt()</i> , <i>htobe16()</i> , <i>htobe32()</i> , <i>htobe64()</i> , <i>htole16()</i> , <i>htole32()</i> ,
137050	<i>htole64()</i> , <i>htonl()</i> , <i>htons()</i> , <i>if_freenameindex()</i> , <i>if_indextoname()</i> , <i>if_nameindex()</i> ,
137051	<i>if_nametoindex()</i> , <i>inet_addr()</i> , <i>inet_ntoa()</i> , <i>inet_ntop()</i> , <i>inet_pton()</i> , <i>le16toh()</i> , <i>le32toh()</i> ,
137052	<i>le64toh()</i> , <i>listen()</i> , <i>ntohl()</i> , <i>ntohs()</i> , <i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , <i>sendto()</i> ,
137053	<i>sethostent()</i> , <i>setnetent()</i> , <i>setprotoent()</i> , <i>setservent()</i> , <i>setsockopt()</i> , <i>shutdown()</i> , <i>socket()</i> ,
137054	<i>socketmark()</i> , <i>socketpair()</i>
137055	POSIX_PIPE: Pipe
137056	<i>pipe()</i> , <i>pipe2()</i>
137057	POSIX_ROBUST_MUTEXES: Robust Mutexes
137058	<i>pthread_mutex_consistent()</i> , <i>pthread_mutexattr_getrobust()</i> , <i>pthread_mutexattr_setrobust()</i>
137059	POSIX_REALTIME_SIGNALS: Realtime Signals
137060	<i>sigqueue()</i> , <i>sigtimedwait()</i> , <i>sigwaitinfo()</i>

137061	POSIX_REGEX: Regular Expressions
137062	<i>regcomp()</i> , <i>regerror()</i> , <i>regexec()</i> , <i>regfree()</i>
137063	POSIX_RW_LOCKS: Reader Writer Locks
137064	<i>pthread_rwlock_clockrdlock()</i> , <i>pthread_rwlock_clockwrlock()</i> , <i>pthread_rwlock_destroy()</i> ,
137065	<i>pthread_rwlock_init()</i> , <i>pthread_rwlock_rdlock()</i> , <i>pthread_rwlock_timedrdlock()</i> ,
137066	<i>pthread_rwlock_timedwrlock()</i> , <i>pthread_rwlock_tryrdlock()</i> , <i>pthread_rwlock_trywrlock()</i> ,
137067	<i>pthread_rwlock_unlock()</i> , <i>pthread_rwlock_wrlock()</i> , <i>pthread_rwlockattr_destroy()</i> ,
137068	<i>pthread_rwlockattr_init()</i> , <i>pthread_rwlockattr_getpshared()</i> , <i>pthread_rwlockattr_setpshared()</i>
137069	POSIX_SEMAPHORES: Semaphores
137070	<i>sem_clockwait()</i> , <i>sem_close()</i> , <i>sem_destroy()</i> , <i>sem_getvalue()</i> , <i>sem_init()</i> , <i>sem_open()</i> ,
137071	<i>sem_post()</i> , <i>sem_timedwait()</i> , <i>sem_trywait()</i> , <i>sem_unlink()</i> , <i>sem_wait()</i>
137072	POSIX_SHELL_FUNC: Shell and Utilities
137073	<i>pclose()</i> , <i>popen()</i> , <i>system()</i> , <i>wordexp()</i> , <i>wordfree()</i>
137074	POSIX_SIGNAL_JUMP: Signal Jump Functions
137075	<i>siglongjmp()</i> , <i>sigsetjmp()</i>
137076	POSIX_SIGNALS: Signals
137077	<i>abort()</i> , <i>alarm()</i> , <i>kill()</i> , <i>pause()</i> , <i>raise()</i> , <i>sigaction()</i> , <i>sigaddset()</i> , <i>sigdelset()</i> , <i>sigemptyset()</i> ,
137078	<i>sigfillset()</i> , <i>sigismember()</i> , <i>signal()</i> , <i>sigpending()</i> , <i>sigprocmask()</i> , <i>sigsuspend()</i> , <i>sigwait()</i>
137079	POSIX_SIGNALS_EXT: Extended Signals
137080	<i>psignal()</i> , <i>psiginfo()</i> , <i>sig2str()</i> , <i>str2sig()</i> , <i>strsignal()</i>
137081	POSIX_SINGLE_PROCESS: Single Process
137082	<i>confstr()</i> , <i>environ</i> , <i>errno</i> , <i>getenv()</i> , <i>secure_getenv()</i> , <i>setenv()</i> , <i>sysconf()</i> , <i>uname()</i> , <i>unsetenv()</i>
137083	POSIX_SPIN_LOCKS: Spin Locks
137084	<i>pthread_spin_destroy()</i> , <i>pthread_spin_init()</i> , <i>pthread_spin_lock()</i> , <i>pthread_spin_trylock()</i> ,
137085	<i>pthread_spin_unlock()</i>
137086	POSIX_SYMBOLIC_LINKS: Symbolic Links
137087	<i>lchown()</i> , <sup>13</sup> <i>lstat()</i> , <i>readlink()</i> , <i>realpath()</i> , <i>symlink()</i>
137088	POSIX_SYMBOLIC_LINKS_FD: Symbolic Links File Descriptor Routines
137089	<i>readlinkat()</i> , <i>symlinkat()</i>
137090	POSIX_SYSTEM_DATABASE: System Database
137091	<i>getgrgid()</i> , <i>getgrnam()</i> , <i>getpwnam()</i> , <i>getpwuid()</i>
137092	POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database
137093	<i>getgrgid_r()</i> , <i>getgrnam_r()</i> , <i>getpwnam_r()</i> , <i>getpwuid_r()</i>
137094	POSIX_THREADS_BASE: Base Threads
137095	<i>pthread_atfork()</i> , <i>pthread_attr_destroy()</i> , <i>pthread_attr_getdetachstate()</i> ,
137096	<i>pthread_attr_getschedparam()</i> , <i>pthread_attr_init()</i> , <i>pthread_attr_setdetachstate()</i> ,
137097	<i>pthread_attr_setschedparam()</i> , <i>pthread_cancel()</i> , <i>pthread_cleanup_pop()</i> , <i>pthread_cleanup_push()</i> ,
137098	<i>pthread_cond_broadcast()</i> , <i>pthread_cond_clockwait()</i> , <i>pthread_cond_destroy()</i> ,
137099	<i>pthread_cond_init()</i> , <i>pthread_cond_signal()</i> , <i>pthread_cond_timedwait()</i> , <i>pthread_cond_wait()</i> ,
137100	<i>pthread_condattr_destroy()</i> , <i>pthread_condattr_init()</i> , <i>pthread_create()</i> , <i>pthread_detach()</i> ,
137101	<i>pthread_equal()</i> , <i>pthread_exit()</i> , <i>pthread_getspecific()</i> , <i>pthread_join()</i> , <i>pthread_key_create()</i> ,
137102	<i>pthread_key_delete()</i> , <i>pthread_kill()</i> , <i>pthread_mutex_clocklock()</i> , <i>pthread_mutex_destroy()</i> ,
137103	<i>pthread_mutex_init()</i> , <i>pthread_mutex_lock()</i> , <i>pthread_mutex_timedlock()</i> ,
137104	<i>pthread_mutex_trylock()</i> , <i>pthread_mutex_unlock()</i> , <i>pthread_mutexattr_destroy()</i> ,

---

137105 13. The *lchown()* function also depends on POSIX\_FILE\_ATTRIBUTES.

137106	<i>pthread_mutexattr_init()</i> , <i>pthread_once()</i> , <i>pthread_self()</i> , <i>pthread_setcancelstate()</i> ,
137107	<i>pthread_setcanceltype()</i> , <i>pthread_setspecific()</i> , <i>pthread_sigmask()</i> , <i>pthread_testcancel()</i> ,
137108	<i>sched_yield()</i>
137109	POSIX_THREADS_EXT: Extended Threads
137110	<i>pthread_attr_getguardsize()</i> , <i>pthread_attr_setguardsize()</i> , <i>pthread_mutexattr_gettype()</i> ,
137111	<i>pthread_mutexattr_settype()</i>
137112	POSIX_TIMERS: Timers
137113	<i>clock_getres()</i> , <i>clock_gettime()</i> , <i>clock_settime()</i> , <i>nanosleep()</i> , <i>timer_create()</i> , <i>timer_delete()</i> ,
137114	<i>timer_getoverrun()</i> , <i>timer_gettime()</i> , <i>timer_settime()</i>
137115	POSIX_USER_GROUPS: User and Group
137116	<i>getegid()</i> , <i>geteuid()</i> , <i>getgid()</i> , <i>getgroups()</i> , <i>getlogin()</i> , <i>getuid()</i> , <i>setegid()</i> , <i>seteuid()</i> , <i>setgid()</i> ,
137117	<i>setuid()</i>
137118	POSIX_USER_GROUPS_R: Thread-Safe User and Group
137119	<i>getlogin_r()</i>
137120	POSIX_WIDE_CHAR_DEVICE_IO: Device Input and Output
137121	<i>fgetc()</i> , <i>fgetws()</i> , <i>fputc()</i> , <i>fputws()</i> , <i>fwide()</i> , <i>fwprintf()</i> , <i>fwscanf()</i> , <i>getwc()</i> , <i>getwchar()</i> ,
137122	<i>open_wmemstream()</i> , <i>putwc()</i> , <i>putwchar()</i> , <i>ungetwc()</i> , <i>vwprintf()</i> , <i>vwscanf()</i> , <i>vwpprintf()</i> ,
137123	<i>vwscanf()</i> , <i>wprintf()</i> , <i>wscanf()</i>
137124	XSI_C_LANG_SUPPORT: XSI General C Library
137125	<i>a64l()</i> , <i>daylight</i> , <i>drand48()</i> , <i>erand48()</i> , <i>ffs()</i> , <i>ffsl()</i> , <i>ffsll()</i> , <i>getdate()</i> , <i>hcreate()</i> , <i>hdestroy()</i> ,
137126	<i>hsearch()</i> , <i>initstate()</i> , <i>insque()</i> , <i>rand48()</i> , <i>l64a()</i> , <i>lcong48()</i> , <i>lfind()</i> , <i>lrand48()</i> , <i>lsearch()</i> ,
137127	<i>memcpy()</i> , <i>mrnd48()</i> , <i>nrnd48()</i> , <i>random()</i> , <i>remque()</i> , <i>seed48()</i> , <i>setstate()</i> , <i>signgam</i> ,
137128	<i>srand48()</i> , <i>srandom()</i> , <i>strptime()</i> , <i>swab()</i> , <i>tdelete()</i> , <i>tfind()</i> , <i>timezone</i> , <i>tsearch()</i> , <i>twalk()</i>
137129	XSI_DBM: XSI Database Management
137130	<i>dbm_clearerr()</i> , <i>dbm_close()</i> , <i>dbm_delete()</i> , <i>dbm_error()</i> , <i>dbm_fetch()</i> , <i>dbm_firstkey()</i> ,
137131	<i>dbm_nextkey()</i> , <i>dbm_open()</i> , <i>dbm_store()</i>
137132	XSI_DEVICE_IO: XSI Device Input and Output
137133	<i>fntmsg()</i> , <i>readv()</i> , <i>writew()</i>
137134	XSI_DEVICE_SPECIFIC: XSI General Terminal
137135	<i>grantpt()</i> , <i>posix_openpt()</i> , <i>ptsname()</i> , <i>unlockpt()</i>
137136	XSI_FILE_SYSTEM: XSI File System
137137	<i>basename()</i> , <i>dirname()</i> , <i>lockf()</i> , <i>mknod()</i> , <i>nftw()</i> , <i>seekdir()</i> , <i>sync()</i> , <i>telldir()</i> , <i>utimes()</i>
137138	XSI_GENERAL_TERMINAL_R: XSI Thread-Safe General Terminal
137139	<i>ptsname_r()</i>
137140	XSI_IPC: XSI Interprocess Communication
137141	<i>ftok()</i> , <i>msgctl()</i> , <i>msgget()</i> , <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semctl()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmctl()</i> ,
137142	<i>shmdt()</i> , <i>shmget()</i>
137143	XSI_MATH: XSI Maths Library
137144	<i>j0()</i> , <i>j1()</i> , <i>jn()</i> , <i>y0()</i> , <i>y1()</i> , <i>yn()</i>
137145	XSI_MULTI_PROCESS: XSI Multiple Process
137146	<i>getpriority()</i> , <i>getrusage()</i> , <i>nice()</i> , <i>setpriority()</i>
137147	XSI_SIGNALS: XSI Signal
137148	<i>killpg()</i> , <i>sigaltstack()</i>

137149	XSI_SINGLE_PROCESS: XSI Single Process
137150	<i>gethostid()</i> , <i>putenv()</i>
137151	XSI_SYSTEM_DATABASE: XSI System Database
137152	<i>endgrent()</i> , <i>endpwent()</i> , <i>getgrent()</i> , <i>getpwent()</i> , <i>setgrent()</i> , <i>setpwent()</i>
137153	XSI_SYSTEM_LOGGING: XSI System Logging
137154	<i>closelog()</i> , <i>openlog()</i> , <i>setlogmask()</i> , <i>syslog()</i>
137155	XSI_USER_GROUPS: XSI User and Group
137156	<i>endutxent()</i> , <i>getresgid()</i> , <i>getresuid()</i> , <i>getutxent()</i> , <i>getutxid()</i> , <i>getutxline()</i> , <i>pututxline()</i> ,
137157	<i>setregid()</i> , <i>setresgid()</i> , <i>setresuid()</i> , <i>setreuid()</i> , <i>setutxent()</i>
137158	XSI_WIDE_CHAR: XSI Wide-Character Library
137159	<i>wcswidth()</i> , <i>wcwidth()</i>



# Index

(time) resolution .....	77
/ .....	197
/dev .....	197
/dev/console .....	197
/dev/null .....	197
/dev/tty .....	197, 3659
/etc/passwd .....	3678
/tmp .....	197
< aio.h > .....	222
< alert > .....	32
< apostrophe > .....	33
< arpa/inet.h > .....	224
< assert.h > .....	226
< backspace > .....	37
< blank > .....	37
< carriage-return > .....	39
< circumflex > .....	40
< complex.h > .....	227
< control >-V .....	2852
< control >-W .....	2852
< cpio.h > .....	230
< ctype.h > .....	232
< devctl.h > .....	234
< dirent.h > .....	235-236
< dlfcn.h > .....	238
< dollar-sign > .....	46
< endian.h > .....	240
< errno.h > .....	242
< fcntl.h > .....	246
< fenv.h > .....	252
< float.h > .....	256
< fmtmsg.h > .....	261
< fnmatch.h > .....	263
< form-feed > .....	54
< ftw.h > .....	264
< glob.h > .....	266
< grp.h > .....	268
< iconv.h > .....	270
< inttypes.h > .....	271
< iso646.h > .....	274
< langinfo.h > .....	275
< libgen.h > .....	279
< libintl.h > .....	280
< limits.h > .....	282
< locale.h > .....	297
< math.h > .....	300
< monetary.h > .....	308

<mqueue.h> .....	309
<ndbm.h> .....	311
<net/if.h> .....	313
<netdb.h> .....	314
<netinet/in.h> .....	318
<netinet/tcp.h> .....	323
<newline> .....	64
<nl_types.h> .....	324
<number-sign> .....	65
<period> .....	69
<poll.h> .....	325
<pthread.h> .....	327, 3817
<pwd.h> .....	334
<regex.h> .....	336
<sched.h> .....	339
<search.h> .....	341
<semaphore.h> .....	343
<setjmp.h> .....	345
<signal.h> .....	346
<slash> .....	81
<space> .....	82
<spawn.h> .....	356
<stdalign.h> .....	359
<stdarg.h> .....	364
<stdatomic.h> .....	360
<stdbool.h> .....	366
<stddef.h> .....	367
<stdint.h> .....	369
<stdio.h> .....	376
<stdlib.h> .....	381
<stdnoreturn.h> .....	386
<string.h> .....	387
<strings.h> .....	389
<sys/dir.h> .....	236
<sys/ipc.h> .....	390
<sys/mman.h> .....	392
<sys/msg.h> .....	396
<sys/resource.h> .....	398
<sys/select.h> .....	400
<sys/sem.h> .....	402
<sys/shm.h> .....	404
<sys/socket.h> .....	406
<sys/stat.h> .....	414
<sys/statvfs.h> .....	420
<sys/time.h> .....	422
<sys/times.h> .....	424
<sys/types.h> .....	425
<sys/uio.h> .....	429
<sys/un.h> .....	430
<sys/utsname.h> .....	432
<sys/wait.h> .....	433
<syslog.h> .....	435



<tab>	87
<tar.h>	437
<termios.h>	439
<tgmath.h>	445
<threads.h>	449
<tilde>	90
<time.h>	452
<uchar.h>	457
<unistd.h>	458
<utmpx.h>	480
<vertical-tab>	92
<wchar.h>	482
<wctype.h>	486
<wordexp.h>	488
±0	94
_asm_builtin_atoi()	3736
_BSD	3838
_CFLAGS	2676
_Complex_I	227
_CS_PATH	465
_CS_POSIX_V7_ILP32_OFF32_CFLAGS	466
_CS_POSIX_V7_ILP32_OFF32_LDFLAGS	466
_CS_POSIX_V7_ILP32_OFF32_LIBS	466
_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS	466
_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS	466
_CS_POSIX_V7_ILP32_OFFBIG_LIBS	466
_CS_POSIX_V7_LP64_OFF64_CFLAGS	466
_CS_POSIX_V7_LP64_OFF64_LDFLAGS	466
_CS_POSIX_V7_LP64_OFF64_LIBS	466
_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS	466
_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS	466
_CS_POSIX_V7_LPBIG_OFFBIG_LIBS	467
_CS_POSIX_V7_THREADS_CFLAGS	467
_CS_POSIX_V7_THREADS_LDFLAGS	467
_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS	467
_CS_POSIX_V8_ILP32_OFF32_CFLAGS	465
_CS_POSIX_V8_ILP32_OFF32_LDFLAGS	465
_CS_POSIX_V8_ILP32_OFF32_LIBS	465
_CS_POSIX_V8_ILP32_OFFBIG_CFLAGS	465
_CS_POSIX_V8_ILP32_OFFBIG_LDFLAGS	465
_CS_POSIX_V8_ILP32_OFFBIG_LIBS	465
_CS_POSIX_V8_LP64_OFF64_CFLAGS	465
_CS_POSIX_V8_LP64_OFF64_LDFLAGS	465
_CS_POSIX_V8_LP64_OFF64_LIBS	465
_CS_POSIX_V8_LPBIG_OFFBIG_CFLAGS	466
_CS_POSIX_V8_LPBIG_OFFBIG_LDFLAGS	466
_CS_POSIX_V8_LPBIG_OFFBIG_LIBS	466
_CS_POSIX_V8_THREADS_CFLAGS	466
_CS_POSIX_V8_THREADS_LDFLAGS	466
_CS_POSIX_V8_WIDTH_RESTRICTED_ENVS	466
_CS_V7_ENV	467
_CS_V8_ENV	466
_ENDIAN	501

_Exit()	568
_exit()	568, 2354, 3754, 3773
_Exit()	3924
_exit()	3924
_FILE_	626
_Fork()	574
_Imaginary_I	227
_IOFBF	376, 1976, 2021
_IOLBF	376, 958, 2021
_IONBF	376, 1976, 2021
_LDFLAGS	2676
_LIBS	2676
_LINE_	626
_longjmp()	3840
_LVL	500
_MAX	499
_MIN	282, 499
_PC constants	
defined in <unistd.h>	467
used in pathconf	988
_PC_2_SYMLINKS	988
_PC_ALLOC_SIZE_MIN	988
_PC_ASYNC_IO	988
_PC_CHOWN_RESTRICTED	988
_PC_FALLOC	988
_PC_FILESIZEBITS	988
_PC_LINK_MAX	988
_PC_MAX_CANON	988
_PC_MAX_INPUT	988
_PC_NAME_MAX	988
_PC_NO_TRUNC	988
_PC_PATH_MAX	988
_PC_PIPE_BUF	988
_PC_PRIO_IO	988
_PC_REC_INCR_XFER_SIZE	988
_PC_REC_MAX_XFER_SIZE	988
_PC_REC_MIN_XFER_SIZE	988
_PC_REC_XFER_ALIGN	988
_PC_SYMLINK_MAX	988
_PC_SYNC_IO	988
_PC_TEXTDOMAIN_MAX	988
_PC_TIMESTAMP_RESOLUTION	988
_PC_VDISABLE	988
_POSIX	282
_POSIX maximum values	
in <limits.h>	287
_POSIX minimum values	
in <limits.h>	287
_POSIX2 constants	
in sysconf	2199
_POSIX2_BC_BASE_MAX	286, 290
_POSIX2_BC_DIM_MAX	286, 290

_POSIX2_BC_SCALE_MAX .....	286, 290
_POSIX2_BC_STRING_MAX .....	287, 290
_POSIX2_CHARCLASS_NAME_MAX .....	287, 290
_POSIX2_CHAR_TERM .....	463, 2201
_POSIX2_COLL_WEIGHTS_MAX .....	287, 290
_POSIX2_C_BIND .....	17, 463, 2201
_POSIX2_C_DEV .....	463, 2201
_POSIX2_EXPR_NEST_MAX .....	287, 290
_POSIX2_FORT_RUN .....	463, 2201
_POSIX2_LINE_MAX .....	287, 290, 293
_POSIX2_LOCALEDEF .....	463, 2201
_POSIX2_RE_DUP_MAX .....	291
_POSIX2_SW_DEV .....	463, 2201
_POSIX2_SYMLINKS .....	464
_POSIX2_UPE .....	463, 2201
_POSIX2_VERSION .....	458, 2201
_POSIX .....	498
_POSIX_ADVISORY_INFO .....	18, 23, 458, 991, 2200, 3933
_POSIX_AIO_LISTIO_MAX .....	283, 287
_POSIX_AIO_MAX .....	283, 288
_POSIX_ARG_MAX .....	283, 288
_POSIX_ASYNCHRONOUS_IO .....	17, 459, 2200, 3645, 3933
_POSIX_ASYNC_IO .....	464, 988
_POSIX_BARRIERS .....	17, 459, 2200, 3645, 3933
_POSIX_CHILD_MAX .....	283, 288
_POSIX_CHOWN_RESTRICTED .....	17, 459, 726, 988, 991, 3638, 3933
_POSIX_CLOCKRES_MIN .....	287
_POSIX_CLOCK_SELECTION .....	17, 459, 2200, 3645, 3933
_POSIX_CPUTIME .....	18, 23, 459, 2200, 3933
_POSIX_C_SOURCE .....	496-497, 3737, 3741
_POSIX_DELAYTIMER_MAX .....	283, 288
_POSIX_DEVICE_CONTROL .....	18, 459, 2200
_POSIX_FALLOC .....	464, 988
_POSIX_FSYNC .....	18-19, 22-23, 459, 2200, 3933
_POSIX_HOST_NAME_MAX .....	283, 288
_POSIX_IPV6 .....	18, 459, 2200, 3934
_POSIX_JOB_CONTROL .....	17, 459, 2200, 3638, 3934, 3937
_POSIX_LINK_MAX .....	285, 288
_POSIX_LOGIN_NAME_MAX .....	283, 288
_POSIX_MAPPED_FILES .....	17, 459, 2200, 3645, 3934
_POSIX_MAX_CANON .....	285, 288
_POSIX_MAX_INPUT .....	285, 288
_POSIX_MEMLOCK .....	18, 22-23, 459, 2200, 3934
_POSIX_MEMLOCK_RANGE .....	18, 22-23, 459, 2200, 3934
_POSIX_MEMORY_PROTECTION .....	17, 459, 2200, 3645, 3934
_POSIX_MESSAGE_PASSING .....	18, 22-23, 460, 2200, 3934
_POSIX_MONOTONIC_CLOCK .....	17, 460, 2200, 3645, 3935
_POSIX_MQ_OPEN_MAX .....	283, 288
_POSIX_MQ_PRIO_MAX .....	283, 288
_POSIX_NAME_MAX .....	285-286, 288, 1457, 1468, 1628, 1944, 1952, 2028
_POSIX_NGROUPS_MAX .....	287-288
_POSIX_NO_TRUNC .....	17, 105, 460, 988, 3638

_POSIX_OPEN_MAX .....	283, 288, 1178
_POSIX_PATH_MAX .....	285, 289, 430, 1457, 1468, 1944, 1952, 2028
_POSIX_PIPE_BUF .....	286, 289
_POSIX_PRIORITIZED_IO .....	18, 22-23, 460, 528-529, 2200, 3934
_POSIX_PRIORITY_SCHEDULING .....	18, 22-23, 460, 528, 2200, 3935
_POSIX_PRIO_IO .....	464, 988
_POSIX_RAW_SOCKETS .....	18, 460, 2200
_POSIX_READER_WRITER_LOCKS .....	17, 460, 2200, 3645
_POSIX_REALTIME_SIGNALS .....	17, 460, 2200, 3645, 3935
_POSIX_REGEX .....	17, 460, 2200, 3935
_POSIX_RE_DUP_MAX .....	287, 289
_POSIX_RTSIG_MAX .....	284, 289, 3748, 3939
_POSIX_SAVED_IDS .....	17, 460, 2200, 3638, 3935
_POSIX_SEMAPHORES .....	17, 460, 2200, 3645, 3935
_POSIX_SEM_NSEMS_MAX .....	284, 289
_POSIX_SEM_VALUE_MAX .....	284, 289
_POSIX_SHARED_MEMORY_OBJECTS .....	18, 22-23, 460, 2200, 3935
_POSIX_SHELL .....	17, 460, 2200, 3935
_POSIX_SIGQUEUE_MAX .....	284, 289
_POSIX_SOURCE .....	497, 3737
_POSIX_SPAWN .....	18, 23, 461, 2200, 3935
_POSIX_SPINLOCKS .....	3935
_POSIX_SPIN_LOCKS .....	17, 461, 2200, 3645
_POSIX_SPORADIC_SERVER .....	18, 23, 461, 2200, 3935
_POSIX_SSIZE_MAX .....	289, 292
_POSIX_SS_REPL_MAX .....	284, 289, 2200, 3789
_POSIX_STREAM_MAX .....	284, 289
_POSIX_SYMLINK_MAX .....	286, 289
_POSIX_SYMLOOP_MAX .....	284, 289
_POSIX_SYNCHRONIZED_IO .....	18, 22-23, 461, 2200, 3936
_POSIX_SYNC_IO .....	464, 988, 3933
_POSIX_THREADS .....	17, 462, 2201, 3645, 3936
_POSIX_THREAD_ATTR_STACKADDR .....	18-19, 461, 2200, 3936
_POSIX_THREAD_ATTR_STACKSIZE .....	18-19, 461, 2200, 3936
_POSIX_THREAD_CPUTIME .....	18, 24, 461, 2200
_POSIX_THREAD_DESTRUCTOR_ITERATIONS .....	284, 289
_POSIX_THREAD_KEYS_MAX .....	284, 290
_POSIX_THREAD_PRIORITY_SCHEDULING .....	18, 24, 461, 2200, 3936
_POSIX_THREAD_PRIO_INHERIT .....	18, 24, 461, 2200, 3936
_POSIX_THREAD_PRIO_PROTECT .....	18, 24, 461, 2200, 3936
_POSIX_THREAD_PROCESS_SHARED .....	18-19, 461, 1764, 2200, 3936
_POSIX_THREAD_ROBUST_PRIO_INHERIT .....	24, 461, 2201
_POSIX_THREAD_ROBUST_PRIO_PROTECT .....	24, 462, 2201
_POSIX_THREAD_SAFE_FUNCTIONS .....	17, 462, 2201, 3645, 3936
_POSIX_THREAD_SPORADIC_SERVER .....	18, 24, 462, 2201, 3937
_POSIX_THREAD_THREADS_MAX .....	284, 290
_POSIX_TIMEOUTS .....	17, 462, 2201, 3645, 3937
_POSIX_TIMERS .....	17, 462, 2201, 3645, 3937
_POSIX_TIMER_MAX .....	285, 290
_POSIX_TIMESTAMP_RESOLUTION .....	464, 988
_POSIX_TTY_NAME_MAX .....	285, 290
_POSIX_TYPED_MEMORY_OBJECTS .....	18, 23, 462, 2201, 3937

_POSIX_TZNAME_MAX.....	285, 290, 3708
_POSIX_V7_ILP32_OFF32 .....	462, 2201
_POSIX_V7_ILP32_OFFBIG .....	462, 2201
_POSIX_V7_LP64_OFF64.....	462, 2201
_POSIX_V7_LPBIG_OFFBIG.....	462, 2201
_POSIX_V8_ILP32_OFF32 .....	462, 2201
_POSIX_V8_ILP32_OFFBIG .....	462, 2201
_POSIX_V8_LP64_OFF64.....	462, 2201
_POSIX_V8_LPBIG_OFFBIG.....	463, 2201
_POSIX_VDISABLE .....	18, 470, 988, 3413, 3638
_POSIX_VERSION .....	17, 458, 2201, 2314
_PROCESS.....	500
_PTHREAD_THREADS_MAX .....	1731
_SC constants	
defined in <unistd.h> .....	<b>468</b>
in sysconf.....	2199
_SC_2_CHAR_TERM .....	2201
_SC_2_C_BIND.....	2201
_SC_2_C_DEV .....	2201
_SC_2_FORT_RUN .....	2201
_SC_2_LOCALEDEF .....	2201
_SC_2_SW_DEV .....	2201
_SC_2_UPE.....	2201
_SC_2_VERSION.....	1549, 2201
_SC_ADVISORY_INFO.....	2200
_SC_AIO_LISTIO_MAX.....	2199
_SC_AIO_MAX .....	2199
_SC_AIO_PRIO_DELTA_MAX.....	2199
_SC_ARG_MAX .....	2199
_SC_ASYNCHRONOUS_IO .....	2200
_SC_ATEXIT_MAX.....	2199
_SC_BARRIERS .....	2200
_SC_BC_BASE_MAX.....	2199
_SC_BC_DIM_MAX.....	2199
_SC_BC_SCALE_MAX.....	2199
_SC_BC_STRING_MAX.....	2199
_SC_CHILD_MAX .....	2199
_SC_CLK_TCK .....	2199, 2276
_SC_CLOCK_SELECTION .....	2200
_SC_COLL_WEIGHTS_MAX .....	2199
_SC_CPUTIME .....	2200
_SC_DELAYTIMER_MAX .....	2199
_SC_DEVICE_CONTROL.....	2200
_SC_EXPR_NEST_MAX.....	2199
_SC_FSYNC .....	2200
_SC_GETGR_R_SIZE_MAX .....	1128, 2199
_SC_GETPW_R_SIZE_MAX .....	2199
_SC_IOV_MAX.....	2199
_SC_IPV6.....	2200
_SC_JOB_CONTROL.....	2200
_SC_LINE_MAX.....	2199
_SC_LOGIN_NAME_MAX .....	2199

_SC_MEMLOCK .....	2200
_SC_MEMLOCK_RANGE.....	2200
_SC_MEMORY_PROTECTION .....	2200
_SC_MESSAGE_PASSING.....	2200
_SC_MONOTONIC_CLOCK.....	2200
_SC_MQ_OPEN_MAX.....	2199
_SC_MQ_PRIO_MAX.....	2199
_SC_NGROUPS_MAX .....	2199
_SC_NPROCESSORS_CONF .....	2199
_SC_NPROCESSORS_ONLN .....	2199
_SC_NSIG.....	2199
_SC_OPEN_MAX.....	2199
_SC_PAGESIZE .....	1572, 2199, 3773-3774
_SC_PAGE_SIZE .....	2199
_SC_PRIORITIZED_IO.....	2200
_SC_PRIORITY_SCHEDULING.....	2200
_SC_RAW_SOCKETS .....	2200
_SC_READER_WRITER_LOCKS .....	2200
_SC_REALTIME_SIGNALS.....	2200
_SC_REGEX .....	2200
_SC_RE_DUP_MAX .....	2200
_SC_RTSIG_MAX.....	2200
_SC_SAVED_IDS.....	2200
_SC_SEMAPHORES .....	2200
_SC_SEM_NSEMS_MAX.....	2200
_SC_SEM_VALUE_MAX .....	2200
_SC_SHARED_MEMORY_OBJECTS.....	2200
_SC_SHELL.....	2200
_SC_SIGQUEUE_MAX .....	2200
_SC_SPAWN .....	2200
_SC_SPIN_LOCKS .....	2200
_SC_SPORADIC_SERVER.....	2200
_SC_SS_REPL_MAX.....	2200
_SC_STREAM_MAX.....	2200
_SC_SYMLOOP_MAX .....	2200
_SC_SYNCHRONIZED_IO .....	2200
_SC_THREADS .....	2201
_SC_THREAD_ATTR_STACKADDR.....	2200
_SC_THREAD_ATTR_STACKSIZE .....	2200
_SC_THREAD_CPUTIME .....	2200
_SC_THREAD_DESTRUCTOR_ITERATIONS.....	2199
_SC_THREAD_KEYS_MAX.....	2200
_SC_THREAD_PRIORITY_SCHEDULING.....	2200
_SC_THREAD_PRIO_INHERIT .....	2200
_SC_THREAD_PRIO_PROTECT.....	2200
_SC_THREAD_PROCESS_SHARED .....	2200
_SC_THREAD_ROBUST_PRIO_INHERIT .....	2201
_SC_THREAD_ROBUST_PRIO_PROTECT.....	2201
_SC_THREAD_SAFE_FUNCTIONS.....	2201
_SC_THREAD_SPORADIC_SERVER.....	2201
_SC_THREAD_STACK_MIN.....	2200
_SC_THREAD_THREADS_MAX.....	2200

_SC_TIMEOUTS.....	2201
_SC_TIMERS.....	2201
_SC_TIMER_MAX .....	2200
_SC_TTY_NAME_MAX.....	2200
_SC_TYPED_MEMORY_OBJECTS .....	2201
_SC_TZNAME_MAX .....	2200
_SC_V7_ILP32_OFF32.....	2201
_SC_V7_ILP32_OFFBIG .....	2201
_SC_V7_LP64_OFF64 .....	2201
_SC_V7_LPBIG_OFFBIG.....	2201
_SC_V8_ILP32_OFF32.....	2201
_SC_V8_ILP32_OFFBIG .....	2201
_SC_V8_LP64_OFF64 .....	2201
_SC_V8_LPBIG_OFFBIG.....	2201
_SC_VERSION.....	2201
_SC_XOPEN_CRYPT.....	2201
_SC_XOPEN_ENH_I18N.....	2201
_SC_XOPEN_REALTIME .....	2201
_SC_XOPEN_REALTIME_THREADS .....	2201
_SC_XOPEN_SHM .....	2201
_SC_XOPEN_UNIX .....	2201
_SC_XOPEN_UUCP .....	2201
_SC_XOPEN_VERSION.....	2201
_setjmp() .....	3840
_t.....	500
_TIME.....	500
_tolower().....	3840
_toupper().....	3840
_XOPEN_CRYPT.....	18, 22, 463, 2201
_XOPEN_ENH_I18N.....	463, 2201
_XOPEN_IOV_MAX.....	283, 291
_XOPEN_NAME_MAX .....	285-286, 291, 1457, 1468, 1628, 1944, 1952, 2028
_XOPEN_PATH_MAX .....	286, 291, 1457, 1468, 1628, 1944, 1952, 2028
_XOPEN_REALTIME .....	18, 22, 463, 913, 2201
_XOPEN_REALTIME_THREADS .....	18, 24, 463, 2201
_XOPEN_SHM .....	463, 2201
_XOPEN_SOURCE.....	497-498, 3738
_XOPEN_UNIX.....	18-19, 463, 2201
_XOPEN_UUCP .....	464, 2201
_XOPEN_VERSION.....	19, 458, 2201
__alignas_is_defined.....	359
__alignof_is_defined.....	359
__errno().....	3745
__STDC_WANT_LIB_EXT1__ .....	497, 3738
a64l() .....	575
ABDAY_.....	277
ABDAY_1.....	1510
ABMON_.....	277
abort() .....	577, 3924
abortive release.....	31
abort_handler_s.....	502
abs() .....	579

absolute pathname.....	31, 105
accept().....	<b>580</b>
accept4().....	580
access mode.....	31
access().....	<b>584</b> , 3677, 3924
acos().....	<b>588</b>
acosf().....	588
acosh().....	<b>590</b>
acoshf().....	590
acoshl().....	590
acosl().....	588, 592
ACTION.....	1215
actions equivalent to functions.....	2457
adb	
rationale for omission.....	3913
additional file access control mechanism.....	31
address families.....	3835
address information.....	1022
address space.....	31
address string.....	1022
addressing.....	3835
addrinfo.....	<b>315</b> , 1022
admin.....	<b>2574</b>
ADV.....	<b>7</b>
advanced realtime.....	23
ADVANCED REALTIME.....	356, 734, 1563, 1565, 1572, 1574, 1576, 1581, 1590, 1592, 1596, 1600 1602, 1604, 1606, 1608, 1610, 1612, 1614, 1624, 1626
advanced realtime threads.....	24
ADVANCED REALTIME THREADS.....	1725
advisory information.....	31, 3759
affirmative response.....	32
AF.....	500
AF_INET.....	410
AF_INET6.....	410
AF_UNIX.....	410
AF_UNSPEC.....	410, 768
AIO.....	499
aio.....	499
AIO_ALLDONE.....	222, 593
aio_cancel().....	<b>593</b> , 3768
AIO_CANCELED.....	222, 593
aio_error().....	<b>595</b>
aio_fsync().....	<b>597</b> , 3751, 3767
AIO_LISTIO_MAX.....	283, 1338, 2199, 3937
AIO_MAX.....	283, 1338, 2199, 3937
AIO_NOTCANCELED.....	222, 593
AIO_PRIO_DELTA_MAX.....	283, 528, 2199, 3937
aio_read().....	<b>600</b> , 3768
aio_return().....	<b>603</b>
aio_suspend().....	<b>605</b> , 3767, 3794
aio_write().....	<b>607</b> , 3768
ai.....	499



AI_ADDRCONFIG .....	315, 1023
AI_ALL .....	315, 1023
AI_CANONNAME .....	315, 1023
AI_INET6 .....	1023
AI_NUMERICHOST .....	315, 1023
AI_NUMERICSERV .....	315, 1023
AI_PASSIVE .....	315, 1023
AI_V4MAPPED .....	315, 1023
alarm() .....	<b>610</b> , 3757, 3793, 3924
alert .....	32
alert character .....	32
alias .....	2470, 2580, 3865, 3928
alias name .....	32
alias substitution .....	2477, 3872
alignas .....	359
aligned_alloc() .....	<b>612</b>
alignment .....	32
alignof .....	359
alphasort() .....	<b>614</b>
alternate file access control mechanism .....	32
alternate signal stack .....	33
ALT_DIGITS .....	277
AM_STR .....	277
anchoring .....	186
ancillary data .....	33
AND list .....	2507, 3901
AND-OR list .....	2505
angle brackets .....	33
anycast .....	559
API .....	33
apostrophe character .....	33
appending redirected output .....	2495
application .....	33
application address .....	33
application conformance .....	27
application program interface .....	33
application-managed thread stack .....	548, 3835
appropriate privileges .....	34, 586, 989, 3649
ar .....	<b>2584</b> , 3930-3931
arbitrary file size .....	3864
archives	
ar command .....	2584
AREGTYPE .....	437
argc .....	874
argument .....	34
ARG_MAX .....	283, 507, 867, 871, 876, 2199, 3605, 3661, 3858, 3937
arithmetic expansion .....	2490, 3886
arithmetic language	
bc .....	2651
arithmetic precision and operations .....	2457
arm (a timer) .....	34
array identifiers .....	2656

as	
rationale for omission.....	3913
asa.....	<b>2592</b> , 3928, 3930, 3932
ASCII.....	3663
asctime().....	<b>617</b>
asctime_s.....	502
asin().....	<b>620</b>
asinf().....	620
asinh().....	<b>622</b>
asinhf().....	622
asinhf_l().....	622
asinl().....	620, 624
asprintf().....	<b>625</b> , 995
assert().....	<b>626</b>
asterisk.....	34
async-cancel safety.....	3832
async-cancel-safe function.....	34
async-signal-safe.....	1643, 3754
async-signal-safe function.....	35
asynchronous AND-OR list.....	2519
asynchronous AND-OR lists.....	2506, 3901
asynchronous error.....	3837
asynchronous events.....	34
asynchronous I/O.....	3767, 3925
completion.....	35
operation.....	35
asynchronous input and output.....	34
asynchronously-generated signal.....	35
at.....	<b>2595</b> , 3928
at-job.....	2595
atan().....	<b>629</b>
atan2().....	<b>631</b>
atan2f().....	631
atan2l().....	631
atanf().....	629, 634
atanh().....	<b>635</b>
atanhf().....	635
atanhl().....	635
atanl().....	629, 637
atexit().....	<b>638</b> , 3832
ATEXIT_MAX.....	283, 627, 638, 2199
atof().....	<b>640</b>
atoi().....	<b>641</b> , 3735-3736
atol().....	<b>643</b>
atoll().....	643
atomic operation.....	35
atomic_bool.....	647
ATOMIC_BOOL_LOCK_FREE.....	361
ATOMIC_CHAR16_T_LOCK_FREE.....	362
ATOMIC_CHAR32_T_LOCK_FREE.....	362
ATOMIC_CHAR_LOCK_FREE.....	361
atomic_compare_exchange_strong().....	<b>644</b>

atomic_compare_exchange_strong_explicit()	644
atomic_compare_exchange_weak()	644
atomic_compare_exchange_weak_explicit ()	644
atomic_exchange()	<b>646</b>
atomic_exchange_explicit()	646
atomic_fetch_add()	<b>647</b>
atomic_fetch_add_explicit()	647
atomic_fetch_and()	647
atomic_fetch_and_explicit()	647
atomic_fetch_or()	647
atomic_fetch_or_explicit()	647
atomic_fetch_sub()	647
atomic_fetch_sub_explicit()	647
atomic_fetch_xor()	647
atomic_fetch_xor_explicit()	647
atomic_flag	360
atomic_flag_clear()	<b>649</b>
atomic_flag_clear_explicit()	649
atomic_flag_test_and_set()	<b>650</b>
atomic_flag_test_and_set_explicit()	650
atomic_init()	<b>651</b>
ATOMIC_INT_LOCK_FREE	362
atomic_is_lock_free()	<b>652</b>
ATOMIC_LLONG_LOCK_FREE	362
atomic_load()	<b>653</b>
atomic_load_explicit()	653
ATOMIC_LONG_LOCK_FREE	362
ATOMIC_POINTER_LOCK_FREE	362
ATOMIC_SHORT_LOCK_FREE	362
atomic_signal_fence()	<b>654</b>
atomic_store()	<b>656</b>
atomic_store_explicit()	656
atomic_thread_fence()	654
ATOMIC_WCHAR_T_LOCK_FREE	362
AT_EACCESS	248
AT_FDCWD	248, 584, 721, 727, 1056, 1075, 1332, 1410, 1417, 1422, 1518, 1864, 1900, 2194 3450
at_quick_exit()	<b>627</b>
AT_REMOVEDIR	249, 2320
AT_SYMLINK_FOLLOW	248, 1332
AT_SYMLINK_NOFOLLOW	248, 721, 727, 1056, 1076
authentication	35
authorization	36
automatic storage class	2660
awk	<b>2605</b> , 3928, 3930
actions	2618
arithmetic functions	2621
escape sequences	2616
expression patterns	2618
expressions	2608
functions	2620

grammar .....	2624
input/output and general functions .....	2623
lexical conventions .....	2631
output statements .....	2619
overall program structure .....	2608
pattern ranges .....	2618
patterns .....	2617
regular expressions .....	2615
special patterns .....	2617
string functions .....	2621
user-defined functions .....	2624
variables and special variables .....	2612
background .....	1997, 2518, 3656-3659, 3720-3721
background job .....	36, 2519
background process .....	36, 2236
background process group .....	36
background work	
at .....	2595
batch .....	2647
bg .....	2666
crontab .....	2749
fg .....	2928
jobs .....	3021
nice .....	3209
nohup .....	3222
renice .....	3330
backquote .....	36
BACKREF .....	191
backslash .....	36, 3867
backspace character .....	37
banner	
rationale for omission .....	3913
barrier .....	37, 3809
basename .....	37, 2644, 3927, 3929
basename() .....	<b>657</b>
basic regular expression .....	37, 181, 3711
batch .....	<b>2647</b> , 3928
baud rate functions .....	714
bc .....	<b>2651</b> , 3927-3928, 3932
grammar .....	2652
lexical conventions .....	2654
operations .....	2656
operators .....	2656
bcc (mailer blind carbon copy) .....	3126
BC_ constants	
in sysconf .....	2199
BC_BASE_MAX .....	286, 2199, 2460, 3858
BC_DIM_MAX .....	286, 2199, 2460, 3858
BC_SCALE_MAX .....	286, 2199, 2460, 3858
BC_STRING_MAX .....	287, 2199, 2460, 2654
be16toh() .....	<b>660</b>
be32toh() .....	660

be64toh()	660
bg	2470, 2666, 3865, 3928
BIG_ENDIAN	240
binary primaries	3433
bind	37
bind()	662
bindtextdomain()	666
bind_textdomain_codeset()	666
blank character	37
blank line	37
blkcnt_t	425
blksize_t	425
BLKTYPE	437
block special file	38
block-mode terminal	38
blocked process (or thread)	37
blocking	38
BOOT_TIME	480, 853-854
bounded response	3926
braces	38
bracket expression	
grammar	3717
brackets	38
BRE	
expression anchoring	3714
grammar lexical conventions	3716
matching a collating element	3711
matching a single character	3711
matching multiple characters	3713
ordinary character	3711
periods	3711
precedence	3714
special character	3711
BRE (ERE) matching a single character	180
BRE (ERE) matching multiple characters	180
break	2527
BRKINT	440
broadcast	38
BSD	3652, 3720, 3746
BSDLY	441
bsearch()	670
bsearch_s	502
BSn	441
btowc()	673
buffer cache	1064
BUFSIZ	376, 1976
built-in	39
built-in utilities	39, 2470, 3864
builtin	2732
BUS_	499
BUS_ADRALN	351
BUS_ADRERR	351

BUS_OBJERR .....	351
byte .....	39
byte input/output functions .....	39
byte-oriented stream .....	524
BYTE_ORDER .....	240
C Shell .....	3656-3657
C-language extensions .....	3921, 3927
c16rtomb() .....	<b>674</b>
c17 .....	<b>2669</b> , 3930
external symbols .....	2675
standard libraries .....	2674
c32rtomb() .....	674
cabs() .....	<b>676</b>
cabsf() .....	676
cabsl() .....	676
cacos() .....	<b>677</b>
cacosf() .....	677
cacosh() .....	<b>679</b>
cacoshf() .....	679
cacoshl() .....	679
cacosl() .....	677, 681
cal .....	<b>2683</b>
calendar	
rationale for omission .....	3913
calloc() .....	<b>684</b>
call_once() .....	<b>682</b>
can .....	5
cancel	
rationale for omission .....	3913
cancel-safe .....	1813
cancelability state .....	542, 1732, 1813
cancelability type .....	1732, 1813
canceling execution of a thread .....	1686
cancellation cleanup handler .....	1691, 1699, 1723, 1735, 3830-3831
cancellation cleanup stack .....	3830
cancellation points .....	543
canonical mode input processing .....	202, 3721
canonical name .....	1023
carg() .....	<b>686</b>
cargf() .....	686
cargl() .....	686
carriage-control characters .....	2592
carriage-return character .....	39
case .....	3902
case conditional construct .....	2509
case folding .....	3678-3679
case insensitive comparisons .....	95, 3676
casin() .....	<b>688</b>
casinf() .....	688
casinh() .....	<b>690</b>
casinhf() .....	690
casinhl() .....	690

casinl()	688, 692
cat	2686, 3864
catan()	693
catanf()	693
catanh()	695
catanhf()	695
catanhl()	695
catanl()	693, 697
catclose()	698, 3929
catgets()	699, 3929
catopen()	701, 3929
CBAUD	501
cbrt()	704
cbrtf()	704
cbrtl()	704
cc (mailer carbon copy)	3126
ccos()	705
ccosf()	705
ccosh()	707
ccoshf()	707
ccoshl()	707
ccosl()	705, 709
CD	7
cd	2470, 2690, 3865, 3929
ceil()	710
ceilf()	710
ceill()	710
CEO	3712
cexp()	712
cexpf()	712
cexpl()	712
cfgetispeed()	714
cfgetospeed()	716
cflow	2697
cfsetispeed()	717
cfsetospeed()	718
change current working directory	720, 2457
change file modes	724
change history	3639, 3731, 3855
change owner and group of file	728
char	563, 3839
char16_t	457
char32_t	457
character	39, 3650
rationale	3650
character array	40
character class	40
character counting	3586
character encoding	120, 3689
state-dependent	125
character set	40, 3688
description file	3690

portable filename.....	3663
character special file.....	40
character string.....	40
CHARCLASS_NAME_MAX.....	287, 3695
charmap	
description.....	121
with localedef.....	3062
writing names with locale.....	3056
charmap file .....	3060, 3413
CHAR_BIT .....	291
CHAR_MAX .....	291, 1349, 1351, 3697
CHAR_MIN .....	291
chdir().....	<b>719</b>
chgrp.....	<b>2701</b> , 3864, 3929-3930
child process.....	40, 3650
CHILD_MAX.....	283, 984, 2199, 3639, 3839, 3858, 3901, 3937
chmod.....	<b>2704</b> , 3864, 3929-3930
grammar .....	2707
chmod().....	<b>721</b> , 3924
chown .....	<b>2711</b> , 3864, 3929-3930
chown().....	<b>726</b> , 3924
chroot	
rationale for omission.....	3913
chroot().....	3665
CHRTYPE.....	437
cimag().....	<b>730</b>
cimagf().....	730
cimagl().....	730
circumflex.....	40
cksum.....	2715, 3864, 3929
CLD_.....	499
CLD_CONTINUED.....	351
CLD_DUMPED.....	351
CLD_EXITED.....	351
CLD_KILLED.....	351
CLD_STOPPED.....	351
CLD_TRAPPED.....	351
clearerr().....	<b>731</b>
CLOCAL.....	442
clock .....	41, 3789
clock jump.....	41
clock tick.....	41, 610, 2202, 2276, 3650
per second .....	2199
rationale.....	3650
clock().....	<b>732</b>
clockid_t.....	425
CLOCKRES_MIN.....	3937
clocks.....	3789
CLOCKS_PER_SEC .....	425, 453, 732
CLOCK_.....	500
clock_.....	500
clock_getcpuclockid().....	<b>734</b> , 3796, 3798



clock_getres()	735
clock_gettime()	735
CLOCK_MONOTONIC	287, 453, 536, 740, 2268, 3794
clock_nanosleep()	739, 3794
CLOCK_PROCESS_CPUTIME_ID	453, 537, 3796, 3798
CLOCK_REALTIME	287, 453, 536, 735, 740, 1494, 1741, 2268, 3789-3794
clock_settime()	735, 742
clock_t	425
CLOCK_THREAD_CPUTIME_ID	453, 537, 3796, 3798
clog()	743
clogf()	743
clogl()	743
close a file	747
close()	745, 3773, 3924
closedir()	751, 3925
closelog()	753, 3930
cmp	2720, 3864, 3928
CMPLX()	566
CMPLXF()	566
CMPLXL()	566
msg_	500
MSG_	501
MSG_DATA	407
MSG_FIRSTHDR	407
MSG_LEN	408
MSG_NXTHDR	407
MSG_SPACE	408
nd_broadcast()	757
nd_destroy()	759
nd_init()	759
nd_signal()	757
nd_timedwait()	761
nd_wait()	761
code block	41
coded character set	41
codes	3642
codeset	41
CODESET	277
codeset conversion	3003
tr	3459
col	
rationale for omission	3913
collating element	41
collating element order	3712
collation	42
collation sequence	42
COLL_ELEM_MULTI	191
COLL_ELEM_SINGLE	191
COLL_WEIGHTS_MAX	287, 2199, 2460, 3858
colon	2530
column position	42, 3651
COLUMNS	174, 3706

comm.....	2724, 3928
command.....	42, 2470, 2728, 3650, 3865, 3927
command execution.....	2502, 3898
command interpreter	
portable.....	2353
command language.....	3921, 3927
command language interpreter.....	42
command mode.....	2815
command search.....	2502, 3898
command substitution.....	2489, 3884
commands with no command name.....	2501, 3897
communications commands	
mailx.....	3102
talk.....	3424
uucp.....	3504
uudecode.....	3508
uuencode.....	3511
uustat.....	3516
uux.....	3519
write.....	3597
compare thread IDs.....	1722
compilation environment.....	496, 3737
compilers	
c17.....	2669
yacc.....	3613
complex.....	227
complex data manipulation.....	3922, 3928
composite graphic symbol.....	43
compound commands.....	2508, 3901
compound-list.....	2505
compress.....	2735
compression	
compress.....	2735
uncompress.....	2735
zcat.....	2735
compression algorithms.....	2737
concepts.....	3642
concurrent execution.....	95, 3676
of processes.....	2453
condition variable.....	43
conditional construct	
case.....	3902
if.....	3903
configurable limits.....	3932, 3937
configuration interrogation.....	3920, 3923
configuration options.....	3931
shell and utilities.....	3931
system interfaces.....	3933
configuration values.....	2973
conformance.....	15, 27, 3639, 3643, 3647, 3649, 3680, 3837
POSIX.....	15
POSIX system interfaces.....	17

XSI .....	15
XSI system interfaces .....	19
conformance document .....	16, 3639
rationale .....	3639
conforming application .....	16, 2086, 2589, 3647, 3747, 3861, 3863
conforming application, strictly .....	610, 874, 3643, 3647, 3753
conforming implementation options .....	20
confstr() .....	<b>763</b> , 3925
conj() .....	<b>767</b>
conjf() .....	767
conjl() .....	767
connect() .....	<b>768</b>
connected socket .....	43
connection .....	43
connection indication queue .....	3837
connection mode .....	43
connectionless mode .....	43
consequences of shell errors .....	2497
continue .....	<b>2532</b>
control character .....	43, 3409
control mode .....	3724
control operator .....	44, 3651
controlling process .....	44, 2518
controlling terminal .....	44, 200, 2453, 2518, 3651, 3720, 3924
CONTTYPE .....	437
conversion descriptor .....	44, 868, 873, 1222-1223, 1225-1226
conversion specification .....	995, 1037, 1081, 1091, 2129
modified .....	2137
conversion specifier	
modified .....	2160
Coordinated Universal Time (UTC) .....	2773
copy .....	147
copy files commands	
cp .....	2741
dd .....	2776
ln .....	3051
mv .....	3198
pax .....	3250
copysign() .....	<b>772</b>
copysignf() .....	772
copysignl() .....	772
core .....	3678
core file .....	570
core image .....	44, 3651
cos() .....	<b>773</b>
cosf() .....	773
cosh() .....	<b>775</b>
coshf() .....	775
coshl() .....	775
cosl() .....	773, 777
covert channel .....	1313, 3680
cp .....	<b>2741</b> , 3864, 3929

cpio	
rationale for omission.....	3913
cpio format.....	3271
cpow().....	778
cpowf().....	778
cpowl().....	778
cpp	
rationale for omission.....	3913
cproj().....	779
cprojf().....	779
cprojl().....	779
CPT.....	7
CPU.....	424
CPU time.....	44, 3443, 3652
clock.....	44
timer.....	45
CRDLY.....	440
CREAD.....	442
creal().....	780
crealf().....	780
creall().....	780
creat().....	781, 3773, 3864
create a per-process timer.....	2269
create an interprocess channel.....	1540
create session and set process group ID.....	2012
CRn.....	440
CRNCYSTR.....	277
cron daemon.....	2752
crontab.....	2749, 3928
CRYPT.....	783, 838, 1988
crypt().....	783
csin().....	785
csinf().....	785
csinh().....	787
csinhf().....	787
csinhl().....	787
csinl().....	785, 789
CSIZE.....	442, 3724
CSn.....	442
csplit.....	2753, 3928
csqrt().....	790
csqrtf().....	790
csqrtl().....	790
CSTOPB.....	442
CS_POSIX_V8_THREADS_LDFLAGS.....	466
ctags.....	2757, 3930
ctan().....	792
ctanf().....	792
ctanh().....	794
ctanhf().....	794
ctanhl().....	794
ctanl().....	792, 796

ctermid()	797
ctime()	799, 3927
ctime_s	502
cu	
rationale for omission	3913
currency_symbol	148
current job	45
current working directory	45, 93, 2453
cursor position	45
cut	2762, 3928
CX	7
cxref	2767
c_	500
C_ constants in <cpio.h>	230
C_IRGRP	230
C_IROTH	230
C_IRUSR	230
C_ISBLK	230
C_ISCHR	230
C_ISCTG	230
C_ISDIR	230
C_ISFIFO	230
C_ISGID	230
C_ISLNK	230
C_ISREG	230
C_ISSOCK	230
C_ISUID	230
C_ISVTX	230
C_IWGRP	230
C_IWOTH	230
C_IWUSR	230
C_IXGRP	230
C_IXOTH	230
C_IXUSR	230
data access	3920, 3924
data key creation	1736
data keywords	3305
data race	45, 100
data segment	45
data structure	
dirent	235
entry	341
group	268
lconv	297
msqid_ds	396
posix_dent	235
stat	414
data type	561, 3837
ACTION	341
cc_t	439
DIR	235
div_t	381

ENTRY .....	341
FILE .....	376
fpos_t.....	376
glob_t .....	266
ldiv_t .....	381
lldiv_t.....	381
max_align_t.....	367
mbstate_t .....	482
msglen_t .....	396
msgqnum_t .....	396
nl_catd.....	324
nl_item .....	324
pid_t .....	346
ptrdiff_t.....	367
regex_t.....	336
regmatch_t.....	336
regoff_t.....	336
shmatt_t .....	404
sigset_t .....	346
sig_atomic_t .....	346
size_t .....	367
speed_t.....	439
tflag_t.....	439
VISIT .....	341
wchar_t .....	367
wctrans_t .....	486
wint_t.....	482
data types	
defined in <fenv.h> .....	<b>252</b>
defined in <sys/types.h> .....	425
date.....	<b>2770</b> , 3929
DATEMSK .....	<b>174</b> , 1109
datum .....	<b>311</b>
daylight.....	<b>801</b> , 2310
DAY_ .....	277
DBL_ constants	
defined in <float.h>.....	<b>257</b>
DBL_DECIMAL_DIG .....	258
DBL_DIG .....	258
DBL_EPSILON .....	259
DBL_MANT_DIG .....	257
DBL_MAX .....	259
DBL_MAX_10_EXP.....	259
DBL_MAX_EXP.....	259
DBL_MIN .....	260, 620, 622, 629, 631, 635, 858, 882, 884, 886, 915, 969
DBL_MIN_10_EXP.....	259
DBL_MIN_EXP.....	258
DBL_TRUE_MIN .....	260
DBM .....	311, 802, 804
DBM_ .....	499
dbm_ .....	499
dbm_clearerr().....	<b>802</b>

dbm_close()	802
dbm_delete()	802
dbm_error()	802
dbm_fetch()	802
dbm_firstkey()	802
DBM_INSERT	311, 804
dbm_nextkey()	802
dbm_open()	802
DBM_REPLACE	311, 804
dbm_store()	802
DC	8
dc	
rationale for omission	3913
dcgettext()	807, 1192
dcgettext_l()	1192
dcngettext()	1192
dcngettext_l()	1192
dd	2776, 3864, 3928-3929
DEAD_PROCESS	480, 853-854
decimal-point character	45, 74
DECIMAL_DIG	258
declaration utility	45
default initialization	95
DEFECHO	501
deferred cancelability	1732
defined types	562, 3837
definitions	3642
delay process execution	2085
DELAYTIMER_MAX	283, 2199, 2273, 3937
delta	2787
dependency order	822
descriptive name	1022
destroying a mutex	1747
destructor functions	1735
detaching a thread	1720
determinism	3920
device	46
output	197
device ID	46
device number	3652
device, logical	3660
DEV_BSIZE	417
dev_t	425
df	2791, 3864, 3929-3930
dgettext()	1192
dgettext_l()	1192
diff	2795, 3928-3929
binary output format	2797
default output format	2797
directory comparison format	2796
-c or -C output format	2798
-e output format	2798

-f output format .....	2798
-u or -U output format .....	2799
difftime() .....	<b>808</b>
DIR .....	235, 562, 751, 1858, 1861, 1906, 1928, 2247
dircmp	
rationale for omission .....	3913
direct I/O .....	3652
directive .....	995, 1037, 1081, 1091
directory .....	46, 3652
device .....	3717
entry .....	46, 3652
files .....	3717
list .....	3078
operations .....	96, 922, 3677
protection .....	96, 3677
root .....	3665
stream .....	46
structure .....	3717
directory commands	
cd .....	2690
pwd .....	3317
dirent .....	236, 922
dirfd() .....	<b>809</b>
dirname .....	<b>2804</b> , 3927, 3929
dirname() .....	<b>811</b>
DIRTYPE .....	437
dis	
rationale for omission .....	3913
disarm (a timer) .....	46
disk space commands	
df .....	2791
du .....	2807
ulimit .....	3476
display .....	46, 3652
display line .....	46
div() .....	<b>814</b>
dladdr() .....	<b>815</b>
dlclose() .....	<b>817</b>
dLError() .....	<b>819</b>
dli_ .....	499
dlopen() .....	<b>821</b>
dlsym() .....	<b>824</b>
Dl_info_t .....	238
dngettext() .....	<b>827</b> , 1192
dngettext_l() .....	1192
documentation .....	16, 3164
dollar-sign .....	46
dollar-single-quotes .....	2474, 3868
domain error .....	109
dot .....	47, 922, 1902, 2534, 3653
dot-dot .....	47, 922, 1902, 3653, 3663, 3684
dot-po file .....	47



double-quote .....	47, 2473, 3867
downshifting .....	47
dprintf() .....	828, 995
drand48() .....	829
driver .....	47
DT_ .....	499
du .....	2807, 3864, 3929-3930
dup() .....	833, 3773, 3924
dup2() .....	833, 3773, 3924
dup3() .....	833, 3773, 3924
duplicating an input file descriptor .....	2497
duplicating an output file descriptor .....	2497
duplocale() .....	836
DUP_COUNT .....	191
dynamic package initialization .....	1785
d_ .....	499
D_FMT .....	277
D_T_FMT .....	277
E2BIG .....	242, 507
EACCES .....	242, 508
EADDRINUSE .....	242, 508
EADDRNOTAVAIL .....	242, 508
EAFNOSUPPORT .....	242, 508
EAGAIN .....	242, 508, 513
EAI_AGAIN .....	316, 1099
EAI_BADFLAGS .....	316, 1099
EAI_FAIL .....	316, 1099
EAI_FAMILY .....	316, 1099
EAI_MEMORY .....	316, 1099
EAI_NONAME .....	316, 1099
EAI_OVERFLOW .....	316, 1099
EAI_SERVICE .....	316, 1099
EAI_SOCKTYPE .....	316, 1099
EAI_SYSTEM .....	316, 1099
EALREADY .....	242, 508
EBADF .....	242, 508
EBADMSG .....	242, 508
EBUSY .....	242, 508, 3745, 3818
ECANCELED .....	242, 508, 3743
ECHILD .....	242, 508
ECHO .....	442
echo .....	2811, 3927
ECHOCTL .....	501
ECHOE .....	442, 3724
ECHOK .....	442, 3724
ECHOKE .....	501
ECHONL .....	442, 3724
ECHOPRT .....	501
ECONNABORTED .....	242, 508
ECONNREFUSED .....	242, 508
ECONNRESET .....	242, 508
ed .....	2815, 3928-3929

addresses .....	2817
append command .....	2820
change command .....	2820
commands .....	2819
copy command .....	2826
delete command .....	2821
edit command .....	2821
edit without checking command .....	2821
filename command.....	2821
global command.....	2822
global non-matched command .....	2826
help command.....	2822
help-mode command.....	2823
insert command.....	2823
interactive global command .....	2822
interactive global not-matched command.....	2826
join command .....	2823
line number command .....	2827
list command .....	2823
mark command .....	2823
move command.....	2824
null command.....	2827
number command.....	2824
print command .....	2824
prompt command .....	2824
quit command.....	2824
quit without checking command.....	2824
read command.....	2825
regular expressions .....	2817
shell escape command.....	2827
substitute command .....	2825
undo command .....	2826
write command .....	2827
EDEADLK.....	242, 508
EDESTADDRREQ.....	242, 509
edit buffer .....	2838, 3526
edit line .....	3371
editors	
ed .....	2815
ex.....	2838
sed.....	3354
vi.....	3526
EDOM.....	242, 509, 3745
EDQUOT .....	242, 509
ED_FILE_MAX.....	2829
ED_LINE_MAX.....	2829
EEXIST .....	242, 509
EFAULT .....	243, 509, 3743
EFBIG .....	243, 509
effective group ID.....	47, 728, 875, 1135, 2453
effective user ID.....	47, 586, 875, 1313, 2453, 3677
EFTYPE.....	3743

EHOSTUNREACH .....	243, 509
EIDRM .....	243, 509
eight-bit transparency.....	48
Eighth Edition UNIX .....	2441, 2732
EILSEQ.....	243, 509, 525, 3745
EINPROGRESS.....	243, 509, 529, 3767
EINTR.....	243, 509, 545, 3743, 3746, 3756-3757
EINVAL .....	243, 509, 3743
EIO.....	243, 509
EISCONN .....	243, 510
EISDIR.....	243, 510
ELOOP .....	243, 510, 3743
ELSIZE .....	1379
emacs	
rationale for omission.....	3914
EMFILE .....	243, 510
EMLINK .....	243, 510
EMPTY .....	480, 854
empty directory .....	48, 3653
empty line.....	48
empty string (or null string).....	48
empty wide-character string .....	48
EMSGSIZE.....	243, 510
EMULTIHOP .....	243, 510
ENAMETOOLONG.....	243, 510, 3744
encoding	
character .....	120
encoding rule .....	48
encrypt().....	838
encryption .....	22
endgrent() .....	840, 3675
endhostent().....	842
endnetent() .....	844
endprotoent() .....	846
endpwent() .....	848, 3675
endservent().....	851
endutxent() .....	853
ENETDOWN .....	243, 510
ENETRESET .....	243, 510
ENETUNREACH.....	243, 510
ENFILE .....	243, 510
ENOBUFS.....	243, 510
ENODEV .....	243, 511
ENOENT.....	243, 511
ENOEXEC .....	243, 511
ENOLCK.....	243, 511
ENOLINK.....	243, 511
ENOMEM.....	243, 511, 3744
ENOMSG .....	243, 511
ENOPROTOPT .....	243, 511
ENOSPC.....	243, 511
ENOSYS.....	243, 511, 3744, 3770

ENOTCONN.....	243, 511
ENOTDIR.....	244, 511
ENOTEMPTY.....	244, 511
ENOTRECOVERABLE.....	244, 511
ENOTSOCK.....	244, 511
ENOTSUP.....	244, 512, 3744
ENOTTY.....	244, 512, 3718, 3743-3744
entire regular expression.....	48, 179
ENTRY.....	1215
env.....	<b>2834</b> , 3927, 3930
environ.....	<b>856</b> , 875
environment access.....	3920, 3925
environment variable.....	3704
definition.....	3704
internationalization.....	169
envp.....	875
ENXIO.....	244, 512
EOF.....	377
EOPNOTSUPP.....	244, 512
E_OVERFLOW.....	244, 512, 3744
EOWNERDEAD.....	244, 512
EPERM.....	244, 512, 2746, 3827
EPIPE.....	244, 512, 3745
Epoch.....	48, 3653, 3685, 3790
EPROTO.....	244, 512
EPROTONOSUPPORT.....	244, 512
EPROTOTYPE.....	244, 512
equivalence class.....	49
era.....	49
ERA.....	277
erand48().....	829, 857
ERANGE.....	244, 512, 3745
ERASE.....	3721
ERA_D_FMT.....	277
ERA_D_T_FMT.....	277
ERA_T_FMT.....	277
ERE.....	3715
alternation.....	3716
bracket expression.....	3715
expression anchoring.....	3716
grammar.....	3717
grammar lexical conventions.....	3716
matching a collating element.....	3715
matching a single character.....	3715
matching multiple characters.....	3716
ordinary character.....	3715
periods.....	3715
precedence.....	3716
special character.....	3715
erf().....	<b>858</b>
erfc().....	<b>861</b>
erfcf().....	861

erfcl()	861
erff()	858, 863
erfl()	858, 863
EROFS	244, 512, 3745
errno	864, 3742
per-thread	3745
error conditions	3687, 3894
mathematical functions	109
error descriptions	1099
error handling	3907
error numbers	507, 3742, 3746
additional	513
escape character	3867
escape character (backslash)	2473
escape sequence	179
escape sequences	
awk	2616
gencat	2961
lex	3042
ESOCKTNOSUPPORT	244, 512
ESPIPE	244, 512
ESRCH	244, 513
EST5EDT	2310
establish cancellation handlers	1691
establish the locale	2457
ESTALE	244, 513
ETIMEDOUT	244, 513
ETXTBSY	244, 513
eval	2536
event management	49
EWOULDBLOCK	244, 513
ex	2838, 3928-3929
<backslash>	2851
<control>-D command	2875
<newline>	2851
abbreviate command	2854
addressing	2845
adjust window command	2872
append command	2855
args command	2855
autoindent option	2877
autoprint option	2877
autowrite option	2878
beautify option	2878
change command	2855
chdir command	2856
command descriptions	2852
copy command	2856
delete command	2856
directory option	2878
edcompatible option	2878
edit command	2856

edit options .....	2877
errorbells option .....	2878
escape command .....	2873
execute command .....	2875
exrc option.....	2878
file command .....	2857
global command .....	2858
ignorecase option .....	2879
initialization .....	2842
input editing .....	2850
insert command.....	2859
join command .....	2859
list command .....	2860
list option.....	2879
magic option .....	2879
map command.....	2860
mark command .....	2861
mesg option.....	2879
move command.....	2862
next command .....	2862
number command.....	2863
number option .....	2879
open command .....	2863
paragraphs option.....	2880
preserve command.....	2838, 2864
print command .....	2864
prompt option.....	2880
put command.....	2864
quit command.....	2865
read command .....	2865
readonly option .....	2880
recover command.....	2865
redraw option .....	2880
regular expressions .....	2875
remap option.....	2880
replacement strings.....	2876
report option .....	2881
rewind command .....	2866
scroll command .....	2850
scroll option.....	2881
sections option.....	2881
set command.....	2866
shell command.....	2867
shell option.....	2881
shift left command .....	2874
shift right command .....	2874
shiftwidth option.....	2882
showmatch option.....	2882
showmode option .....	2882
slowopen option.....	2882
source command .....	2867
substitute command .....	2867

suspend command .....	2868
tabstop option .....	2882
tag command .....	2868
taglength option .....	2882
tags option .....	2883
term option .....	2883
terse option .....	2883
unabbrev command .....	2869
undo command .....	2869
unmap command .....	2870
version command .....	2870
visual command .....	2870
warn option .....	2883
window option .....	2883
wrapmargin option .....	2884
wrapscan option .....	2884
write command .....	2871
write line number command .....	2875
writeany option .....	2884
xit command .....	2872
yank command .....	2872
examine and change blocked signals .....	1820
examine and change signal action .....	2046
EXDEV .....	244, 513
exec .....	866, 2538, 3223
of shell scripts .....	874
exec family .....	586, 747, 910, 956, 986, 1643, 1998, 2353, 2470, 2733, 3206, 3605, 3657
.....	3771, 3859, 3904, 3924
execl() .....	866
execle() .....	866
execlp() .....	866
executable file .....	49
execute .....	49
execute a file .....	874
execution time .....	44, 49, 3652
measurement .....	99, 3681
monitoring .....	49, 536, 3795
execution unit .....	2201
execv() .....	866
execve() .....	866
execvp() .....	866
EXINIT .....	2838
exit .....	2541
exit status .....	3894
and errors .....	2497
for commands .....	2499
exit() .....	880, 3754, 3924
EXIT_FAILURE .....	381, 568, 880, 3926
EXIT_SUCCESS .....	381, 568, 880, 3926
exp() .....	882
exp2() .....	884
exp2f() .....	884

exp2l()	884
expand	50, 2912, 3928
expf()	882
expl()	882
expm1()	886
expm1f()	886
expm1l()	886
export	2544
expr	2915, 3927-3928
matching expression	2917
expression argument	2619
expression list	2619
EXPR_NEST_MAX	287, 2199, 2460, 3858
EXTA	501
EXTB	501
extended regular expression	50, 187, 2615, 2745, 2944, 2991, 3041, 3200, 3335, 3602 3715
extended security controls	50, 96, 3677
extension	
CX	7
OH	9
XSI	12
F-LOCK	467
fabs()	888
fabsf()	888
fabsl()	888
faccessat()	584, 890
false	2920, 3927
fc	2470, 2922, 3865, 3928
fchdir()	891
fchmod()	892, 3924
fchmodat()	721, 894
fchown()	895
fchownat()	726, 897
fclose()	898, 3924
fcntl()	901, 3718, 3743, 3774, 3924
fcntl() locks	3834
fdatasync()	913
fdim()	915
fdimf()	915
fdiml()	915
fdopen()	917, 3773
fdopendir()	920
fds_	499
FD_	499
fd_	499
FD_CLOEXEC	247, 409, 523, 701, 754, 833-834, 842, 844, 846, 851, 867, 901, 917, 920 1226, 1503, 1515, 1547, 1582, 1592, 1627, 2023
FD_CLOFORK	247, 409, 523, 833-834, 901, 1515, 1547, 1582, 1627
FD_CLR	1635
FD_CLR()	567
FD_ISSET	567, 1635



fd_set.....	400, 422
FD_SET .....	567, 1635
FD_SETSIZE.....	400
FD_ZERO .....	567, 1635
feature test macro.....	50, 496, 1103, 3737-3738, 3838
_POSIX_C_SOURCE.....	496
_XOPEN_SOURCE .....	497
__STDC_WANT_LIB_EXT1__.....	497
feclearexcept() .....	924
fegetenv() .....	925
fegetexceptflag().....	926
fegetround().....	927
feholdexcept().....	929
fenv_t .....	252
feof() .....	930
feraiseexcept() .....	931
ferror() .....	932
fesetenv().....	925, 933
fesetexceptflag() .....	926, 934
fesetround() .....	927, 935
fetestexcept() .....	936
feupdateenv() .....	938
fexcept_t.....	252
fexecve .....	940
fexecve() .....	866
FE_.....	501
FE_ constants	
defined in <fenv.h> .....	252
FE_ALL_EXCEPT.....	252
FE_DFL_ENV .....	253
FE_DIVBYZERO.....	252
FE_DOWNWARD.....	252
FE_INEXACT.....	252
FE_INVALID.....	252
FE_OVERFLOW .....	252
FE_TONEAREST.....	252
FE_TOWARDZERO .....	252
FE_UNDERFLOW .....	252
FE_UPWARD.....	252
FFDLY .....	441
fflush() .....	941
FFn.....	441
ffs().....	945
fg.....	2470, 2928, 3865, 3928
fgetc().....	946, 3925
fgetpos() .....	948
fgets().....	950
fgetwc() .....	952
fgetws() .....	954
field.....	50
field splitting.....	2491, 3889
FIFO.....	51, 1417, 1419, 1521, 2439, 3653, 3663, 3783

FIFO special file .....	51, 3175, 3653
FIFOTYPE .....	437
file .....	51
FILE .....	377, 482, 562
file .....	<b>2931</b> , 3653
locking.....	910
file access permissions.....	97, 2454, 3677
file accessibility .....	586
file characteristics	
data structure .....	417
header .....	417
file classes .....	3653
file comparisons	
cmp .....	2720
comm.....	2724
diff .....	2795
uniq .....	3497
file contents .....	2456
file control.....	910
file conversion	
cut .....	2762
dd.....	2776
expand .....	2912
fold .....	2952
head.....	3000
join.....	3026
od.....	3226
paste .....	3234
patch.....	3238
sort.....	3388
strings.....	3400
tail .....	3419
tr.....	3459
tsort.....	3468
unexpand.....	3491
uniq .....	3497
uudecode .....	3508
uuencode .....	3511
file creation.....	2454, 3856
file description .....	51
file descriptor.....	51, 2453, 2494, 2503, 3757-3758, 3890
file format notation .....	3688
file group class .....	51
file hierarchy .....	97, 3678
file hierarchy manipulation .....	3922, 3929
file lock.....	51
file mode .....	52
file mode bits.....	52
file mode creation mask .....	2453
FILE object.....	521
file offset .....	52
file other class .....	52

file owner class .....	52
file permission bits .....	53, 586
file permission commands	
chgrp .....	2701
chmod .....	2704
chown.....	2711
umask.....	3480
file permissions.....	586, 991, 1058, 3677, 3720
file position indicator.....	521
file read .....	2454, 3856
file removal.....	2456, 3856
file searching	
grep.....	2991
file serial number.....	53
file size, arbitrary.....	3864
file system.....	53, 3654
file system cache.....	98, 3680
file system, mounted.....	3660
file system, root.....	3665
file time values.....	2456
file times update.....	98, 3680
file tree commands	
diff .....	2795
find .....	2940
ls.....	3078
mkdir.....	3171
rmdir .....	3343
file type .....	53
file write.....	2454, 3856
file, passwd.....	3663
filename .....	52, 97, 3654, 3678
filename portability.....	98, 3679
filename string .....	52
FILENAME_MAX.....	376
fileno() .....	956, 3662
FILESIZEBITS .....	285, 988
filter .....	53
filters	
asa .....	2592
awk .....	2605
compress.....	2735
dd.....	2776
expand .....	2912
fold .....	2952
head .....	3000
iconv .....	3003
more .....	3178
nl .....	3213
paste .....	3234
pax .....	3250
pr.....	3290
read.....	3320

sed.....	3354
tail.....	3419
tee.....	3428
tr.....	3459
uncompress.....	2735
unexpand.....	3491
zcat.....	2735
FIND.....	1215
find.....	<b>2940</b> , 3928-3929
find string token.....	2178
FIPS.....	17
FIPS requirements.....	3638
first open (of a file).....	53
flockfile().....	<b>957</b> , 3746
floor().....	<b>959</b>
floorf().....	959
floorl().....	959
flow control.....	53
FLT_ constants	
defined in <float.h>.....	<b>257</b>
FLT_DECIMAL_DIG.....	258
FLT_DIG.....	258
FLT_EPSILON.....	259
FLT_EVAL_METHOD.....	256
FLT_MANT_DIG.....	257
FLT_MAX.....	259
FLT_MAX_10_EXP.....	259
FLT_MAX_EXP.....	259
FLT_MIN.....	260, 620, 622, 629, 631, 635, 858, 882, 884, 886, 915, 969
FLT_MIN_10_EXP.....	259
FLT_MIN_EXP.....	258
FLT_RADIX.....	257, 1369
FLT_ROUNDS.....	256, 961
FLT_TRUE_MIN.....	260
FLUSHO.....	501
fma().....	<b>961</b>
fmaf().....	961
fmal().....	961
fmax().....	<b>963</b>
fmaxf().....	963
fmaxl().....	963
fmemopen().....	<b>964</b>
fmin().....	<b>968</b>
fminf().....	968
fminl().....	968
fmod().....	<b>969</b>
fmodf().....	969
fmodl().....	969
fmtmsg().....	<b>971</b>
fnmatch().....	<b>974</b> , 3930
FNM_.....	499

FNM_constants	
in <fnmatch.h> .....	<b>263</b>
FNM_CASEFOLD .....	263
FNM_IGNORECASE .....	263
FNM_NOESCAPE .....	263, 974
FNM_NOMATCH .....	263, 974
FNM_PATHNAME .....	263, 974
FNM_PERIOD .....	263, 974
fold .....	<b>2952</b> , 3928
fopen() .....	<b>976</b> , 3655, 3864, 3924
FOPEN_MAX .....	284, 376, 918, 965, 979, 2280
fopen_s .....	502
for loop .....	2508, 3902
foreground .....	1997, 2518, 3656-3659, 3719-3721
foreground job .....	54, 2519
foreground process .....	54
group .....	54
group ID .....	54
fork() .....	<b>983</b> , 3657, 3720, 3763, 3771, 3773, 3838, 3842, 3924
forkall .....	986
form-feed character .....	54
format of entries .....	<b>221</b> , 493
fpathconf() .....	<b>988</b> , 3923, 3925
fpclassify() .....	<b>994</b>
FPE_ .....	499
FPE_FLTDIV .....	351
FPE_FLTINV .....	351
FPE_FLTOVF .....	351
FPE_FLTRES .....	351
FPE_FLTSUB .....	351
FPE_FLTUND .....	351
FPE_INTDIV .....	351
FPE_INTOVF .....	351
fprintf() .....	<b>995</b>
fprintf_s .....	502
fputc() .....	<b>1009</b> , 3925
fputs() .....	<b>1011</b>
fputwc() .....	<b>1013</b>
fputws() .....	<b>1016</b>
FP_ILOGB0 .....	1232
FP_ILOGBNAN .....	1232
FQDN .....	1146
FR .....	<b>8</b>
frac_digits .....	148
fread() .....	<b>1018</b> , 3925
free() .....	<b>1020</b> , 3754, 3831
freeaddrinfo() .....	<b>1022</b>
freelocale() .....	<b>1028</b>
freopen() .....	<b>1030</b> , 3924
freopen_s .....	502
frexp() .....	<b>1035</b>
frexpf() .....	1035

frexpl()	1035
fsblkcnt_t	425
FSC	8
fscanf()	1037
fscanf_s	502
fseek()	1045, 3925
fseeko()	1045
fsetpos()	1049, 3925
fsfilcnt_t	425
fstat()	1052, 3924
fstatat()	1055
fstatvfs()	1061
fsync()	1064, 3767
ftell()	1066
ftello()	1066
ftok()	1068
ftruncate()	1070, 3773, 3775, 3924
ftrylockfile()	957, 1073
FTW	264, 499, 1502-1503
ftw()	3840
FTW_constants	
in <ftw.h>	264
FTW_CHDIR	264, 1502
FTW_D	264, 1502
FTW_DEPTH	264, 1502
FTW_DNR	264, 1502-1503
FTW_DP	264, 1502
FTW_F	264, 1503
FTW_MOUNT	264, 1502
FTW_NS	264, 1503
FTW_PHYS	264, 1502
FTW_SL	264, 1503
FTW_SLN	264, 1503
FTW_XDEV	264, 1502
fully-qualified domain name	1146
function definition command	2511, 3903
function identifiers	2656
functions	495
implementation	495, 3735
use	495, 3735
funlockfile()	957, 1074
fuser	2956
futimens()	1075, 3924
fwide()	1079
fwprintf()	1081
fwprintf_s	502
fwrite()	1089, 3925
fwscanf()	1091
fwscanf_s	502
f_	500
F_	501
F_DUPFD	246, 901, 906

F_DUPFD_CLOEXEC.....	246, 901, 906
F_DUPFD_CLOFORK.....	246, 901, 906
F_GETFD.....	246, 901, 906
F_GETFL.....	246, 901, 906
F_GETLK.....	246, 903, 906
F_GETOWN.....	246, 902, 906
F_GETOWN_EX.....	247, 902, 906
F_LOCK.....	543, 1358
F_OFD_GETLK.....	246, 903, 906
F_OFD_SETLK.....	246, 903, 906
F_OFD_SETLKW.....	246, 543, 903, 906
F_OK.....	464
F_OWNER_PGRP.....	247
F_OWNER_PID.....	247
F_RDLCK.....	247, 906
F_SETFD.....	246, 901, 906
F_SETFL.....	246, 901, 906
F_SETLK.....	246, 903, 906
F_SETLKW.....	246, 543, 903, 906
F_SETOWN.....	247, 902, 906
F_SETOWN_EX.....	247, 902, 906
F_TEST.....	467, 1358
F_TLOCK.....	467, 1358
F_ULOCK.....	467, 1358
F_UNLCK.....	247, 903, 905
F_WRLCK.....	247
g-file.....	2787
gai_strerror().....	1099
gencat.....	2960, 3929
escape sequences.....	2961
general terminal interface.....	3718
generated file.....	2787
get.....	2964
get configurable pathname variables.....	991
get configurable system variables.....	2202
get file status.....	1058
get process times.....	2276
get supplementary group IDs.....	1135
get system time.....	2266
get thread ID.....	1811
get user name.....	1144
getaddrinfo().....	1022, 1100
GETALL.....	402, 1955
getc().....	1101, 3821, 3925
getch().....	3925
getchar().....	1104
getchar_unlocked().....	1102, 1105
getconf.....	2973, 3857, 3923, 3930
getcwd().....	1106
getc_unlocked().....	1102
getdate().....	1109
getdate_err.....	1109

getdelim()	1114
getegid()	1117, 3924
getentropy()	1119
GETENTROPY_MAX	293
getenv()	875, 1120
getenv_s	502
geteuid()	1123, 3924
getgid()	1125, 3924
getgrent()	840, 1127, 3675
getgrgid()	1128, 3675, 3822, 3924
getgrgid_r()	1128
getgrnam()	1132, 3675, 3680, 3822, 3924
getgrnam_r()	1132
getgroups()	1135, 3666
gethostent()	842, 1137
gethostid()	1138
gethostname()	1139
getitimer()	3841
getline()	1114, 1140
getlocalename_l()	1141
getlogin()	1143, 3924
getlogin_r()	1143
getnameinfo()	1146
GETNCNT	402, 1955-1956
getnetbyaddr()	844, 1149
getnetbyname()	844, 1149
getnetent()	844, 1149
getopt()	1150, 3726, 3929-3930
getopts	2470, 2979, 3865, 3930
getpeername()	1155
getpgid()	1157
getpgrp()	1158, 3658
GETPID	402, 1955-1956
getpid()	1159, 3757, 3924
getppid()	1160, 3924
getpriority()	1161, 3784
getprotent()	1164
getprotobyname()	846, 1164
getprotobynumber()	846, 1164
getprotoent()	846
getpwent()	848, 1165, 3675
getpwnam()	1166, 3675, 3680, 3822, 3924
getpwnam_r()	1166
getpwuid()	1170, 3675, 3822, 3924
getpwuid_r()	1170
getresgid()	1174
getresuid()	1175
getrlimit()	1176, 3864
getrusage()	1180, 3797
gets()	3841
getservbyname()	851, 1182
getservbyport()	851, 1182



getservent()	851, 1182
getsid()	<b>1183</b>
getsockname()	<b>1184</b>
getsockopt()	<b>1186</b>
getsubopt()	<b>1188</b>
gets_s	502
gettext	<b>1192</b> , 2985
gettext_l()	1192
gettimeofday()	3841
getty	3720
getuid()	<b>1200</b> , 3757, 3839, 3924
getutxent()	853, 1202
getutxid()	853, 1202
getutxline()	853, 1202
GETVAL	402, 1955-1956
getwc()	<b>1203</b>
getwchar()	<b>1204</b>
GETZCNT	402, 1955-1956
gid_t	425, 3675
glob()	<b>1205</b> , 3930
global storage class	2660
globfree()	1205, 3930
GLOB_	499
GLOB_ constants	
defined in <glob.h>	<b>266</b>
error returns of glob	1207
used in glob	1205
GLOB_ABORTED	266, 1207
GLOB_APPEND	266, 1205-1206
GLOB_DOOFFS	266, 1205-1206
GLOB_ERR	266, 1205, 1207
GLOB_MARK	266, 1206
GLOB_NOCHECK	266, 1206-1207
GLOB_NOESCAPE	266, 1206
GLOB_NOMATCH	266, 1207
GLOB_NOSORT	266, 1206
GLOB_NOSPACE	266, 1207
gl_	499
GMT0	2310
gmtime()	<b>1211</b> , 3685
gmtime_r()	1211
gmtime_s	502
GNU make	3155
grammar	
conventions	3860
locale	160
regular expression	191
grantpt()	<b>1213</b>
graphic character	54
grep	<b>2991</b> , 3928-3929
group database	55, 3654
group database access	3675

group file .....	3654
group ID .....	55
group name .....	55
grouping commands.....	2508, 3901
HALT.....	972
hard limit.....	55
hard link.....	46, 55, 58, 1332, 3051, 3652-3653, 3655
hash .....	2470, 2997, 3865
hcreate().....	<b>1215</b>
hdestroy().....	1215
head.....	<b>3000</b> , 3928
headers.....	<b>221</b> , 3728
here-document.....	2495, 3893
high resolution sleep.....	1494
historical implementations .....	3655
history command	
fc .....	2922
hole.....	55
HOME.....	<b>174</b> , 2856, 3639
home directory.....	56
host byte order.....	56, 99, 3681
host name .....	1022
hosted implementation .....	3655
hostent.....	<b>314</b>
HOST_NAME_MAX .....	283
hsearch().....	1215
htobe16() .....	660, 1218
htobe32() .....	660, 1218
htobe64() .....	660, 1218
htole16() .....	660, 1218
htole32() .....	660, 1218
htole64() .....	660, 1218
htonl().....	<b>1219</b>
htons() .....	1219
HUGE_VAL.....	301, 2396
HUGE_VALF .....	301
HUGE_VALL .....	301
hunk .....	3240
HUPCL .....	442
hypot().....	<b>1220</b>
hypotf() .....	1220
hypotl().....	1220
h_ .....	499
I .....	227
IANA timezone database.....	178, 3708
ICANON .....	442, 3721, 3724
iconv .....	<b>3003</b>
iconv().....	<b>1222</b> , 3929
iconv_close().....	<b>1225</b> , 3929
iconv_open().....	<b>1226</b> , 3929
ICRNL.....	440
id.....	<b>3007</b> , 3930

idtype_t.....	433
id_t.....	425
IEEE Std 754-1985.....	493
IEEE Std 854-1987.....	493
IEXTEN.....	442
if.....	3903
if conditional construct.....	2510
ifc_.....	500
ifra.....	500
ifru.....	500
IF.....	499
if_.....	499-500
if_freenameindex().....	1228
if_indextoname().....	1229
if_nameindex.....	313
if_nameindex().....	1230
IF_NAMESIZE.....	313
if_nametoindex().....	1231
IGNBRK.....	440
IGNCR.....	440
ignore_handler_s.....	502
IGNPAR.....	440
ILL.....	499
ILL_BADSTK.....	351
ILL_COPROC.....	351
ILL_ILLADR.....	351
ILL_ILLOPC.....	351
ILL_ILLOPN.....	351
ILL_ILLTRP.....	351
ILL_PRVOPC.....	351
ILL_PRIVREG.....	351
ilogb().....	1232
ilogbf().....	1232
ilogbl().....	1232
imaginary.....	227
imaxabs().....	1235
imaxdiv().....	1236
implementation.....	3655
historical.....	3655
hosted.....	3655
native.....	3661
specific.....	3655
implementation-defined.....	5, 3639-3641
rationale.....	3639
IMPLINK.....	501
in6addr.....	499
in6addr_any.....	1237
in6addr_loopback.....	1237
in6.....	499
IN6.....	501
IN6_IS_ADDR_LINKLOCAL.....	321
IN6_IS_ADDR_LOOPBACK.....	320

IN6_IS_ADDR_MC_GLOBAL .....	321
IN6_IS_ADDR_MC_LINKLOCAL .....	321
IN6_IS_ADDR_MC_NODELOCAL .....	321
IN6_IS_ADDR_MC_ORGLOCAL .....	321
IN6_IS_ADDR_MC_SITELOCAL .....	321
IN6_IS_ADDR_MULTICAST .....	321
IN6_IS_ADDR_SITELOCAL .....	321
IN6_IS_ADDR_UNSPECIFIED .....	320
IN6_IS_ADDR_V4COMPAT .....	321
IN6_IS_ADDR_V4MAPPED .....	321
INADDR_ .....	499
include line .....	3135
incomplete line .....	56
incomplete pathname .....	3655
INET6_ADDRSTRLEN .....	320
inet_ .....	499
inet_addr() .....	<b>1238</b>
INET_ADDRSTRLEN .....	320
inet_ntoa() .....	1238
inet_ntop() .....	<b>1240</b>
inet_pton() .....	1240
Inf .....	56, 620, 622, 629, 635, 886
INF .....	999, 1084
Inf .....	1365, 2080, 2082, 2212, 2215
inference rule .....	3130
INFINITY .....	301, 999, 1084
INFO .....	972
infu_ .....	500
init .....	571, 1313
initialization .....	3677
initialize a named semaphore .....	1945
initializing a mutex .....	1747
initstate() .....	<b>1242</b>
INIT_PROCESS .....	480, 853-854
INLCR .....	440
ino_t .....	425
INPCK .....	440
input and output rationale .....	1854
input file descriptor	
duplication .....	3893
input mode .....	2815, 3723
input processing .....	3721
canonical mode .....	3721
non-canonical mode .....	3722
insque() .....	<b>1245</b>
INT .....	501
inter-user communication .....	3922, 3929
interactive device .....	56, 3655
interactive facilities .....	3921, 3928
interactive shell .....	56
interface .....	3836
characteristics .....	3719

international environment .....	1991
internationalization .....	56
internationalization variable .....	3705
Internet Protocols .....	558
interprocess communication .....	56, 3758
INTMAX_MAX .....	373
INTMAX_MIN .....	373
INTN_MAX .....	372
INTN_MIN .....	372
INTPTR_MAX .....	373
INTPTR_MIN .....	373
intrinsic utilities .....	2470, 3656, 3865
intrinsic utility .....	57
int_curr_symbol .....	148
INT_FASTN_MAX .....	373
INT_FASTN_MIN .....	372
int_frac_digits .....	148
INT_LEASTN_MAX .....	372
INT_LEASTN_MIN .....	372
INT_MAX .....	291, 1232
INT_MIN .....	291, 579
int_n_cs_precedes .....	149
int_n_sep_by_space .....	149
int_n_sign_posn .....	149
int_p_cs_precedes .....	149
int_p_sep_by_space .....	149
int_p_sign_posn .....	149
invalid .....	180
use in RE .....	3711
invariant values .....	293
invoke .....	57
in_ .....	499
IN_ .....	501
in_addr .....	318
ioctl() .....	3718, 3744
iovec .....	429
iov_ .....	500
IOV_ .....	501
IOV_MAX .....	283, 429, 1867, 2199, 2444
IP6 .....	8
IPC .....	390, 526, 1472, 1474, 1477, 1479, 1959, 1964, 2034, 2037, 3758
ipcrm .....	3011
ipcs .....	3014
IPC_ .....	499
ipc_ .....	499
IPC_ constants	
defined in <sys/ipc.h> .....	390
used in semctl .....	1956
used in shmctl .....	2032
IPC_CREAT .....	390, 1473, 1958, 2036
IPC_EXCL .....	390, 1473, 1958
IPC_NOWAIT .....	390, 1475-1476, 1478-1479, 1960

IPC_PRIVATE .....	390, 1473, 1958, 2036
IPC_RMID .....	390, 1471, 1956, 2032
IPC_SET .....	390, 1471, 1956, 2032
IPC_STAT .....	390, 1471, 1956, 2032
IPPORT_ .....	501
IPPROTO_ .....	499
IPPROTO_ICMP .....	319
IPPROTO_IP .....	319
IPPROTO_IPV6 .....	319
IPPROTO_RAW .....	320
IPPROTO_TCP .....	320
IPPROTO_UDP .....	320
IPv4.....	558
IPv4-compatible address.....	559
IPv4-mapped address.....	559
IPv6.....	558
compatibility with IPv4.....	559
interface identification.....	560
options .....	560
IPv6 address	
anycast .....	559
loopback .....	559
multicast .....	559
unicast.....	559
unspecified .....	559
IPV6_ .....	499
IPV6_JOIN_GROUP .....	320, 560
IPV6_LEAVE_GROUP.....	320, 560
ipv6_mreq.....	<b>319</b>
IPV6_MULTICAST_HOPS.....	320, 560
IPV6_MULTICAST_IF.....	320, 560
IPV6_MULTICAST_LOOP .....	320, 561
IPV6_UNICAST_HOPS.....	320, 561
IPV6_V6ONLY .....	320, 561
ip_ .....	499
IP_ .....	501
isalnum().....	<b>1248</b>
isalnum_l().....	1248
isalpha() .....	<b>1250</b>
isalpha_l() .....	1250
isascii().....	3841
isatty().....	<b>1252</b>
isblank() .....	<b>1253</b>
isblank_l() .....	1253
iscntrl() .....	<b>1255</b>
iscntrl_l().....	1255
isdigit() .....	<b>1257</b>
isdigit_l().....	1257
isfinite() .....	<b>1259</b>
isgraph().....	<b>1260</b>
isgraph_l().....	1260
isgreater().....	<b>1262</b>

isgreaterequal.....	1262
ISIG.....	442
isinf().....	<b>1264</b>
isless.....	1262
isless().....	<b>1265</b>
islessequal.....	1262, 1265
islessgreater.....	1262, 1265
islower().....	<b>1266</b>
islower_l().....	1266
isnan().....	<b>1269</b>
isnormal().....	<b>1270</b>
ISO/IEC 646: 1991 standard.....	3663
ISO C standard.....	227, 493, 610, 874, 910, 1103, 1382, 1849, 1902, 1991, 2047, 2060, 2069 2266, 3647, 3650, 3685, 3718, 3735
isprint().....	<b>1271</b>
isprint_l().....	1271
ispunct().....	<b>1273</b>
ispunct_l().....	1273
isspace().....	<b>1275</b>
isspace_l().....	1275
ISTRIP.....	440, 3723
isunordered().....	<b>1277</b>
isupper().....	<b>1278</b>
isupper_l().....	1278
iswalnum().....	<b>1280</b>
iswalnum_l().....	1280
iswalpha().....	<b>1282</b>
iswalpha_l().....	1282
iswblank().....	<b>1284</b>
iswblank_l().....	1284
iswcntrl().....	<b>1286</b>
iswcntrl_l().....	1286
iswctype().....	<b>1288</b>
iswctype_l().....	1288
iswdigit().....	<b>1291</b>
iswdigit_l().....	1291
iswgraph().....	<b>1293</b>
iswgraph_l().....	1293
iswlower().....	<b>1295</b>
iswlower_l().....	1295
iswprint().....	<b>1297</b>
iswprint_l().....	1297
iswpunct().....	<b>1299</b>
iswpunct_l().....	1299
iswspace().....	<b>1301</b>
iswspace_l().....	1301
iswupper().....	<b>1303</b>
iswupper_l().....	1303
iswxdigit().....	<b>1305</b>
iswxdigit_l().....	1305
isxdigit().....	<b>1307</b>
isxdigit_l().....	1307

itimerspec .....	452
it_ .....	500
IXANY .....	440
IXOFF .....	440
IXON .....	440
I_ISVTX .....	2706
j0() .....	1309
j1() .....	1309
jn() .....	1309
job .....	57
job control.....	57, 571, 1158, 1313, 1997, 2012, 2202, 2353, 2518, 3656-3659, 3662
.....	3719-3721, 3747, 3753, 3924
implementing applications .....	3658
implementing shells.....	3656
implementing systems.....	3659
job ID.....	57
jobs.....	2470, 3021, 3865, 3928
join .....	3026, 3928
joinable thread .....	58
jrnd48() .....	829, 1311
JST-9.....	2310
kernel .....	3659
kernel entity .....	3824
key_t.....	425
kill .....	2470, 3031, 3866, 3928
kill().....	1312, 3747-3748, 3751, 3753-3754, 3838
killpg().....	1316
kill_dependency().....	1315
l64a() .....	575, 1318
labs() .....	1319
LANG.....	169, 701
LANGUAGE .....	169
last close.....	2028, 3774
last close (of a file).....	58
lchown() .....	1320
lcong48().....	829, 1323
LC_ALL .....	170, 297, 868, 1351, 1510, 1990, 1992
LC_COLLATE.....	170, 287, 297, 1205-1206, 1990, 1992, 2117, 2189, 2370, 2415, 3695
description.....	139
LC_CTYPE .....	170, 277, 297, 486, 1288, 1387, 1399, 1401, 1990, 1992, 2285, 2291, 3694
description.....	131
LC_GLOBAL_LOCALE .....	836, 1028
LC_MESSAGES .....	170, 277, 297, 324, 701, 1192, 1990-1992, 2125, 3700
description.....	159
LC_MONETARY .....	170, 277, 297, 1351, 1990, 1992, 2131, 3697
description.....	147
LC_NUMERIC.....	170, 277, 297, 996, 1037, 1082, 1091, 1351, 1990, 1992, 2131, 2172
.....	2396, 3699, 3726
description.....	151
LC_TIME.....	171, 277, 297, 1110, 1511, 1990, 1992, 3699
description.....	152



ld	
rationale for omission.....	3914
LDBL_constants	
defined in <float.h>.....	257
LDBL_DECIMAL_DIG.....	258
LDBL_DIG.....	258
LDBL_EPSILON.....	259
LDBL_MANT_DIG.....	257
LDBL_MAX.....	259
LDBL_MAX_10_EXP.....	259
LDBL_MAX_EXP.....	259
LDBL_MIN.....	260, 620, 622, 629, 631, 635, 858, 882, 884, 886, 915, 969
LDBL_MIN_10_EXP.....	259
LDBL_MIN_EXP.....	258
LDBL_TRUE_MIN.....	260
ldexp().....	1324
ldexpf().....	1324
ldexpl().....	1324
ldiv().....	1326
le16toh().....	660, 1327
le32toh().....	660, 1327
le64toh().....	660, 1327
leftmost.....	179
legacy.....	5, 3640
rationale.....	3640
lex.....	3037, 3930, 3932
actions.....	3043
definitions.....	3039
escape sequences.....	3042
regular expressions.....	3041
rules.....	3040
table sizes.....	3040
translation table.....	3047
user subroutines.....	3041
lfind().....	1328, 1379
lgamma().....	1329
lgammaf().....	1329
lgammal().....	1329
libraries	
ar command.....	2584
library routine.....	3659
LIMIT.....	2459
limit	
numerical.....	291
limits.....	3857
line.....	58
rationale for omission.....	3914
line counting.....	3586
LINES.....	174, 3706
LINE_MAX.....	287, 2199, 2460, 2606, 2829, 2839, 3125, 3364, 3500, 3661, 3673, 3858
linger.....	58
link.....	58, 3049, 3659

link count.....	58
link to a file.....	1335
link() .....	<b>1332</b> , 3925
linkat() .....	1332
LINK_MAX.....	285, 510, 988, 1333, 1900, 3858, 3938
lint	
rationale for omission.....	3914
LIO_.....	499
lio_.....	499
lio_listio() .....	<b>1337</b> , 3751, 3768
LIO_NOP.....	222, 1337
LIO_NOWAIT.....	222, 1337
LIO_READ.....	222, 1337
LIO_WAIT.....	222, 1337
LIO_WRITE.....	222, 1337
list directed I/O.....	1339
listen().....	<b>1341</b>
lists.....	2505, 2519, 3900
AND-OR.....	2505
compound-list.....	2505
LITTLE_ENDIAN .....	240
live process.....	59
live thread.....	59
llabs() .....	1319, 1343
lldiv() .....	1326, 1344
LLONG_MAX.....	291, 2181, 2403
LLONG_MIN.....	292, 2181, 2403
llrint().....	<b>1345</b>
llrintf() .....	1345
llrintl() .....	1345
llround() .....	<b>1347</b>
llroundf().....	1347
llroundl().....	1347
ln.....	<b>3051</b> , 3864, 3929
LNKTYPE .....	437
load order .....	822
LOBLK .....	501
local customs.....	59
local IPC.....	59
local mode .....	3724
locale .....	59, 127, 3056, 3692, 3929
configuration.....	3922, 3929
definition .....	128, 3693
definition example .....	3701
definition grammar.....	3701
grammar .....	160, 3701
lexical conventions.....	3701
POSIX.....	128
localeconv() .....	<b>1349</b>
localedef .....	<b>3062</b> , 3929-3930
localization.....	59
localtime().....	<b>1354</b> , 3685, 3820

localtime_r()	1354
localtime_s	502
lock-free operation	59
lockf()	<b>1358</b>
locking	910
advisory	910
mandatory	910
locking and unlocking a mutex	1758
locking file	2708
log()	<b>1361</b>
log10()	<b>1363</b>
log10f()	1363
log10l()	1363
log1p()	<b>1365</b>
log1pf()	1365
log1pl()	1365
log2()	<b>1367</b>
log2f()	1367
log2l()	1367
logb()	<b>1369</b>
logbf()	1369
logbl()	1369
logf()	1361, 1371
logger	<b>3067</b> , 3930
logical device	3660
login	60
rationale for omission	3914
login name	60
login shell	874
LOGIN_NAME_MAX	283, 1143, 2199, 3938
LOGIN_PROCESS	480, 853-854
logl()	1361, 1371
LOGNAME	<b>175</b>
logname	<b>3071</b>
LOGNAME	3639, 3706
logname	3930
LOG_	500
LOG_ constants in syslog	<b>753</b>
LOG_ALERT	436, 753
LOG_AUTH	435
LOG_CONS	435, 754
LOG_CRIT	436, 753
LOG_CRON	435
LOG_DAEMON	435
LOG_DEBUG	436, 753
LOG_EMERG	436, 753
LOG_ERR	436, 753
LOG_INFO	436, 753
LOG_KERN	435
LOG_LOCAL	435, 753
LOG_LPR	435
LOG_MAIL	435

LOG_MASK	435
LOG_NDELAY	435, 754
LOG_NEWS	435
LOG_NOTICE	436, 753
LOG_NOWAIT	435, 754
LOG_ODELAY	435, 754
LOG_PID	435, 754
LOG_UPTO	436
LOG_USER	435, 753-754
LOG_UUCP	435
LOG_WARNING	436, 753
longjmp()	1372, 3743, 3754, 3829-3830, 3927
LONG_BIT	291-292
LONG_MAX	292, 2181, 2403, 3725
LONG_MIN	292, 2181, 2403, 3725
lorder	
rationale for omission	3914
lp	3073, 3930
lpstat	
rationale for omission	3914
LR(1) grammars	3627
lrand48()	829, 1374
lrint()	1375
lrintf()	1375
lrintl()	1375
lround()	1377
lroundf()	1377
lroundl()	1377
ls	3078, 3864, 3929
lsearch()	1379
lseek()	1381, 3767-3768, 3773, 3819, 3925
lstat()	1055, 1384, 3864, 3924
L	499-500
L_ANCHOR	191
L_ctermid	376, 797
l_sysid	910
L_tmpnam	376
L_tmpnam_s	502
m4	3090
macro	3641
macro processor	3090
macros	
implementation	496, 3737
use	496, 3737
MAGIC	230
magic file	2936
mail	
rationale for omission	3914
mailx	3102, 3928-3929
change current directory	3113
change folder	3115
command escapes	3122

commands .....	3112
copy messages.....	3113
declare aliases .....	3113
declare alternatives .....	3113
delete aliases .....	3120
delete messages .....	3114
delete messages and display.....	3114
direct messages to mbox.....	3117
discard header fields.....	3114
display beginning of messages .....	3120
display current message number.....	3122
display header summaries.....	3116
display header summary.....	3116
display list of folders.....	3115
display message.....	3117-3118
display message size.....	3120
echo a string .....	3114
edit message.....	3114, 3121
execute commands conditionally.....	3116
exit .....	3115
follow up specified messages .....	3115
help .....	3116
hold messages.....	3116
internal variables.....	3109
invoke a shell .....	3120
invoke shell command .....	3121
list available commands.....	3116
mail a message.....	3117
null command.....	3122
pipe message.....	3117
process next specified message .....	3117
quit.....	3118
read mailx commands from a file .....	3120
receive mode .....	3102
reply to a message .....	3118
reply to a message list.....	3118
retain header fields.....	3119
save messages .....	3119
scroll header display .....	3121
send mode .....	3102
set variables.....	3120
start-up.....	3109
touch messages.....	3120
undelete messages.....	3121
unset variables.....	3121
write messages to a file.....	3121
main() .....	3754
make.....	3130, 3930-3931
default rules .....	3146
inference rules.....	3143
internal macros .....	3144
libraries .....	3144

macros.....	3139
makefile execution .....	3136
makefile syntax.....	3135
target rules.....	3137
make, GNU version .....	3155
malloc() .....	<b>1385</b> , 3754, 3777-3778, 3807, 3820-3821, 3831-3832
man.....	<b>3164</b> , 3928
manipulate signal sets .....	2055
map.....	60, 3660
mapped.....	3660
mappings.....	1442
MAP_.....	499
MAP_ANON .....	392, 1438
MAP_ANONYMOUS.....	392, 1438
MAP_FAILED .....	1443
MAP_FIXED .....	392, 1438
MAP_PRIVATE.....	392, 983, 1438, 1442, 1447, 1481
MAP_SHARED .....	392, 986, 1438-1439
margin code.....	3642
notation.....	12, 3643
matched .....	60, 179
mathematical functions .....	2459
domain error .....	109
error conditions .....	109, 3687
NaN arguments .....	110, 3687
pole error .....	109
range error .....	110
MAXARGS .....	365
maximum values.....	287
MAX_CANON .....	285, 988, 3721, 3858, 3938
MAX_INPUT .....	285, 988, 3858, 3938
may.....	6, 3640
rationale.....	3640
mblen() .....	<b>1387</b>
mbrlen().....	<b>1389</b>
mbrtoc16().....	<b>1391</b>
mbrtoc32().....	1391
mbrtowc() .....	<b>1393</b>
mbsinit().....	<b>1395</b>
mbsnrtowcs().....	1396
mbsrtowcs().....	<b>1396</b>
mbsrtowcs_s.....	502
mbstate_t .....	457
mbstowcs().....	<b>1399</b>
mbstowcs_s .....	502
mbtowc() .....	<b>1401</b>
MB_CUR_MAX.....	381, 1387, 1389, 1392-1393, 1401, 2363, 2417
MB_LEN_MAX .....	291-292
MC1 .....	<b>8</b>
MCL_.....	499
MCL_CURRENT .....	392, 1435
MCL_FUTURE .....	392, 1435, 1443, 3771

MCL_INHERIT.....	3772
mcontext_t.....	349
memccpy().....	1403
memchr().....	1404
memcmp().....	1405
memcpy().....	1406
memmem().....	1407
memmove().....	1408
memory consistency.....	102
memory locking.....	3769
memory management.....	529, 3769, 3924
memory management unit.....	3770
memory mapped files.....	60
memory object.....	60, 3660
memory ordering.....	100, 102, 3681
memory synchronization.....	104, 3681
memory-resident.....	60, 3660
memory_order.....	361
memory_order_acquire.....	361, 649, 654, 656
memory_order_acq_rel.....	361, 644, 649, 653-654, 656
memory_order_consume.....	361, 654, 656
memory_order_relaxed.....	361, 654
memory_order_release.....	361, 644, 653-654
memory_order_seq_cst.....	361, 644, 646-647, 649-650, 653-654, 656
memset().....	1409
mesg.....	3168, 3929-3930
message.....	61
message catalog.....	61
descriptor.....	61, 568, 868, 873
generation.....	2960
message passing.....	3760, 3924, 3926
message queue.....	61, 3760
messages object.....	61
MET-1MEST.....	2310
META_CHAR.....	191
minimum values.....	287
MINSIGSTKSZ.....	349, 2051
mkdir.....	3171, 3929
mkdir().....	1410, 3925
mkdirat().....	1410
mkdtemp().....	1414
mkfifo.....	3175, 3929
mkfifo().....	1417, 3655
mkfifoat().....	1417
mknod.....	
rationale for omission.....	3914
mknod().....	1421, 3655
mknodat().....	1421
mkostemp().....	1414, 1425
mkstemp().....	1414, 1426
mktime().....	1427, 3685
ML.....	8

mlock()	1433
mlockall()	1435, 3771
MLR	8
mmap()	1437, 3773-3777
MMU	3770
MM_	499
MM_ macros	261
MM_APPL	261, 971
MM_CONSOLE	261, 971
MM_ERROR	261, 972-973
MM_FIRM	261
mm_FIRM	971
MM_HALT	261, 972
MM_HARD	261, 971
MM_INFO	261, 972
MM_NOCON	262, 972
MM_NOMSG	261, 972
MM_NOSEV	261, 972
MM_NOTOK	261, 972
MM_NRECOV	261, 971
MM_NULLACT	261
MM_NULLLBL	261
MM_NULLMC	261, 971
MM_NULLSEV	261
MM_NULLTAG	261
MM_NULLTXT	261
MM_OK	261, 972
MM_OPSYS	261, 971
MM_PRINT	261, 971, 973
MM_RECOVER	261, 971
MM_SOFT	261, 971
MM_UTIL	261, 971
MM_WARNING	261, 972
mode	61
modem disconnect	3722
mode_t	425
modf()	1445
modff()	1445
modfl()	1445
monotonic clock	61, 3793
MON_	277
mon_decimal_point	148
mon_grouping	148
mon_thousands_sep	148
more	3178, 3928-3929
discard and refresh	3185
display position	3187
examine new file	3186
examine next file	3186
examine previous file	3186
go to beginning of file	3184
go to end-of-file	3184



go to tag .....	3186
help .....	3183
invoke editor .....	3187
mark position .....	3185
quit .....	3187
refresh the screen .....	3185
repeat search .....	3186
repeat search in reverse .....	3186
return to mark .....	3185
return to previous position .....	3185
scroll backward one half screenful .....	3184
scroll backward one line .....	3184
scroll backward one screenful .....	3183
scroll forward one half screenful .....	3184
scroll forward one line .....	3184
scroll forward one screenful .....	3183
search backward for pattern .....	3185
search forward for pattern .....	3185
skip forward one line .....	3184
motion command .....	3372
mount point .....	62, 3665
mounted file system .....	3660
mprotect() .....	1447, 3773
MQ .....	499
mq .....	499
mq_close() .....	1449
mq_getattr() .....	1450
mq_notify() .....	1452
mq_open() .....	1455, 3761
MQ_OPEN_MAX .....	283, 2199, 3938
MQ_PRIO_MAX .....	283, 1462-1463, 2199, 3938
mq_receive() .....	1459, 3762
mq_send() .....	1462, 3762
mq_setattr() .....	1464
mq_timedreceive() .....	1459, 1466, 3794
mq_timedsend() .....	1462, 1467, 3794
mq_unlink() .....	1468
rand48() .....	829, 1470
MSG .....	9
msg .....	499
msg*() .....	3758
msgctl() .....	1471, 3759
msgfmt .....	3191
msgget() .....	1473, 3759
msgrcv() .....	1475, 3759
msgsnd() .....	1478, 3759
MSGVERB .....	175, 972-973
MSG_ .....	499-500
msg_ .....	500
MSG_CMSG_CLOEXEC .....	409, 1881
MSG_CMSG_CLOFORK .....	409, 1881
MSG_CTRUNC .....	409

MSG_DONTRROUTE.....	409
MSG_EOR .....	409, 1965, 1968, 1972, 2091, 2094
MSG_NOERROR .....	396, 1475-1476
MSG_NOSIGNAL.....	409, 1965, 1968, 1972
MSG_OOB.....	409, 1881, 1965, 1968, 1972
MSG_PEEK .....	410, 1881
msg_perm.....	526
MSG_TRUNC .....	410
MSG_WAITALL .....	410, 1881
msgid.....	526
MST7MDT .....	2310
msync().....	<b>1481</b> , 3773
MS .....	499
MS_ASYNC.....	392, 1439, 1481
MS_INVALIDATE.....	392, 1481-1482
MS_SYNC.....	392, 1439, 1481
mtx_destroy() .....	<b>1484</b>
mtx_init() .....	1484
mtx_lock().....	<b>1486</b>
mtx_plain.....	1484
mtx_recursive .....	1484
mtx_timed .....	1484
mtx_timedlock().....	1486
mtx_trylock().....	1486
mtx_unlock() .....	1486
multi-byte character.....	3721, 3723
multi-character collating element .....	62
multi-threaded library .....	62
multi-threaded process.....	62
multi-threaded program .....	62
multicast .....	559
multiple tasks.....	3921, 3928
munlock().....	1433, 1488
munlockall() .....	1435, 1489
munmap().....	<b>1490</b> , 3773, 3775, 3777, 3780
mutex.....	62, 3811
attributes.....	1764
extended attributes .....	3815
initialization .....	3828
initialization attributes .....	1763
performance .....	1764
mv.....	<b>3198</b> , 3864, 3929
MX .....	<b>9</b>
MXC .....	<b>9</b>
MXX .....	<b>9</b>
M_1_PI.....	301
M_1_PII.....	301
M_1_SQRTPI.....	301
M_1_SQRTPII.....	301
M_2_PI.....	301
M_2_PII.....	301
M_2_SQRTPI.....	301

M_2_SQRTPII.....	301
M_E.....	301
M_EGAMMA.....	301
M_EGAMMAI.....	301
M_EI.....	301
M_LN.....	301
M_LN10.....	301
M_LN10l.....	301
M_LNl.....	301
M_LOG10E.....	301
M_LOG10El.....	301
M_LOG2E.....	301
M_LOG2El.....	301
M_PHI.....	301
M_PHIl.....	301
M_PI.....	301
M_PIl.....	301
M_PI_2.....	301
M_PI_2l.....	301
M_PI_4.....	301
M_PI_4l.....	301
M_SQRT1_2.....	301
M_SQRT1_2l.....	301
M_SQRT1_3.....	301
M_SQRT1_3l.....	301
M_SQRT2.....	301
M_SQRT2l.....	301
M_SQRT3.....	301
M_SQRT3l.....	301
name.....	63
name information.....	1146
name space.....	<b>498</b> , 3738
name space pollution.....	3737-3738
namespace.....	3728
NAME_MAX.....	105, 235-236, 285, 510, 988, 2589, 3280, 3858, 3938
NaN.....	63, 256
NAN.....	301
NaN.....	620, 622, 629, 635, 886
NAN.....	999
NaN.....	999
NAN.....	1084
NaN.....	1084, 1365, 2080, 2082, 2212, 2215
NaN arguments.....	3687
mathematical functions.....	110
nan().....	<b>1492</b>
nanf().....	1492
nanl().....	1492
nanosleep().....	<b>1494</b> , 3791, 3793-3794, 3926
native implementation.....	3661
native language.....	63
NCCS.....	439
NDEBUG.....	226, 506, 626

nearbyint()	1496
nearbyintf()	1496
nearbyintl()	1496
negative	63
negative response	63
negative_sign	148
netent	314
network	63
network address	63
network byte order	64, 99, 3681
network interfaces	550
newgrp	3204, 3930
newline character	64
newlocale()	1497
news	
rationale for omission	3914
NEW_TIME	480, 853-854
nextafter()	1500
nextafterf()	1500
nextafterl()	1500
nexttoward()	1500
nexttowardf()	1500
nexttowardl()	1500
nftw()	1502, 3864
ngettext	3208
ngettext()	1192, 1507
ngettext_l()	1192
NGROUPS_MAX	287, 1136, 2199, 3207, 3638, 3666, 3858, 3933, 3938
nice	3209, 3928
nice value	64, 3661
nice()	1508, 3784
Ninth Edition UNIX	2663, 3302
NI_DGRAM	315
NI_NAMEREQD	315
NI_NOFQDN	315
NI_NUMERICHOST	315
NI_NUMERICSCOPE	315
NI_NUMERICSERV	315
nl	3213
NLDLY	440
nlink_t	425
NLn	440
NLSPATH	171, 701
NL_	499
NL_ARGMAX	293, 995, 1037, 1081, 1091, 3297
NL_CAT_LOCALE	324, 701
nl_langinfo()	1510, 3927
nl_langinfo_l()	1510
NL_LANGMAX	293
NL_MSGMAX	293
NL_SETD	324
NL_SETMAX	293

NL_TEXTMAX .....	293
nm .....	3217, 3930
noclobber option .....	3248, 3890
NOEXPR .....	277
NOFLSH .....	442
nohup .....	875, 3222, 3928
non-blocking .....	64
non-built-in utility execution.....	2503, 3899
non-canonical mode input processing .....	202, 3722
non-local jumps .....	2069
non-printable .....	2829, 3362, 3427, 3673
non-spacing characters.....	64
non-volatile storage .....	1064
normative references .....	3639
NOSTR.....	277
rand48( ).....	829, 1513
NSIG_MAX.....	293-294
ntohl( ).....	1219, 1514
ntohs( ) .....	1219, 1514
NUL.....	64
NULL .....	367, 453, 764, 805, 819, 1442, 1860
null byte.....	65
null pointer.....	65, 3661
null string .....	65
null terminator.....	65
null wide-character code.....	65
number-sign .....	65
numerical limits.....	291
NUM_EMPL .....	1216
NZERO.....	293, 1161, 1508
n_ .....	499
n_cs_precedes .....	149
n_sep_by_space .....	149
n_sign_posn .....	149
OB .....	9
object file.....	65, 3217
obsolescent .....	3640
rationale.....	3640
OCRNL .....	440
octet .....	65
od .....	3226, 3928, 3930
OF .....	9
OFD-owned file lock.....	66
OFDEL .....	440
offset maximum.....	66
off_t .....	425
OFILL .....	440
OH .....	9
OLD_TIME.....	480, 853-854
ONCE_FLAG_INIT.....	449, 682
ONLCR .....	440
ONLRET .....	440

ONOCR .....	440
opaque address.....	66
open a file .....	1521
open a named semaphore.....	1945
open a shared memory object.....	2025
open file .....	66
open file description .....	66, 3662
open file descriptors.....	3894
for reading and writing.....	2497
open mode.....	2838
open() .....	1515, 3655, 3720, 3773, 3775-3776, 3864, 3924
openat() .....	1515, 1529
opendir() .....	920, 1530, 3925
openlog().....	753, 1531, 3930
OPEN_MAX .....	283, 343, 849, 907, 920, 1455, 1592, 1596, 2199, 2281, 3639, 3858, 3938-3939
open_memstream() .....	1526
open_wmemstream() .....	1526
operand.....	66
operator .....	66
OPOST .....	440
optarg .....	1150, 1532
opterr.....	1150, 1532
optind .....	1150, 1532
option.....	67
ADV .....	7
CD.....	7
CPT .....	7
DC.....	8
FR.....	8
FSC .....	8
IP6.....	8
MC1 .....	8
ML .....	8
MLR.....	8
MSG.....	9
MX .....	9
MXC .....	9
MXX .....	9
PIO.....	9
PS.....	9
RPI .....	10
RPP .....	10
RS.....	10
SD .....	10
SHM .....	10
SIO.....	10
SPN.....	10
SS .....	10
TCT .....	11
TPI.....	11
TPP.....	11
TPS.....	11

TSA .....	11
TSH .....	11
TSP .....	11
TSS .....	12
TYM .....	12
UP .....	12
UU .....	12
option-argument .....	67, 3662
optional behavior .....	3940
options .....	3837
shell and utilities .....	26
system interfaces .....	25
optopt .....	1150, 1153, 1532
optstring .....	1153
OR lists .....	2507, 3901
ordinary identifiers .....	2656
ORD_CHAR .....	191
orientation .....	67
orphaned process group .....	67, 571, 3662, 3753
output device .....	197, 3717
output file descriptor	
duplication .....	3893
output mode .....	3723
output processing .....	3722
O_ .....	501
O_ constants	
defined in <fcntl.h> .....	247-248
used in dbm_open() .....	802
used in open() .....	1515
used in posix_openpt() .....	1578
O_ACCMODE .....	248, 901
O_APPEND .....	248, 381, 528, 607, 917, 1414, 1515, 2436
O_CLOEXEC .....	247, 381, 393, 802, 833, 920, 1414, 1515, 1539, 1578, 1627, 3776
O_CLOFORK .....	247, 382, 393, 833, 1414, 1515, 1539, 1578, 1627
O_CREAT .....	247, 309, 343, 393, 781, 802, 1455-1456, 1515, 1935, 1943, 2023-2024, 2026
O_DIRECTORY .....	247, 920, 1516
O_DSYNC .....	248, 382, 597, 803, 1414, 1516-1517, 1853, 2437
O_EXCL .....	247, 309, 343, 393, 803, 1456, 1516, 1943, 2023-2024
O_EXEC .....	248, 1515
O_NDELAY .....	2441
O_NOCTTY .....	247, 382, 1516, 1578
O_NOFOLLOW .....	247, 1516
O_NONBLOCK .....	248, 309, 509, 902, 1456, 1516, 1539
O_RDONLY .....	248, 309, 393, 802, 812, 1455, 1515, 2023, 2026
O_RDWR .....	248, 309, 382, 393, 802, 891, 1358, 1455, 1515, 1578, 2023, 2026
O_RSYNC .....	248, 382, 803, 1414, 1517, 1853
O_SEARCH .....	248, 1332, 1410, 1417, 1422, 1515, 1518, 1864, 1900, 2194, 2320
O_SYNC .....	248, 597, 803, 1414, 1517, 1853, 2437
O_TRUNC .....	247, 393, 781, 803, 1517, 2024, 2026
O_TTY_INIT .....	199, 247, 1517
O_WRONLY .....	248, 309, 393, 781, 802, 891, 1358, 1455, 1515

pack	
rationale for omission.....	3914
page.....	67, 3662, 3773, 3776
page size.....	67
PAGESIZE.....	284, 529, 1433, 1651, 2199, 3773, 3819, 3938
PAGE_SIZE.....	284, 2199
paginators	
more.....	3178
parallel I/O.....	3819
parameter.....	68, 3723, 3874
expansion.....	2485, 3883
positional.....	3874
special.....	3874
parameters and variables.....	2478
PARENB.....	442
parent directory.....	68, 3663
parent process.....	68
parent process ID.....	68
PARMRK.....	440
PARODD.....	442
passwd	
rationale for omission.....	3914
passwd file.....	3663
paste.....	3234, 3928
patch.....	3238, 3929-3930
application.....	3241
file format.....	3240
filename determination.....	3241
PATH.....	175, 764, 876, 3706
path prefix.....	69
pathchk.....	3245, 3929
pathconf().....	988, 1533, 3655, 3857, 3923, 3925
pathname.....	68, 3663
component.....	69
expansion.....	2493, 3889
incomplete.....	3655
resolution.....	105, 2457, 3683
variable values.....	285
pathname manipulation	
basename.....	2644
dirname.....	2804
pathchk.....	3245
PATH_MAX.....	285, 294, 510, 988, 2460, 3285, 3338, 3858, 3938
pattern.....	69
filename expansion.....	3911
for filename expansion.....	2525
scanning and processing language.....	2605
pattern matching.....	2941, 3260, 3505, 3521
definition.....	2523
in case statements.....	2509
in shell variables.....	2487
multiple character.....	3910



multiple characters.....	2524
notation.....	2523, 3278, 3908
single character.....	2523, 3908
pause().....	<b>1534</b> , 3752, 3757, 3924
pax.....	<b>3250</b> , 3929-3930
archive character set encoding/decoding.....	3284
cpio file data.....	3274
cpio filename.....	3274
cpio header.....	3272
cpio interchange format.....	3271
cpio special entries.....	3274
extended header.....	3264
extended header file times.....	3267
extended header keyword precedence.....	3267
list mode format specifications.....	3258
ustar format.....	3268
ustar interchange format.....	3268
pcat	
rationale for omission.....	3915
pclose().....	<b>1535</b> , 3930
pd_.....	499
PENDIN.....	501
pending error.....	3836
per-thread errno.....	3745
performance enhancements.....	3920
period.....	69
permissions.....	69
perror().....	<b>1537</b>
persistence.....	69
PF_.....	500
pg	
rationale for omission.....	3915
physical write.....	1064
ph_.....	499
PID_MAX.....	3838
pid_t.....	425
PIO.....	<b>9</b>
pipe.....	70, 985, 1521, 2439, 3656, 3663
pipe().....	<b>1539</b> , 3753, 3758, 3924
pipe2.....	1539
pipe2().....	3924
pipeline.....	2519
pipelines.....	2504, 3899
PIPE_BUF.....	286, 988, 2437, 2440, 3858, 3938
PIPE_MAX.....	2441
plain characters.....	2129
PM_STR.....	277
pointer to a function.....	517
pole error.....	109
POLL.....	499
poll().....	<b>1542</b>
POLLERR.....	325, 1542

pollfd.....	325
POLLHUP.....	325, 1542
POLLIN.....	325, 1542
polling.....	70
POLLNVAL.....	325, 1543
POLLOUT.....	325, 1542
POLLPRI.....	325, 1542
POLLRDBAND.....	325, 1542
POLLRDNORM.....	325, 1542
POLLWRBAND.....	325, 1542
POLLWRNORM.....	325, 1542
popen().....	1547, 3927, 3930-3931
portability.....	3642
portability codes.....	3642
portable character set.....	70, 117, 3688
portable filename.....	70
portable filename character set.....	70, 3663
portable messages object source file.....	70
positional parameter.....	71, 2479, 3874
positive.....	71
positive_sign.....	148
POSIX conformance.....	15
POSIX locale.....	128, 3693
POSIX shell and utilities.....	18
POSIX system interfaces	
conformance.....	17
POSIX.1 symbols.....	496, 3737
POSIX.13.....	3777
POSIX2_BC_BASE_MAX.....	2459-2460, 3932
POSIX2_BC_DIM_MAX.....	2459-2460, 3932
POSIX2_BC_SCALE_MAX.....	2459-2460, 3932
POSIX2_BC_STRING_MAX.....	2459-2460, 3932
POSIX2_CHAR_TERM.....	19, 26, 3932
POSIX2_COLL_WEIGHTS_MAX.....	2459-2460, 3932
POSIX2_C_BIND.....	3859, 3931
POSIX2_C_DEV.....	19, 26, 3859, 3931
POSIX2_EXPR_NEST_MAX.....	2459-2460, 3932
POSIX2_FORT_RUN.....	19, 26, 3859, 3932
POSIX2_LINE_MAX.....	2459, 2461, 3932
POSIX2_LOCALEDEF.....	19, 26, 3859, 3929, 3932
POSIX2_RE_DUP_MAX.....	3932
POSIX2_SW_DEV.....	19, 26, 3859, 3931
POSIX2_SYMLINKS.....	988, 2461, 3859
POSIX2_UPE.....	19, 27, 3859, 3931-3932
POSIX2_VERSION.....	3932
POSIX.....	498
posix.....	498
POSIX_ALLOC_SIZE_MIN.....	286, 988, 3760
POSIX_ASYNCHRONOUS_IO.....	3943
POSIX_BARRIERS.....	3943
POSIX_CLOCK_SELECTION.....	3945
posix_close().....	1556

POSIX_CLOSE_RESTART .....	470
POSIX_C_LANG_ATOMICS.....	3943
POSIX_C_LANG_JUMP .....	3943
POSIX_C_LANG_MATH.....	3943
POSIX_C_LANG_SUPPORT .....	3944
POSIX_C_LANG_SUPPORT_R .....	3944
POSIX_C_LANG_THREADS.....	3944
POSIX_C_LANG_UCHAR.....	3944
POSIX_C_LANG_WIDE_CHAR .....	3944
POSIX_C_LANG_WIDE_CHAR_EXT.....	3944
POSIX_C_LIB_EXT .....	3945
posix_devctl().....	<b>1557</b>
POSIX_DEVICE_IO .....	3945
POSIX_DEVICE_IO_EXT.....	3945
POSIX_DEVICE_SPECIFIC .....	3945
POSIX_DEVICE_SPECIFIC_R.....	3945
POSIX_DYNAMIC_LINKING .....	3945
posix_fadvise().....	<b>1563</b> , 3759
POSIX_FADV_DONTNEED.....	249, 1563, 3759
POSIX_FADV_NOREUSE.....	249, 1563, 3759
POSIX_FADV_NORMAL .....	249, 1563
POSIX_FADV_RANDOM.....	249, 1563, 3759
POSIX_FADV_SEQUENTIAL .....	249, 1563, 3759
POSIX_FADV_WILLNEED .....	249, 1563, 3759
posix_fallocate().....	<b>1565</b>
POSIX_FD_MGMT.....	3945
POSIX_FIFO .....	3945
POSIX_FIFO_FD.....	3945
POSIX_FILE_ATTRIBUTES .....	3945
POSIX_FILE_ATTRIBUTES_FD.....	3945
POSIX_FILE_LOCKING .....	3945
POSIX_FILE_SYSTEM.....	3945
POSIX_FILE_SYSTEM_EXT .....	3945
POSIX_FILE_SYSTEM_FD.....	3945
POSIX_FILE_SYSTEM_GLOB.....	3946
POSIX_FILE_SYSTEM_R .....	3946
posix_getdents().....	<b>1567</b>
POSIX_I18N .....	3946
POSIX_JOB_CONTROL .....	3946
posix_madvise().....	<b>1572</b> , 3759
POSIX_MADV_DONTNEED.....	392, 1572, 3759
POSIX_MADV_NORMAL.....	392, 1572
POSIX_MADV_RANDOM.....	393, 1572, 3759
POSIX_MADV_SEQUENTIAL .....	393, 1572, 3759
POSIX_MADV_WILLNEED .....	393, 1572, 3759
POSIX_MAPPED_FILES.....	3946
posix_memalign().....	<b>1576</b>
POSIX_MEMORY_PROTECTION .....	3946
posix_mem_offset().....	<b>1574</b> , 3777-3778
POSIX_MULTI_CONCURRENT_LOCALES.....	3946
POSIX_MULTI_PROCESS .....	3946
POSIX_MULTI_PROCESS_FD.....	3946

POSIX_NETWORKING .....	3946
posix_openpt() .....	<b>1578</b>
POSIX_PIPE .....	3946
POSIX_REALTIME_SIGNALS .....	3946
POSIX_REC_INCR_XFER_SIZE .....	286, 988, 3760
POSIX_REC_MAX_XFER_SIZE .....	286, 988, 3760
POSIX_REC_MIN_XFER_SIZE .....	286, 988, 3760
POSIX_REC_XFER_ALIGN .....	286, 988, 3759
POSIX_REGEX .....	3947
POSIX_RE_DUP_MAX .....	2459, 2461
POSIX_ROBUST_MUTEXES .....	3946
POSIX_RW_LOCKS .....	3947
POSIX_SEMAPHORES .....	3947
POSIX_SHELL_FUNC .....	3947
POSIX_SIGNALS .....	3947
POSIX_SIGNALS_EXT .....	3947
POSIX_SIGNAL_JUMP .....	3947
POSIX_SINGLE_PROCESS .....	3947
posix_spawn() .....	<b>1581</b> , 3842, 3924
posix_spawnattr_destroy() .....	<b>1602</b>
posix_spawnattr_getflags() .....	<b>1604</b>
posix_spawnattr_getpgroup() .....	<b>1606</b>
posix_spawnattr_getschedparam() .....	<b>1608</b>
posix_spawnattr_getschedpolicy() .....	<b>1610</b>
posix_spawnattr_getsigdefault() .....	<b>1612</b>
posix_spawnattr_getsigmask() .....	<b>1614</b>
posix_spawnattr_init() .....	1602, 1616
posix_spawnattr_setflags() .....	1604, 1617
posix_spawnattr_setpgroup() .....	1606, 1618
posix_spawnattr_setschedparam() .....	1608, 1619
posix_spawnattr_setschedpolicy() .....	1610, 1620
posix_spawnattr_setsigdefault() .....	1612, 1621
posix_spawnattr_setsigmask() .....	1614, 1622
posix_spawnnp() .....	1581, 1623, 3842, 3924
posix_spawn_file_actions_addchdir() .....	<b>1590</b>
posix_spawn_file_actions_addclose() .....	<b>1592</b>
posix_spawn_file_actions_adddup2() .....	<b>1596</b>
posix_spawn_file_actions_addfchdir() .....	1590, 1598
posix_spawn_file_actions_addopen() .....	1592, 1599
posix_spawn_file_actions_destroy() .....	<b>1600</b>
posix_spawn_file_actions_init() .....	1600
POSIX_SPAWN_RESETIDS .....	356, 1583, 1604
POSIX_SPAWN_SETPGROUP .....	356, 1582, 1604, 1606
POSIX_SPAWN_SETSCHEDPARAM .....	356, 1604, 1608
POSIX_SPAWN_SETSCHEDULER .....	356, 1583, 1604, 1608, 1610
POSIX_SPAWN_SETSID .....	356, 1582, 1604
POSIX_SPAWN_SETSIGDEF .....	356, 1583, 1604, 1612
POSIX_SPAWN_SETSIGMASK .....	356, 1604, 1614
POSIX_SPIN_LOCKS .....	3947
POSIX_SYMBOLIC_LINKS .....	3947
POSIX_SYMBOLIC_LINKS_FD .....	3947
POSIX_SYSTEM_DATABASE .....	3947

POSIX_SYSTEM_DATABASE_R .....	3947
POSIX_THREADS_BASE.....	3947
POSIX_THREADS_EXT .....	3948
POSIX_TIMERS .....	3948
POSIX_TYPED_MEM_ALLOCATE.....	393, 1437-1438, 1574, 1624, 1626
POSIX_TYPED_MEM_ALLOCATE_CONTIG.....	393, 1437-1438, 1574, 1624, 1626
posix_typed_mem_get_info() .....	<b>1624</b> , 3777
posix_typed_mem_info.....	<b>393</b>
POSIX_TYPED_MEM_MAP_ALLOCATABLE .....	393, 1490, 1626
posix_typed_mem_open().....	<b>1626</b> , 3777
POSIX_USER_GROUPS .....	3948
POSIX_USER_GROUPS_R .....	3948
POSIX_VERSION .....	3938
POSIX_WIDE_CHAR_DEVICE_IO.....	3948
pow() .....	<b>1629</b>
powf().....	1629
powl() .....	1629
ppoll().....	1542, 1632
pr.....	<b>3290</b> , 3928, 3930
pread() .....	<b>1633</b> , 1852, 3819
preallocation .....	71
predefined stream	
standard error .....	524
standard input .....	524
standard output.....	524
preempted process (or thread).....	71
preempted thread.....	1699
previous job.....	71
PRI .....	501
print-related commands	
fold .....	2952
lp.....	3073
pr.....	3290
printable character .....	71
printable file .....	71
printf .....	<b>3296</b> , 3927-3928
printf() .....	995, 1634
printf_s.....	502
printing .....	3922
priority .....	72
inversion.....	72
scheduling.....	72
priority-based scheduling.....	72
PRIO_ .....	499
PRIO_ constants	
defined in <sys/resource.h> .....	<b>398</b>
PRIO_INHERIT .....	1741
PRIO_PGRP.....	398, 1161
PRIO_PROCESS.....	398, 1161
PRIO_USER.....	398, 1161
privilege.....	72, 3169, 3211, 3677
process .....	72

attributes.....	2453
concurrent execution .....	985
ID .....	73, 2453
ID reuse.....	106, 3685
ID, 1 .....	571
ID, rationale .....	3838
lifetime .....	73, 3664
memory locking.....	73
scheduling .....	531, 3782, 3924
setting real and effective user IDs.....	2008
single-threaded .....	985
termination.....	73, 3664
virtual time.....	74
process creation .....	985
process group.....	72, 3719
concepts in job control.....	3656
ID.....	72, 2453, 3657, 3719-3720
leader.....	72
lifetime .....	73, 3720
orphaned .....	571, 3662, 3753
termios .....	199
process group ID .....	1158, 1998, 2012, 2518
process lifetime.....	1314
process management .....	3920, 3924
process shared memory.....	1764
process status report .....	3310
process synchronization.....	1764
process termination.....	570
process-owned file lock.....	74
process-to-process communication .....	74
prof	
rationale for omission.....	3915
profiling .....	3931
program .....	74
prompting .....	3879-3880
protocol.....	74, 3835
protoent .....	314
PROT_.....	499
PROT_EXEC.....	392, 1438, 1447
PROT_NONE .....	392, 530, 1437-1438, 1447
PROT_READ.....	392, 1438, 1447
PROT_READ constants	
in <sys/mman.h> .....	392
PROT_WRITE.....	392, 1438-1439, 1442, 1447
prs.....	3304
PS .....	9
ps .....	3310, 3928, 3930
pselect() .....	1635
pseudo-random sequence generation functions .....	1849
pseudo-terminal .....	74, 3664
psiginfo().....	1641
psignal() .....	1641

PST8PDT.....	2310
ps_ .....	499
PTHREAD_ .....	499
pthread_.....	499
pthread_atfork().....	<b>1643</b>
pthread_attr_destroy().....	<b>1646</b>
pthread_attr_getdetachstate().....	<b>1649</b>
pthread_attr_getguardsize() .....	<b>1651</b> , 3819
pthread_attr_getinheritsched().....	<b>1654</b>
pthread_attr_getschedparam().....	<b>1656</b>
pthread_attr_getschedpolicy().....	<b>1658</b>
pthread_attr_getscope().....	<b>1660</b>
pthread_attr_getstack().....	<b>1662</b>
pthread_attr_getstacksize().....	<b>1665</b>
pthread_attr_init() .....	1646, 1667
pthread_attr_setdetachstate() .....	1649, 1668
pthread_attr_setguardsize().....	1651, 1669, 3819
pthread_attr_setinheritsched() .....	1654, 1670
pthread_attr_setschedparam().....	1656, 1671
pthread_attr_setschedpolicy() .....	1658, 1672
pthread_attr_setscope() .....	1660, 1673
pthread_attr_setstack() .....	1662, 1674
pthread_attr_setstacksize() .....	1665, 1675
pthread_barrierattr_destroy().....	<b>1680</b>
pthread_barrierattr_getpshared() .....	<b>1682</b>
pthread_barrierattr_init() .....	1680, 1684
pthread_barrierattr_setpshared().....	1682, 1685
pthread_barrier_destroy().....	<b>1676</b>
pthread_barrier_init() .....	1676
PTHREAD_BARRIER_SERIAL_THREAD .....	327, 1678, 3809
pthread_barrier_wait().....	<b>1678</b> , 3810, 3831
pthread_cancel() .....	<b>1686</b>
PTHREAD_CANCELED.....	327, 546, 1724
PTHREAD_CANCEL_ASYNCHRONOUS .....	327, 542, 1812
PTHREAD_CANCEL_DEFERRED .....	327, 543, 546, 869, 1696, 1812
PTHREAD_CANCEL_DISABLE .....	327, 542, 546, 1812
PTHREAD_CANCEL_ENABLE .....	327, 542, 546, 1812
PTHREAD_CANCEL_ENABLED.....	869
pthread_cleanup_pop() .....	<b>1688</b>
pthread_cleanup_push().....	1688
pthread_condattr_destroy() .....	<b>1708</b>
pthread_condattr_getclock().....	<b>1710</b>
pthread_condattr_getpshared().....	<b>1712</b>
pthread_condattr_init() .....	1708, 1714
pthread_condattr_setclock() .....	1710, 1715
pthread_condattr_setpshared() .....	1712, 1716
pthread_cond_broadcast() .....	<b>1693</b>
pthread_cond_clockwait().....	<b>1696</b> , 3794, 3816
pthread_cond_destroy() .....	<b>1703</b>
pthread_cond_init().....	1703, 3806
PTHREAD_COND_INITIALIZER .....	327, 1703
pthread_cond_signal() .....	1693, 1706

pthread_cond_timedwait()	1696, 1707, 3746, 3793, 3816, 3933
pthread_cond_wait()	1696, 1707, 3746, 3764, 3816
pthread_create()	1717, 3806-3807
PTHREAD_CREATE_DETACHED	327, 516, 1649, 3829
PTHREAD_CREATE_JOINABLE	327, 516, 869, 1649, 1732
PTHREAD_DESTRUCTOR_ITERATIONS	284, 1729, 1734, 2199, 2305, 3939
pthread_detach()	1720, 3829
pthread_equal()	1722
pthread_exit()	1723
PTHREAD_EXPLICIT_SCHED	327, 1654
pthread_getconcurrency()	3841
pthread_getcpuclockid()	1725, 3796, 3798
pthread_getschedparam()	1726
pthread_getspecific()	1729
PTHREAD_INHERIT_SCHED	327, 1654
pthread_join()	1731, 3746, 3829
PTHREAD_KEYS_MAX	284, 1734, 2200, 3939
pthread_key_create()	1734, 3809
pthread_key_delete()	1737
pthread_kill()	1739
pthread_mutexattr_destroy()	1763
pthread_mutexattr_getprioceiling()	1768
pthread_mutexattr_getprotocol()	1770
pthread_mutexattr_getpshared()	1773
pthread_mutexattr_getrobust()	1775
pthread_mutexattr_gettype()	1777, 3817
pthread_mutexattr_init()	1763, 1779
pthread_mutexattr_setprioceiling()	1768, 1780
pthread_mutexattr_setprotocol()	1770, 1781
pthread_mutexattr_setpshared()	1773, 1782
pthread_mutexattr_setrobust()	1775, 1783
pthread_mutexattr_settype()	1777, 1784, 3817
pthread_mutex_clocklock()	1741, 3794
pthread_mutex_clockrdlock()	3794
pthread_mutex_clockwrlock()	3794
pthread_mutex_consistent()	1744
PTHREAD_MUTEX_DEFAULT	327, 1756, 1777, 3816
pthread_mutex_destroy()	1746
PTHREAD_MUTEX_ERRORCHECK	327, 1752, 1756, 1777, 3816
pthread_mutex_getprioceiling()	1752
pthread_mutex_init()	1746, 1755, 3806
PTHREAD_MUTEX_INITIALIZER	327, 1746
pthread_mutex_lock()	1756, 3746, 3816, 3831
PTHREAD_MUTEX_NORMAL	327, 1756, 1777, 3816
PTHREAD_MUTEX_RECURSIVE	104, 327, 1756, 1777, 3816
PTHREAD_MUTEX_ROBUST	1775
pthread_mutex_setprioceiling()	1752, 1760
PTHREAD_MUTEX_STALLED	1775
pthread_mutex_timedlock()	1741, 1761, 3794
pthread_mutex_trylock()	1756, 1762, 3816
pthread_mutex_unlock()	1756, 1762, 3816
PTHREAD_NULL	327, 1722



pthread_once()	1785
PTHREAD_ONCE_INIT	327, 1785
PTHREAD_PRIO_INHERIT	327, 1770
PTHREAD_PRIO_NONE	327, 1752, 1770
PTHREAD_PRIO_PROTECT	327, 1757, 1770
PTHREAD_PROCESS_PRIVATE	327, 1764, 3818
PTHREAD_PROCESS_SHARED	327, 1682, 1712, 1764, 1773, 1807, 1822, 3818
pthread_rwlockattr_destroy()	1805, 3818
pthread_rwlockattr_getpshared()	1807, 3818
pthread_rwlockattr_init()	1805, 1809, 3817
pthread_rwlockattr_setpshared()	1807, 1810, 3818
pthread_rwlock_clockrdlock()	1787
pthread_rwlock_clockwrlock()	1789
pthread_rwlock_destroy()	1791
pthread_rwlock_init()	1791, 3818
PTHREAD_RWLOCK_INITIALIZER	327, 3818
pthread_rwlock_rdlock()	1794, 3818
pthread_rwlock_t	3817
pthread_rwlock_timedrdlock()	1787, 1797
pthread_rwlock_timedwrlock()	1789, 1798
pthread_rwlock_tryrdlock()	1794, 1799, 3818
pthread_rwlock_trywrlock()	1800, 3818
pthread_rwlock_unlock()	1802, 3818, 3833
pthread_rwlock_wrlock()	1800, 1804, 3818
PTHREAD_SCOPE_PROCESS	327, 540-541, 1660
PTHREAD_SCOPE_SYSTEM	327, 540-541, 1660
pthread_self()	1811, 3808
pthread_setcancelstate()	1812
pthread_setcanceltype()	1812
pthread_setconcurrency()	3841
pthread_setprio()	3827
pthread_setschedparam()	1726, 1814, 3827
pthread_setschedprio()	1815
pthread_setspecific()	1729, 1817, 3809
pthread_sigmask()	1818
pthread_spin_destroy()	1822
pthread_spin_init()	1822
pthread_spin_lock()	1824, 3811, 3831
pthread_spin_trylock()	1824, 3811
pthread_spin_unlock()	1826
PTHREAD_STACK_MIN	284, 1662, 1665, 2200, 3939
pthread_testcancel()	1812, 1828
PTHREAD_THREADS_MAX	284, 1717, 2200, 3939
PTRDIFF_MAX	373
PTRDIFF_MIN	373
ptsname()	1829
ptsname_r()	1829
public locale	3056
putc()	1831, 3821, 3925
putchar()	1833, 3925
putchar_unlocked()	1102, 1834
putc_unlocked()	1102, 1832

putenv()	1835
puts()	1837
pututxline()	853, 1839
putwc()	1840
putwchar()	1841
PWD	175
pwd	3317, 3929
pwrite()	1842, 2436, 3819
pw_	499
p_	499
P_	500
P_ALL	433, 2357
p_cs_precedes	148
P_PGID	433, 2357
P_PID	433, 2357
p_sep_by_space	148
p_sign_posn	149
qsort()	1843
qsort_r()	1843
qsort_s	502
queue a signal to a process	2068
queuing of waiting threads	3833
quick_exit()	1845
quiet NaN	256
quote removal	2493, 3889
QUOTED_CHAR	191
quoting	2472, 3866
radix character	74
RADIXCHAR	277
raise()	1846
rand()	1848, 3831
random()	1242, 1851
RAND_MAX	381, 1848
rand_r()	3841
range error	110
result overflows	110
result underflows	110
RCS	
rationale for omission	3915
RE	
bracket expression	3711
grammar	3717
read	2470, 3320, 3865, 3927
read lock	3817
read()	1852, 3656, 3720-3721, 3743, 3752-3753, 3756, 3767-3768, 3772-3773, 3819
	3830, 3839, 3925
read-only file system	75
read-write attribute	3817
read-write lock	75, 3817
readdir()	1858, 3925
readdir_r()	1858
reading data	3721

readlink.....	3325
readlink().....	1864, 3924
readlinkat().....	1864
readonly.....	2548
readv().....	1867
real group ID.....	75, 2453
real time.....	75
real user ID.....	75, 586, 1313, 2453
realloc().....	1869
reallocarray().....	1869
realpath.....	3327
realpath().....	1872, 3924
realtime.....	22
REALTIME.....	309, 913, 1433, 1435, 1449-1450, 1452, 1455, 1459, 1462, 1464, 1468, 1917-1921
.....	1923, 2023, 2028
realtime.....	3759
realtime signal delivery.....	3749
realtime signal extension.....	75
realtime signal generation.....	3749
realtime signals.....	3765
REALTIME THREADS.....	24
realtime threads.....	24
REALTIME THREADS.....	1654, 1658, 1660, 1726, 1752, 1768, 1770, 1815
record.....	75
record lock.....	76
recv().....	1875
recvfrom().....	1878
recvmsg().....	1881
red	
rationale for omission.....	3915
redirect input.....	2494, 3891
redirect output.....	2494, 3891
redirection.....	76, 2493, 3890
redirection operator.....	76
referenced shared memory object.....	76
references.....	3639
refresh.....	76
regcomp().....	1884, 3930
regerror().....	1884, 3930
regexec().....	1884, 3930
regfree().....	1884, 3930
region.....	76
register fork handlers.....	1643
REGTYPE.....	437
regular built-in.....	76, 3665
regular built-in utilities.....	76, 3665
regular expressions.....	76, 2615, 2745, 2817, 2875, 2917, 2944, 2991, 3041, 3183, 3200
.....	3213, 3257, 3335, 3356, 3543, 3602, 3709
basic.....	181
definitions.....	3709
extended.....	187
general requirements.....	3710

grammar .....	191, 3716
related to shell patterns .....	2523
regular file .....	77, 3665
REG_ .....	499
REG_ constants	
defined in <regex.h> .....	<b>336</b>
error return values of regcomp .....	1886
used in regcomp .....	1884-1885
REG_BADBR .....	337, 1886
REG_BADPAT .....	336, 1886
REG_BADRPT .....	337, 1886
REG_EBRACE .....	337, 1886
REG_EBRACK .....	337, 1886
REG_ECOLLATE .....	336, 1886
REG_ECTYPE .....	336, 1886
REG_EESCAPE .....	336, 1886
REG_EPAREN .....	337, 1886
REG_ERANGE .....	337, 1886
REG_ESPACE .....	337, 1886
REG_ESUBREG .....	337, 1886
REG_EXTENDED .....	336, 1884
REG_ICASE .....	336, 1884
REG_MINIMAL .....	336, 1884
REG_NEWLINE .....	336, 1884
REG_NOMATCH .....	336, 1886
REG_NOSUB .....	336, 1884
REG_NOTBOL .....	336, 1885
REG_NOTEOL .....	336, 1885
rejected utilities .....	3913
relational database operator .....	3026
relative pathname .....	77, 105
relocatable file .....	77
relocation .....	77
remainder() .....	<b>1892</b>
remainderf() .....	1892
remainderl() .....	1892
remove a directory .....	1910, 3343
remove a directory entry .....	2324
remove a file .....	3334
remove() .....	<b>1894</b>
remque() .....	1245, 1896
remquo() .....	<b>1897</b>
remquof() .....	1897
remquol() .....	1897
rename a file .....	1902
rename() .....	<b>1899</b> , 3925
renameat() .....	1899
renice .....	<b>3330</b> , 3928
replenishment period .....	3786
requirements .....	15
reserved words .....	2478, 3873
result overflows .....	110

result underflows .....	110
return.....	<b>2551</b>
rewind().....	<b>1905</b>
rewinddir() .....	<b>1906</b> , 3925
re_.....	499
RE_DUP_MAX .....	287, 2200, 2460, 3858
rint().....	<b>1907</b>
rintf().....	1907
rintl().....	1907
rlimit.....	<b>398</b>
RLIMIT_.....	499
RLIMIT_AS.....	399, 1177
RLIMIT_CORE.....	398, 1176
RLIMIT_CPU .....	398, 1176
RLIMIT_DATA.....	399, 1176
RLIMIT_FSIZE.....	399, 1176
RLIMIT_NOFILE .....	399, 1176, 1178
RLIMIT_STACK.....	399, 1177
rlim_.....	499
RLIM_.....	501
RLIM_INFINITY .....	398, 1176-1177
RLIM_SAVED_CUR.....	398, 1177
RLIM_SAVED_MAX.....	398, 1177
rm .....	<b>3334</b> , 3864, 3929
rm del.....	<b>3340</b>
rm dir .....	<b>3343</b> , 3929
rm dir().....	<b>1909</b> , 3745, 3925
robust mutex .....	77, 539, 1750, 3812
root directory .....	77, 2453, 3665, 3684
root file system.....	3665
root of a file system .....	3665
round robin .....	533
round() .....	<b>1912</b>
roundf().....	1912
roundl().....	1912
routing .....	550, 3836
RPI.....	<b>10</b>
RPP.....	<b>10</b>
RS.....	<b>10</b>
rsh	
rationale for omission.....	3915
RSIZE_MAX.....	502
RTLD_.....	499
RTLD_DEFAULT.....	824
RTLD_GLOBAL .....	238, 822
RTLD_LAZY .....	238, 821
RTLD_LOCAL .....	238, 822
RTLD_NEXT .....	824
RTLD_NOW.....	238, 822
RTSIG_MAX.....	284, 347, 2200, 3939
runnable process (or thread).....	77
running process (or thread).....	77

runtime values	
increasable .....	286
invariant .....	282
rusage .....	<b>398</b>
RUSAGE_ .....	499
RUSAGE_CHILDREN .....	398, 1180
RUSAGE_SELF .....	398, 1180
ru_ .....	499
R_ANCHOR .....	192
R_OK .....	464
s6_ .....	499
sact .....	<b>3346</b>
samefile() .....	3838
saved resource limits .....	78
saved set-group-ID .....	78, 2453
saved set-user-ID .....	78, 2453
SA_ .....	499
sa_ .....	499-500
SA_ macros	
declared in <signal.h> .....	349
SA_NOCLDSTOP .....	349, 516, 2042, 2047, 3657
SA_NOCLDWAIT .....	349, 1180, 2044
SA_NODEFER .....	349, 2044
SA_ONSTACK .....	349, 868, 2043
SA_RESETHAND .....	349, 2043-2044
SA_RESTART .....	349, 1638, 2043
SA_SIGINFO .....	349, 2042-2043, 2046, 2067, 3751
scalbn() .....	<b>1913</b>
scalbnf() .....	1913
scalbln() .....	1913
scalbn() .....	1913
scalbnf() .....	1913
scalbln() .....	1913
scandir() .....	614, 1915
scanf() .....	1037, 1916
scanf_s .....	502
sccs .....	<b>3349</b>
SCCS commands	
admin .....	2574
delta .....	2787
get .....	2964
prs .....	3304
rmdel .....	3340
sact .....	3346
sccs .....	3349
unget .....	3494
val .....	3523
what .....	3590
SCHAR_MAX .....	291-292
SCHAR_MIN .....	291-292
schedule alarm .....	610
scheduling .....	78

scheduling allocation domain .....	78, 3826
scheduling contention scope .....	78, 3825-3826
scheduling documentation .....	542, 3826
scheduling policy .....	79, 107, 3685
round robin .....	533
SCHED_.....	499
sched_.....	499
SCHED_FIFO .....	339, 528, 532, 541, 869, 983, 1161, 1508, 1656, 1658, 1726, 1768, 1794, 1947
.....	3926
sched_getparam() .....	1918
sched_getscheduler() .....	1919
sched_get_priority_max() .....	1917
sched_get_priority_min() .....	1917
SCHED_OTHER .....	339, 532, 535, 1161, 1658, 1726
SCHED_RR .....	339, 528, 532-533, 541, 869, 983, 1161, 1508, 1656, 1658, 1726, 1794, 1947
.....	3926
sched_rr_get_interval() .....	1920
sched_setparam() .....	1921
sched_setscheduler() .....	1923
SCHED_SPORADIC .....	339, 528, 532-533, 869, 1794, 1947, 3926
sched_yield() .....	1925
SCM_.....	500
SCM_RIGHTS .....	407
SCN .....	501
scope .....	3637
screen .....	79
scroll .....	79
SD .....	10
sdb	
rationale for omission .....	3915
sdiff	
rationale for omission .....	3915
search pattern .....	2757
seconds since the Epoch .....	107, 3685
secure_getenv() .....	1926
security considerations .....	570, 728, 1058, 1313, 1997, 3649, 3653, 3659, 3675, 3677, 3720
security, monolithic privileges .....	3649
sed .....	3354, 3928-3929
addresses .....	3356
editing commands .....	3357
regular expressions .....	3356
seed48() .....	829, 1927
seekdir() .....	1928
SEEK_.....	501
SEEK_CUR .....	247, 376, 467, 904, 1045, 1381
SEEK_DATA .....	1381
SEEK_END .....	247, 376, 467, 904, 964, 1045, 1381
SEEK_GET .....	1905
SEEK_HOLE .....	1381
SEEK_SET .....	247, 376, 467, 528, 600, 607, 904, 1045, 1381
SEGV_.....	499
SEGV_ACCERR .....	351

SEGV_MAPERR.....	351
select() .....	1635, 1930
sem .....	500
sem*().....	3758
semaphore .....	79, 108, 3686, 3763, 3926
lock operation .....	108
unlock operation .....	108
semctl().....	1955, 3758
semget().....	1958, 3758
semid.....	526
semop().....	1960, 3758
SEM_.....	499
sem_.....	499
SEM_.....	500
sem_clockwait() .....	1931, 3794
sem_close() .....	1935
sem_destroy().....	1937
SEM_FAILED.....	343, 1944-1945
sem_getvalue().....	1939
sem_init() .....	1941, 3763
SEM_NSEMS_MAX .....	284, 1941, 2200, 3939
sem_open() .....	1943, 3763
sem_perm .....	526
sem_post().....	1947
sem_timedwait() .....	1931, 1949, 3794
sem_trywait() .....	1950, 3746, 3764
SEM_UNDO.....	402, 1960
sem_unlink() .....	1952
SEM_VALUE_MAX .....	284, 1941, 1943, 2200, 3939
sem_wait() .....	1950, 1954, 3746, 3764
send().....	1965
sendmsg() .....	1968
sendto() .....	1972
sequential AND-OR list .....	2519
sequential AND-OR lists.....	2507, 3901
servent.....	314
service name .....	1022
session.....	79, 571, 1313, 1998, 2012, 2518, 3657, 3662, 3720
session leader .....	79
session lifetime .....	79
session membership.....	2453
set.....	2553, 3879
set cancelability state .....	1813
set file creation mask.....	2312
set process group ID for job control .....	1997
set-group-ID .....	570, 724, 875, 911, 2453, 2747
set-user-ID .....	570, 875, 1108, 1313, 2453, 2709, 2747
set-user-ID scripts .....	3381
SETALL.....	402, 1955
setbuf() .....	1976
setegid().....	1978
setenv().....	1979



seteuid()	1981
setgid()	1982, 3666
setgrent()	840, 1984, 3675
sethostent()	842, 1985
setitimer()	3841
setjmp()	1986, 3927
setkey()	1988
setlocale()	1990, 3927
extensions to	1991
setlogmask()	753, 1995, 3930
setnetent()	844, 1996
setpgid()	1997, 3656-3658, 3719-3720
setpgrp()	3841
setpriority()	1161, 2000, 3784
setprotoent()	846, 2001
setpwent()	848, 2002, 3675
setregid()	2003
setresgid()	2005
setresuid()	2007
setreuid()	2008
setrlimit()	1176, 2010, 3864
setservent()	851, 2011
setsid()	2012, 3719
setsockopt()	2014
setstate()	1242, 2016
setuid()	2017, 3666
setutxent()	853, 2020
SETVAL	402, 1955
setvbuf()	2021
set_constraint_handler_s	502
sh	3366, 3929, 3935
command history list	3370
command line editing	3370
vi line editing command mode	3372
vi line editing insert mode	3371
vi-mode command line editing	3371
shall	6, 3640
rationale	3640
shar	
rationale for omission	3915
shared memory	3774
shared memory object	80
shell	80
SHELL	175
shell	571, 874, 1144, 1158, 1313, 1998, 2353, 2518, 3656-3659
SHELL	3708
shell	3719, 3721, 3747, 3753
commands	2499, 3895
errors	3894
execution environment	2522, 2582, 3323, 3482, 3874, 3907
grammar	2512, 3905
grammar rules	2513, 3906

grammar, lexical conventions.....	2512, 3906
introduction .....	2472
job control.....	1313, 3656, 3747, 3753
login.....	874, 1144
variables.....	2481, 3879
shell command language .....	2472
alias substitution .....	2477
appending redirected output .....	2495
arithmetic expansion .....	2490
command substitution.....	2489
compound commands.....	2508
consequences of shell errors .....	2497
dollar-single-quotes .....	2474
double-quote.....	2473
duplicating an input file descriptor.....	2497
duplicating an output file descriptor .....	2497
escape character (backslash).....	2473
exit status and errors .....	2497
exit status for commands .....	2499
field splitting.....	2491
function definition command.....	2511
grammar .....	2512
here-document.....	2495
introduction .....	2472
lists.....	2505
open file descriptors for reading and writing.....	2497
parameter expansion .....	2485
parameters and variables.....	2478
pathname expansion.....	2493
pattern matching notation .....	2523
patterns matching a single character.....	2523
patterns matching multiple characters .....	2524
patterns used for filename expansion .....	2525
pipelines .....	2504
positional parameters .....	2479
quote removal.....	2493
quoting.....	2472
redirecting input.....	2494
redirecting output .....	2494
redirection .....	2493
reserved words.....	2478
shell commands.....	2499
shell execution environment .....	2522
shell grammar lexical conventions .....	2512
shell grammar rules .....	2513
shell variables .....	2481
signals and error handling.....	2521
simple commands .....	2500
single-quote.....	2473
special built-in utilities .....	2526
special parameter .....	2479
tilde expansion.....	2485

token recognition.....	2475
word expansions .....	2483
shell script .....	80
exec.....	874
shell, the.....	80
shift.....	<b>2561</b>
shl	
rationale for omission.....	3915
SHM .....	<b>10</b> , 500
shm .....	500
shm*() .....	3758
shmat() .....	<b>2030</b>
shmctl() .....	<b>2032</b> , 3759
shmdt() .....	<b>2034</b> , 3759
shmget() .....	<b>2036</b>
shmid .....	526
SHMLBA .....	404, 2030
shm_ .....	499
SHM_ .....	500
shm_open().....	<b>2023</b> , 3773, 3775-3776
shm_perm.....	526
SHM_RDONLY .....	404, 2030
SHM_RND .....	404, 2030
shm_unlink().....	<b>2028</b> , 3775-3776
should .....	6, 3640
rationale.....	3640
SHRT_MAX.....	236, 292
SHRT_MIN.....	292
shutdown() .....	<b>2038</b>
SHUT_ .....	500
SHUT_RD.....	410
SHUT_RDWR .....	410
SHUT_WR.....	410
sig2str().....	<b>2040</b>
SIG2STR_MAX .....	347
SIGABRT.....	347, 577, 3672, 3747
sigaction() .....	<b>2042</b> , 3749, 3751
sigaddset() .....	<b>2050</b>
SIGALRM .....	347, 610, 2085
sigaltstack().....	<b>2051</b>
SIGBUS .....	347, 351, 530, 1439, 1443, 1818, 3672, 3747
SIGCANCEL .....	1686
SIGCHLD .....	347, 351, 754, 1180, 1213, 2042, 2047, 2207, 2358, 3657, 3749, 3752-3753
SIGCLD .....	2047, 3752-3753
SIGCONT.....	347, 520, 569, 571, 1312-1313, 2841, 3180, 3657, 3749, 3752-3753
sigdelset().....	<b>2054</b>
sigemptyset().....	<b>2055</b>
SIGEMT .....	3747
SIGEV_ .....	499
sigev_ .....	499
SIGEV_NONE.....	346, 515, 529, 3749
SIGEV_SIGNAL .....	346, 515, 2268, 3749-3750

SIGEV_THREAD .....	346, 515-516, 1338, 3750
sigfillset() .....	<b>2057</b>
SIGFPE .....	347, 351, 1818, 2060, 3672, 3747, 3749
sighold() .....	3841
SIGHUP .....	347, 569, 571, 745, 2816, 2841, 3526, 3567, 3753
sigignore() .....	3841
SIGILL .....	347, 351, 1818, 2060, 3672, 3747
siginfo_t .....	<b>350</b>
SIGINT .....	347, 985, 2207, 2521, 2789, 2816, 2840, 3579, 3658, 3823, 3907
siginterrupt() .....	3841
SIGIOT .....	3747
sigismember() .....	<b>2058</b>
SIGKILL .....	347, 1313, 2042, 2046-2047, 3747, 3749, 3753
siglongjmp() .....	<b>2059</b> , 3743, 3754, 3927
signal .....	80, 513, 3665, 3837, 3907
acceptance .....	3748
actions .....	3752
concepts .....	3746
delivery .....	513, 3748
error handling .....	2521
generation .....	513, 3748
names .....	3746
realtime delivery .....	515
realtime generation .....	515
stack .....	80
signal handler .....	2060
signal processes .....	3031
signal() .....	<b>2060</b> , 3746, 3749
signaling NaN .....	256
signbit() .....	<b>2063</b>
signgam .....	1329
signgam() .....	<b>2064</b>
sigpause() .....	3841
sigpending() .....	<b>2065</b>
SIGPIPE .....	347, 899, 942, 1010, 1014, 1046, 1050, 2439, 3672, 3745
sigprocmask() .....	1818, 2066, 3748
sigqueue() .....	<b>2067</b>
SIGQUEUE_MAX .....	284, 2067, 2200
SIGQUIT .....	347, 2207, 2521, 2816, 3907
sigrelse() .....	3841
SIGRTMAX .....	347, 514, 516, 2045, 2067, 2073, 2077, 3751
SIGRTMIN .....	347, 514, 516, 2045, 2067, 2073, 2077, 3751
SIGSEGV .....	347, 351, 530, 1177, 1490, 1651, 1818, 2060, 3672, 3747
sigset() .....	3841
sigsetjmp() .....	<b>2069</b> , 3927
sigset_t .....	3746
SIGSTKSZ .....	349, 2051
SIGSTOP .....	347, 514, 2042, 2047, 3753
sigsuspend() .....	<b>2071</b> , 3752, 3757
SIGSYS .....	347, 3747
SIGTERM .....	347, 2841, 3747
sigtimedwait() .....	<b>2073</b> , 3746, 3767, 3794

SIGTRAP .....	347, 351, 3747
SIGTSTP .....	347, 514, 2887, 3658, 3753
SIGTTIN .....	347, 514, 946, 952, 1853, 3658, 3721, 3753
SIGTTOU .....	347, 514, 898, 941, 1009, 1013, 1046, 1050, 2218, 2220, 2222, 2233, 2235-2236 2238, 2240, 2438, 3657, 3721, 3753
SIGURG .....	347
SIGUSR1 .....	347, 3747
SIGUSR2 .....	347, 3747
SIGVTALRM .....	347
sigwait() .....	<b>2077</b> , 3746, 3830
sigwaitinfo() .....	2073, 2079, 3746, 3767
sigwait_multiple() .....	3748
SIGWINCH .....	347, 2841, 3180
SIGXCPU .....	347, 1176
SIGXFSZ .....	347, 1176, 2296
SIG_ .....	501
SIG_ATOMIC_MAX .....	373
SIG_ATOMIC_MIN .....	373
SIG_BLOCK .....	349, 1818
SIG_DFL .....	346, 516, 868, 1177, 2042, 2044, 2060-2061, 3748-3749, 3752
SIG_ERR .....	346, 2061
SIG_IGN .....	346, 517, 868, 875, 1180, 2042, 2060-2061, 2521, 3657, 3748-3749, 3752, 3755
SIG_SETMASK .....	349, 1818
SIG_UNBLOCK .....	349, 1818
simple commands .....	2500, 3896
command search and execution .....	2502, 3898
commands with no command name .....	2501, 3897
order of processing .....	2500, 3896
standard file descriptors .....	2503, 3899
variable assignments .....	2500, 3897
sin() .....	<b>2080</b>
sin6_ .....	499
sinf() .....	2080
single-quote .....	80, 2473, 3867
single-threaded process .....	80
single-threaded program .....	81
sinh() .....	<b>2082</b>
sinhf() .....	2082
sinhl() .....	2082
sinl() .....	2080, 2084
sin_ .....	499
SIO .....	<b>10</b>
SIOCATMARK .....	2089
sival_ .....	499
size .....	
rationale for omission .....	3915
SIZE_MAX .....	373
size_t .....	425
SI_ .....	499
si_ .....	499
SI_ASYNCIO .....	351, 518
SI_MESGQ .....	351, 518

SI_QUEUE.....	351, 518
SI_TIMER .....	351, 518
SI_USER.....	351, 518, 3751
slash.....	81
sleep .....	3385, 3927
sleep().....	2085, 3754, 3756-3757, 3924, 3926
SLR(1) grammars.....	3627
SNDBTIMEO.....	555
snprintf().....	995, 2088
snprintf_s.....	502
snwprintf_s .....	502
SO .....	500
sockaddr_in.....	318
sockaddr_in6.....	318
socketmark().....	2089
socket .....	81, 549, 3835
address.....	81
address families.....	549
addressing .....	549
asynchronous errors .....	553
connection indication queue.....	553
I/O mode.....	551, 3836
Internet Protocols .....	558, 3837
IPv4.....	558, 3837
IPv6.....	558, 3837
local UNIX connection.....	3837
local UNIX connections.....	557
options .....	554
out-of-band data.....	552
out-of-band data state.....	3836
owner .....	551, 3836
pending error .....	551
protocols .....	549
queue limit .....	3836
queue limits.....	551
receive queue .....	552, 3836
signals .....	553
types.....	550, 3836
socket() .....	2091
socketpair() .....	2094, 3758
SOCK_.....	501
SOCK_CLOEXEC.....	408, 581, 2091, 2094
SOCK_CLOFORK .....	408, 581, 2091, 2094
SOCK_DGRAM.....	408, 558, 2091, 2094
SOCK_NONBLOCK .....	408, 581, 2091, 2094
SOCK_RAW .....	408, 558
SOCK_SEQPACKET .....	408, 558, 2091, 2094
SOCK_STREAM.....	408, 558, 2091, 2094
soft limit.....	81
software development.....	3922, 3930
SOL_SOCKET .....	408
SOMAXCONN .....	409

sort .....	3388, 3928-3929
source code.....	81
SO_ACCEPTCONN.....	409, 555, 2014
SO_BROADCAST .....	409, 555
SO_DEBUG .....	409, 555
SO_DOMAIN.....	409, 555
SO_DONTROUTE .....	409, 555
SO_ERROR.....	409, 555, 2014
SO_KEEPAIVE .....	409, 555
SO_LINGER.....	409, 555
SO_OOBINLINE .....	409, 555
SO_PROTOCOL .....	409, 555
SO_RCVBUF .....	409, 555
SO_RCVLOWAT .....	409, 555
SO_RCVTIMEO.....	409, 555, 2014
SO_REUSEADDR.....	409, 555
SO_SNDBUF .....	409, 555
SO_SNDLOWAT .....	409, 555
SO_SNDTIMEO.....	409, 2014
SO_TYPE .....	409, 555, 2014
space character.....	82
sparse file.....	82
spawn.....	82
spawn example.....	3842
special built-in.....	82, 2732, 3211, 3224, 3318, 3380, 3443, 3903
special built-in utilities .....	2526, 3912
break.....	2527, 2532
characteristics.....	2526
colon.....	2530
dot.....	2534
eval .....	2536
exec.....	2538
exit.....	2541
export.....	2544
readonly.....	2548
return.....	2551
set.....	2553
shift.....	2561
times .....	2563
trap .....	2565
unset.....	2571
special characters.....	3722
special control character.....	3724
special device drivers .....	108, 3686
special parameter .....	82, 2479, 3874
special targets.....	3137
specific implementation .....	3655
SPEC_CHAR.....	192
spell	
rationale for omission.....	3915
spin lock.....	82, 3810-3811
split.....	3396, 3928

split files	
csplit .....	2753
split .....	3396
SPN .....	<b>10</b>
spoofing .....	2549
sporadic server .....	82
sporadic server policy	
execution capacity .....	533
replenishment period .....	533
scheduling .....	3786
sprintf() .....	995, 2097
sprintf_s .....	502
spurious wakeup .....	1694
sqrt() .....	<b>2098</b>
sqrtf() .....	2098
sqrtl() .....	2098
srand() .....	1848, 2100
srand48() .....	829, 2101
srandom() .....	1242, 2102
SS .....	<b>10</b>
sscanf() .....	1037, 2103
scanf_s .....	502
SSIZE_MAX .....	292, 426, 1459, 1475, 1852, 1864, 2131, 2436, 3839, 3939
ssize_t .....	425
SS_ .....	499
ss_ .....	499-500
SS_DISABLE .....	349, 2051-2052
SS_ONSTACK .....	349, 2051
SS_REPL_MAX .....	284, 3789
stack size .....	1646
stack_t .....	<b>349</b>
standard error .....	82
standard file descriptors .....	2503, 3899
standard I/O stream .....	521, 3757
standard input .....	83
standard output .....	83
standard utilities .....	83
START .....	2220
stat .....	3864
stat data structure .....	<b>414</b>
stat() .....	1055, 2104, 3652, 3773, 3864, 3924
state-dependent character encoding .....	3690
status information .....	3840
statvfs() .....	1061, 2105, 3864
stderr .....	82, 377, 2106, 3757
STDERR_FILENO .....	470, 2106
stdin .....	83, 377, 2106, 3757
STDIN_FILENO .....	470, 1547, 2106
stdio	
locking functions .....	957
with explicit client locking .....	1102
stdout .....	83, 377, 2106, 3757



STDOUT_FILENO .....	470, 1547, 2106
STOP .....	2220
stpcpy().....	<b>2108</b> , 2119
stpncpy() .....	<b>2109</b> , 2152
str2sig().....	2040, 2110
strcasecmp().....	<b>2111</b>
strcasecmp_l().....	2111
strcat().....	<b>2113</b>
strchr() .....	<b>2114</b>
strcmp() .....	<b>2115</b>
strcoll().....	<b>2117</b>
strcoll_l().....	2117
strcpy() .....	<b>2119</b>
strcspn().....	<b>2122</b>
strdup() .....	<b>2123</b>
stream.....	83
byte-oriented.....	524
interaction with file descriptors .....	522
stream orientation .....	524
wide-oriented.....	524
STREAMS.....	3840
STREAM_MAX .....	284, 917, 978, 1548, 2200, 2280, 3939
strerror() .....	<b>2125</b>
strerror_l().....	2125
strerror_r() .....	2125
strfmon() .....	<b>2129</b>
strfmon_l() .....	2129
strftime() .....	<b>2134</b>
strftime_l() .....	2134
string .....	83
strings.....	<b>3400</b> , 3930
strip .....	<b>3403</b> , 3930
strlcat() .....	<b>2145</b>
strlcpy() .....	2145
strlen() .....	<b>2147</b>
strncasecmp().....	2111, 2149
strncasecmp_l().....	2111, 2149
strncat() .....	<b>2150</b>
strncmp().....	<b>2151</b>
strncpy() .....	<b>2152</b>
strndup() .....	2123, 2154
strnlen() .....	<b>2155</b>
strpbrk() .....	<b>2156</b>
strptime() .....	<b>2157</b>
strrchr() .....	<b>2165</b>
strsignal() .....	<b>2167</b>
strspn() .....	<b>2169</b>
strstr() .....	<b>2170</b>
strtod().....	<b>2171</b>
strtof().....	2171
strtoimax() .....	<b>2175</b>
strtok() .....	<b>2177</b>

strtok_r()	2177
strtol()	<b>2180</b>
strtoldd()	2171, 2183
strtoll()	2180, 2184
strtoul()	<b>2185</b>
strtoull()	2185
strtoumax()	2175, 2188
structures, additions to	3738
strxfrm()	<b>2189</b>
strxfrm_l()	2189
stty	<b>3405</b> , 3930
combination modes	3410
control modes	3405
informational queries	3411
input modes	3406
local modes	3408
output modes	3407
special control character assignments	3409
terminal window size	3411
ST_	500
st_	500
st_gid	2589
st_mode	2589
st_mtime	2589
ST_NOSUID	420, 868, 1061
ST_RDONLY	420, 1061
st_size	2589
st_uid	2589
su	
rationale for omission	3915
subprofiling	20, 3645
subprofiling option groups	3943
subshell	84, 2519, 3658
successfully completed	3673
successfully transferred	84
sum	
rationale for omission	3916
sun_	500
superuser	586, 728, 1335, 2324, 2926, 3086, 3277, 3649, 3666, 3677, 3879, 3913
supplementary group ID	84, 2453, 3666
supplementary groups	728, 1135, 3677
supported threads functions	3813
suseconds_t	425
suspended job	84, 2519
SVID	2069
SVR4	1441, 1494
sv_	499
SV_	501
swab()	<b>2191</b>
swprintf()	1081, 2192
swprintf_s	502
swscanf()	1091, 2193

swscanf_s.....	502
SWTCH.....	501
symbolic constant.....	84, 3641, 3667
symbolic link.....	58, 85, 1332, 3051, 3668
symbolic name.....	3641
symbols.....	3737
POSIX.1.....	496
symlink().....	<b>2194</b>
symlinkat().....	2194
SYMLINK_MAX.....	286, 294, 988, 2195
SYMLOOP_MAX.....	284, 2200, 3743
SYMTYPE.....	437
sync().....	<b>2198</b>
synchronization operation.....	85
synchronized I/O.....	3767, 3925
completion.....	85
data integrity completion.....	85, 3673, 3767
file integrity completion.....	85, 3673, 3767
synchronized I/O data integrity completion.....	3672
synchronized I/O operation.....	86
synchronized input and output.....	85
synchronous I/O operation.....	86
synchronously accept a signal.....	2074
synchronously-generated signal.....	86, 3672
sysconf().....	<b>2199</b> , 3655, 3770, 3773-3774, 3822, 3857, 3923, 3925
syslog().....	753, 2206, 3930
system.....	86
boot.....	86
call.....	3673
clock.....	86
configuration values.....	2973
console.....	86, 3673
crash.....	86, 1064
database.....	3673
databases.....	87
documentation.....	87
name.....	2314, 3487
process.....	87, 3673
reboot.....	87, 3673
system documentation.....	3640
system environment.....	3922, 3930
System III.....	728, 2314, 3663, 3838
system interfaces.....	565, 3840
System V.....	571, 610, 728, 876, 910, 991, 1158, 1313, 1412, 1910, 2012, 2047, 2069, 2224
.....	2314, 3652, 3659, 3747
system().....	<b>2207</b> , 3927, 3930-3931
system-wide.....	87
s.....	499
S.....	501
S_ constants	
defined in <sys/stat.h>.....	<b>415</b>

S_ macros	
defined in <sys/stat.h> .....	415
S_IFBLK .....	415, 1421
S_IFCHR .....	415, 1421
S_IFDIR .....	415, 1421
S_IFIFO .....	415, 1421
S_IFLNK .....	415
S_IFMT .....	415
S_IFREG .....	415, 1421
S_IFSOCK .....	415
S_IRGRP .....	892, 1052, 1055, 1421
S_IROTH .....	892, 1052, 1055, 1421
S_IRUSR .....	892, 1052, 1055, 1421
S_IRWXG .....	1421
S_IRWXO .....	1421
S_IRWXU .....	1421
S_ISBLK .....	415
S_ISCHR .....	416
S_ISDIR .....	416
S_ISFIFO .....	416
S_ISGID .....	417, 721, 724, 1421, 2296, 2437
S_ISLNK .....	416
S_ISREG .....	416
S_ISSOCK .....	416
S_ISUID .....	417, 721, 724, 1421, 2296, 2437
S_ISVTX .....	721, 1421
S_IWGRP .....	892, 1052, 1055, 1421
S_IWOTH .....	892, 1052, 1055, 1421
S_IWUSR .....	892, 1052, 1055, 1421
S_IXGRP .....	1421
S_IXOTH .....	1421
S_IXUSR .....	1421
S_TYPEISMQ .....	416
S_TYPEISSEM .....	416
S_TYPEISSHM .....	416
S_TYPEISTMO .....	416
tab character .....	87
TABDLY .....	441
TABn .....	441
tabs .....	3415, 3928
TABSIZE .....	670, 1379
tag file creation .....	2757
tail .....	3419, 3928
talk .....	3424, 3929
tan() .....	2212
tanf() .....	2212
tanh() .....	2215
tanhf() .....	2215
tanh1() .....	2215
tanl() .....	2212, 2217
tar	
rationale for omission .....	3916

tar format.....	3268
target rule .....	3130
tcdrain().....	<b>2218</b>
tcflow().....	<b>2220</b>
tcflush().....	<b>2222</b>
tcgetattr().....	<b>2224</b> , 3658
tcgetpgrp().....	<b>2226</b> , 3658, 3719-3720
tcgetsid().....	<b>2228</b>
tcgetwinsize().....	<b>2229</b>
TCIFLUSH.....	443, <b>2222</b>
TCIOFF.....	443, <b>2220</b>
TCIOFLUSH.....	443, <b>2222</b>
TCION.....	443, <b>2220</b>
TCOFLUSH.....	<b>2222</b>
TCOOFF.....	443, <b>2220</b>
TCOON.....	443, <b>2220</b>
TCP_.....	499
TCP_NODELAY.....	323
TCSADRAIN.....	443, <b>2235</b>
TCSAFLUSH.....	443, <b>2235</b>
TCSANOW.....	443, <b>2235</b>
tcsendbreak().....	<b>2233</b>
tcsetattr().....	<b>2235</b> , 3658, 3718
tcsetpgrp().....	<b>2238</b> , 3657-3658
tcsetwinsize().....	<b>2240</b>
TCT.....	<b>11</b>
tdelete().....	<b>2242</b>
tee.....	<b>3428</b> , 3927
telldir().....	<b>2247</b>
tempnam().....	3842
TERM.....	<b>175</b>
terminal.....	87
access control.....	2224, 2236, 3720
controlling.....	200
device file.....	3719
device file, closing.....	3722
device name.....	2307
type.....	197, 3717
terminal characteristics	
stty.....	3405
tabs.....	3415
tput.....	3456
tty.....	3471
terminal device.....	87
terminate a process.....	570, 3031
terminology.....	3639
termios.....	199
canonical mode input processing.....	202
control modes.....	209
controlling terminal.....	200
input modes.....	206
local modes.....	210

non-canonical mode input processing .....	202
output modes .....	207
process group .....	199
special control characters .....	212
termios structure .....	2224, 3723
test .....	<b>3431</b> , 3927, 3929
TeX .....	3929
text column .....	87
text domain .....	88
text file .....	88, 3673
TEXTDOMAIN .....	<b>173</b>
textdomain() .....	666, 2248
TEXTDOMAINDIR .....	<b>173</b>
TEXTDOMAINMAX .....	499
TEXTDOMAIN_MAX .....	286, 988
tfind() .....	2242, 2249
tgamma() .....	<b>2250</b>
tgammaf() .....	2250
tgammaL() .....	2250
TGEXEC .....	437
TGREAD .....	437
TGWRITE .....	437
THOUSEP .....	277
thrd_busy .....	1487
thrd_create() .....	<b>2253</b>
thrd_current() .....	<b>2255</b>
thrd_detach() .....	<b>2256</b>
thrd_equal() .....	<b>2257</b>
thrd_error .....	757, 759, 762, 1484, 1487, 2253, 2256, 2260, 2301, 2305
thrd_exit() .....	<b>2258</b>
thrd_join() .....	<b>2260</b>
thrd_nomem .....	759, 2253
thrd_sleep() .....	<b>2262</b>
thrd_success .....	757, 759, 762, 1484, 1487, 2253, 2256, 2260, 2301, 2305
thrd_timedout .....	762, 1487
thrd_yield() .....	<b>2264</b>
thread .....	88, 3674
thread cancelability	
states .....	3830
type .....	3830
thread cancellation .....	3829-3830
cleanup handlers .....	546
thread cancellation points .....	3830
thread concurrency level .....	3818
thread creation .....	1718
thread creation attributes .....	1646, 3806
thread ID .....	88, 538, 1722, 3674, 3823
thread interactions .....	3835
Thread Lifetime .....	89
thread list .....	89
thread mutex .....	539, 3824
thread read-write lock .....	3833

thread scheduling.....	540, 3824
thread stack guard size.....	3819
thread termination .....	89, 1723
thread-safe.....	3674
thread-safety .....	89, 108, 537, 957, 3687, 3820
rationale.....	3687
thread-specific data.....	3808
thread-specific data key .....	89
creation .....	1735
deletion .....	1737
thread-specific data management.....	1729
threads .....	537, 3806
extensions.....	3815
file operations .....	547
implementation models .....	3807
thread_local.....	449
tilde.....	90
tilde expansion.....	2485, 3881
time.....	3440, 3927, 3930
time() .....	2265, 3743
timeout.....	3445
timeouts.....	90, 3798
timer .....	90
ID .....	2270
overflow .....	90
timers .....	3789
TIMER_ .....	500
timer_ .....	500
TIMER_ABSTIME.....	453, 536, 739, 2272, 3790-3792
timer_create() .....	2268
timer_delete() .....	2271
timer_getoverrun().....	2272
timer_gettime() .....	2272
TIMER_MAX .....	285, 2200, 3939
timer_settime().....	2272, 3790-3792
timer_t.....	425
times.....	2563
times().....	2275, 3743, 3797, 3924
timespec.....	452, 3686
timespec_get() .....	2278
timeval .....	400, 422
timezone().....	2279
time_t .....	425, 3685
TIME_UTC .....	453
tm.....	452
TMAGIC .....	437
TMAGLEN.....	437
TMPDIR.....	175, 3254
tmpfile().....	2280
tmpfile_s .....	502
tmpnam() .....	2283
tmpnam_s.....	502

TMP_MAX	377, 2281, 2283
TMP_MAX_S	502
tms	424
tms_	500
tm_	500
toascii()	3841
TOEXEC	437
token	90
token recognition	2475, 3871
tolower()	2285
tolower_l()	2285
TOREAD	437
TOSTOP	442, 898, 941, 1009, 1013, 1046, 1050, 2438, 3657
touch	3450, 3864, 3929
toupper()	2287
toupper_l()	2287
towctrans()	2289
towctrans_l()	2289
towlower()	2291
towlower_l()	2291
TOWRITE	437
towupper()	2293
towupper_l()	2293
TPI	11
TPP	11
TPS	11
tput	3456, 3930
tr	3459, 3928-3929
Tracing	3840
trap	2565, 3907
TRAP_	499
TRAP_BRKPT	351
TRAP_TRACE	351
troff	3929
trojan horse	3086, 3649
true	3466, 3927
trunc()	2295
truncate()	2296
truncf()	2295, 2299
truncl()	2295, 2299
TSA	11
tsearch()	2242, 2300
TSGID	437
TSH	11
tsort	3468
TSP	11
TSS	12
tss_create()	2301
tss_delete()	2303
TSS_DTOR_ITERATIONS	449
tss_get()	2305
tss_set()	2305



TSUID .....	437
TSVTX .....	437
tty .....	<b>3471</b> , 3930
ttynam e() .....	<b>2307</b> , 3820
ttynam e_r() .....	2307
TTY_NAME_MAX .....	285, 2200, 2307, 3939
TUEXEC .....	437
TUREAD .....	437
TUWRITE .....	437
TVERSION .....	437
TVERSLEN .....	437
tv_ .....	500
twalk() .....	2242, 2309
TYM .....	<b>12</b>
type .....	2470, 3473, 3865
typed memory .....	3777
name space .....	90
object .....	90
pool .....	90
port .....	91
TZ .....	<b>175</b> , 3708
tzname .....	2310
TZNAME_MAX .....	285, 2200, 3939
tzset .....	2310
tzset() .....	<b>2310</b> , 3927
T_FMT .....	277
T_FMT_AMP M .....	277
ualarm() .....	3924
UCHAR_MAX .....	291-292
ucontext_t .....	<b>349</b>
uc_ .....	499
UID_MAX .....	3839
uid_t .....	425, 3675
UINT .....	501
UINTMAX_MAX .....	373
UINTN_MAX .....	372
UINTPTR_MAX .....	373
UINT_FASTN_MAX .....	373
UINT_LEASTN_MAX .....	372
UINT_MAX .....	292, 610, 2086
UO_MAXIOV .....	500
ulimit .....	2470, 3476, 3865
ulimit() .....	3842
ULLONG_MAX .....	292, 2186
ULONG_MAX .....	292, 2186, 2410, 3857
umask .....	2470, 3480, 3865, 3930
umask() .....	<b>2312</b> , 3924
unalias .....	2470, 3484, 3865, 3928
uname .....	<b>3487</b> , 3930
uname() .....	<b>2314</b> , 3923
unary primaries .....	3433
unbind .....	91

unbounded priority inversion.....	3828
uncompress.....	2735, 3490
undefined.....	6, 3640
rationale.....	3640
underlying function.....	524
unexpand.....	3491, 3928
unget.....	3494
ungetc().....	2316
ungetwc().....	2318
unicast.....	559
uniq.....	3497, 3928-3929
unlink.....	3502
unlink().....	2320, 3745, 3773, 3775-3776, 3925
unlinkat().....	2320
unlockpt().....	2326
unpack	
rationale for omission.....	3916
unsafe functions.....	3754
unset.....	2571
unsetenv().....	2327
unspecified.....	6, 3640
rationale.....	3640
until loop.....	2511, 3903
UP.....	12
upshifting.....	91
uselocale().....	2328
user database.....	91, 3675
access.....	3675
user ID.....	91, 3007
logname.....	3071
newgrp.....	3204
real and effective.....	2008
setting real and effective.....	2008
who.....	3593
user name.....	91
user requirements.....	3919
USER_PROCESS.....	480, 853-854
USHRT_MAX.....	292
usleep().....	3924
ustar format.....	3268
UTC.....	2310
utility.....	92, 111, 3687
argument syntax.....	3724
conventions.....	3724
description defaults.....	3860
limits.....	3857
option parsing.....	2979
syntax guidelines.....	215, 3725
utime().....	3842
utimensat().....	1075, 2330, 3924
utimes().....	1075, 2330
UTIME_NOW.....	416, 1075

UTIME_OMIT.....	416, 1075
utmpx.....	<b>480</b>
uts_.....	500
ut_.....	500
UU.....	<b>12</b>
uucp.....	<b>3504</b> , 3929
uudecode.....	<b>3508</b> , 3928-3929
uuencode.....	<b>3511</b> , 3928-3929
uustat.....	<b>3516</b>
uux.....	<b>3519</b>
val.....	<b>3523</b>
variable.....	92, 3874
variable assignment.....	111, 3687
variable assignments.....	2500, 3897
vasprintf().....	<b>2332-2333</b>
va_arg().....	<b>2331</b>
va_copy().....	2331
va_end().....	2331
va_start().....	2331
VDISCARD.....	501
vdprintf().....	2333
VDSUSP.....	501
VEOF.....	439, 3724
VEOL.....	439, 3724
VERASE.....	439
Version 7.....	610, 1313, 2314, 3684, 3866
vertical-tab character.....	92
vfprintf().....	<b>2333</b>
vfprintf_s.....	502
VFS.....	420
vfscanf().....	<b>2335</b>
vfscanf_s.....	502
vfwprintf().....	<b>2336</b>
vfwprintf_s.....	502
vfwscanf().....	<b>2338</b>
vfwscanf_s.....	502
vhangup().....	3659
vi.....	<b>3526</b> , 3928-3929
<ESC>.....	3566
append.....	3547
change.....	3548
change to end-of-line.....	3549
clear and redisplay.....	3534
command descriptions.....	3527
control-D.....	3563
control-H.....	3563
control-T.....	3565
control-U.....	3565
control-V.....	3565
current and line above.....	3541
delete.....	3549
delete character.....	3559

delete to end-of-line .....	3550
display information .....	3533
edit the alternate file .....	3535
enter ex mode .....	3555
execute .....	3546
execute an ex command .....	3545
exit .....	3561
find character .....	3550-3551
find regular expression .....	3543
Initialization .....	3527
input mode commands .....	3561
insert .....	3552
insert empty line .....	3554
join .....	3552
mark position .....	3553
move back .....	3541-3542, 3547-3548
move cursor .....	3533, 3536-3537, 3556-3557
move down .....	3534
move forward .....	3541-3542
move to bigword .....	3550, 3558
move to bottom of screen .....	3552
move to first character in line .....	3545
move to first non-<blank> .....	3541
move to line .....	3551
move to matching character .....	3538
move to middle of screen .....	3553
move to next section .....	3540
move to specific column .....	3542
move to top of screen .....	3551
move to word .....	3550, 3558
move up .....	3534
newline .....	3564
nul .....	3563, 3566
page backwards .....	3532
page forward .....	3533
put from buffer .....	3554-3555
redraw screen .....	3535
redraw window .....	3560
regular expression .....	3546
repeat .....	3543
repeat find .....	3545, 3553
repeat substitution .....	3539
replace character .....	3555-3556
replace text with command .....	3537
return to previous context .....	3539
return to previous section .....	3540
reverse case .....	3547
reverse find character .....	3542
scroll backward .....	3535
scroll backward by line .....	3535
scroll forward .....	3532
scroll forward by line .....	3532

search for tagstring .....	3536
shift left .....	3545
shift right .....	3546
substitute character .....	3556
substitute lines .....	3556
terminate command or input mode .....	3536
undo .....	3557
undo current line .....	3558
yank .....	3559
yank current line .....	3560
VINTR .....	439
virtual processor .....	3675
VISIT .....	2242, 2309
visual mode .....	2838
VKILL .....	439
VLNEXT .....	501
VMIN .....	3724
vprintf() .....	2333, 2339
vprintf_s .....	502
VQUIT .....	439
VREPRINT .....	501
vscanf() .....	2335, 2340
vscanf_s .....	502
vsnprintf() .....	2333, 2341
vsnprintf_s .....	502
vsnwprintf_s .....	502
vsprintf() .....	2333, 2341
vsprintf_s .....	502
vsscanf() .....	2335, 2342
vsscanf_s .....	502
VSTART .....	439
VSTATUS .....	501
VSTOP .....	439
VSUSP .....	439
vswprintf() .....	2336, 2343
vswprintf_s .....	502
vswscanf() .....	2338, 2344
vswscanf_s .....	502
VTDLY .....	441
VTIME .....	3724
VTn .....	441
VWERASE .....	501
vwprintf() .....	2336, 2345
vwprintf_s .....	502
vwscanf() .....	2338, 2346
vwscanf_s .....	502
wait .....	2470, 3581, 3865, 3927
for process termination .....	2353
for thread termination .....	1732
wait() .....	2347, 3743, 3747, 3752, 3754, 3757, 3924
waitid() .....	2357, 3754, 3840, 3924
waiting on a condition .....	1698

waitpid()	2347, 2360, 3657, 3662, 3754, 3838, 3924
wall	
rationale for omission	3916
WARNING	972
warning	
OB	9
OF	9
wc	3586, 3928
WCHAR_MAX	374, 482
WCHAR_MIN	373, 482
WCONTINUED	433, 2347, 2357
WCOREDUMP	381, 433, 2348
wpcpy()	2361, 2372
wpnncpy()	2362, 2385
wrtomb()	2363
wrtomb_s	502
wscasecmp()	2365
wscasecmp_l()	2365
wscat()	2367
wchr()	2368
wscmp()	2369
wscoll()	2370
wscoll_l()	2370
wscopy()	2372
wscspn()	2374
wcsdup()	2375
wcsftime()	2376
wslcat()	2378
wslcpy()	2378
wcslen()	2380
wcncasecmp()	2365, 2382
wcncasecmp_l()	2365, 2382
wcncat()	2383
wcncmp()	2384
wcncpy()	2385
wcslen()	2380, 2387
wcsnrtombs()	2388, 2391
wcspbrk()	2389
wcsrchr()	2390
wcsrtombs()	2391
wcsspn()	2393
wcstr()	2394
wcstod()	2395
wcstof()	2395
wcstoimax()	2399
wcstok()	2400
wcstol()	2402
wcstold()	2395, 2405
wcstoll()	2402, 2406
wcstombs()	2407
wcstombs_s	502
wcstoul()	2409

wcstoull()	2409
wcstoumax()	2399, 2412
wcswidth()	2413
wcsxfrm()	2414
wcsxfrm_l()	2414
wctob()	2416
wctomb()	2417
wctomb_s	502
wctrans()	2419
wctrans_l()	2419
wctype()	2421
wctype_l()	2421
wcwidth()	2423
WEOF	482, 486, 563, 1280, 1282, 1286, 1288, 1291, 1293, 1295, 1297, 1299, 1301, 1303 1305, 2291, 2293, 2318
WERASE	3721
WEXITED	433, 2357
WEXITSTATUS	381, 433, 2348
we_	500
what	3590
while loop	2510, 3903
white space	92, 3676
white-space byte	92
white-space character	92
white-space wide character	92
who	3593, 3929-3930
wide characters	120
wide-character code	3689
wide-character code (C language)	93
wide-character input/output functions	93
wide-character string	93
wide-oriented stream	524
WIFCONTINUED	433, 2348
WIFEXITED	381, 433, 2348
WIFSIGNALED	381, 433, 2348
WIFSTOPPED	381, 433, 2348, 2354
WINT_MAX	374
WINT_MIN	374
wmemchr()	2424
wmemcmp()	2425
wmemcpy()	2426
wmemcpy_s	502
wmemmove()	2427
wmemmove_s	502
wmemset()	2429
WNOHANG	381, 433, 2047, 2347, 2357
WNOWAIT	433, 2357, 3840
word	93
word counting	3586
word expansions	2483, 3880
wordexp()	2430, 3930
wordfree()	2430, 3930

WORD_BIT .....	291, 293
working directory .....	93
worldwide portability interface .....	93
wprintf() .....	1081, 2435
wprintf_s .....	502
WRDE_ .....	500
WRDE_APPEND .....	488, 2431
WRDE_BADCHAR .....	488, 2432
WRDE_BADVAL .....	488, 2432
WRDE_CMDSUB .....	488, 2432
WRDE_DOOFFS .....	488, 2431
WRDE_NOCMD .....	488, 2431
WRDE_NOSPACE .....	488, 2432
WRDE_REUSE .....	488, 2431
WRDE_SHOWERR .....	488, 2431
WRDE_SYNTAX .....	488, 2432
WRDE_UNDEF .....	488, 2431
write .....	93, 3597, 3928-3929
write lock .....	3817
write to a file .....	2439
write() .....	<b>2436</b> , 3656-3657, 3720-3721, 3743, 3752-3753, 3756, 3767-3768, 3772-3773 3839, 3925
writev() .....	<b>2444</b>
writing data .....	3722
wscanf() .....	1091, 2446
wscanf_s .....	502
WSTOPPED .....	433, 2357
WSTOPSIG .....	381, 433, 2348
ws_ .....	500
WTERMSIG .....	381, 433, 2348
WUNTRACED .....	381, 433, 2348, 2353, 3657
W_OK .....	464
xargs .....	<b>3600</b> , 3927
xgettext .....	<b>3608</b>
XOPEN_UNIX .....	19, 27
XOPEN_UUCP .....	19, 27
XSI .....	<b>12</b> , 93, 3676
conformance .....	15, 19, 94
XSI interprocess communication .....	<b>526</b>
XSI IPC .....	3758
XSI option groups .....	22, 3646
XSI system interfaces	
conformance .....	19
XSI_C_LANG_SUPPORT .....	3948
XSI_DBM .....	3948
XSI_DEVICE_IO .....	3948
XSI_DEVICE_SPECIFIC .....	3948
XSI_FILE_SYSTEM .....	3948
XSI_GENERAL_TERMINAL_R .....	3948
XSI_IPC .....	3948
XSI_MATH .....	3948
XSI_MULTI_PROCESS .....	3948



XSI_SIGNALS .....	3948
XSI_SINGLE_PROCESS .....	3949
XSI_SYSTEM_DATABASE .....	3949
XSI_SYSTEM_LOGGING .....	3949
XSI_USER_GROUPS .....	3949
XSI_WIDE_CHAR .....	3949
X_OK .....	464, 587
y0() .....	2447
y1() .....	2447
yacc .....	3613, 3930, 3932
algorithms .....	3625
code file .....	3615
completing the program .....	3624
conflicts .....	3622
debugging the parser .....	3625
declarations section .....	3616
description file .....	3615
error handling .....	3623
grammar rules .....	3618
header file .....	3615
input grammar .....	3620
input language .....	3616
interface to the lexical analyzer .....	3624
lexical structure of the grammar .....	3616
library .....	3624
limits .....	3625
programs section .....	3620
YESEXPR .....	277
yn() .....	2447
zcat .....	2735, 3631
zombie process .....	94
zombie thread .....	94





# RAISING THE WORLD'S STANDARDS

---

Connect with us on:



Facebook: [facebook.com/ieeesa](https://facebook.com/ieeesa)



LinkedIn: [linkedin.com/groups/1791118](https://linkedin.com/groups/1791118)



Beyond Standards blog: [beyondstandards.ieee.org](https://beyondstandards.ieee.org)



YouTube: [youtube.com/ieeesa](https://youtube.com/ieeesa)

[standards.ieee.org](https://standards.ieee.org)

Phone: +1 732 981 0060